



**Universidad Rey Juan Carlos**

**Escuela Técnica Superior de Ingeniería Informática**

*Departamento de Ciencias de la Computación, Arquitectura de la  
Computación, Lenguajes y Sistemas Informáticos y Estadística e  
Investigación Operativa*

## **An ADM-Based Method for Migrating CMS-Based Web Applications**

**Doctoral Thesis**  
by Feliu Trias Nicolau

**Thesis Supervisor:** Valeria de Castro Martínez  
**Thesis Co-Advisor:** Marcos López Sanz

July 2014





La Dra. Da. Valeria de Castro Martínez y el Dr. D. Marcos López Sanz, ambos profesores del Departamento de Ciencias de la Computación, Arquitectura de la Computación, Lenguajes y Sistemas Informáticos y Estadística e Investigación Operativa de la Universidad Rey Juan Carlos, directora y co-director, respectivamente, de la Tesis Doctoral: “An ADM-based Method for Migrating CMS-based Web Applications” realizada por el doctorando Feliu Trias Nicolau,

HACEN CONSTAR QUE:

esta tesis doctoral reúne los requisitos para su defensa y aprobación

En Madrid, a 21 de julio de 2014

Fdo.: Valeria de Castro Martínez

Fdo.: Marcos López Sanz



I didn't know it was impossible when I did it.

(Albert Einstein)

Genius is one percent inspiration, ninety nine percent perspiration.

(Thomas Edison)

*-A mon pare i ma mare,  
i a tots aquells que m'estimen-*



## Abstract

In the last decade, the volume of digital content managed by Web applications has grown dramatically as well as the processes supported by these Web applications have become more complex. Thus, organizations have experienced the necessity of relying on powerful platforms to maintain their large-scale Web applications and to manage all their content in a robust and reliable manner. One of the most adopted solutions has been to base Web applications on *Content Management Systems (CMS)*. **CMS-based Web applications** provide organizations with some additional features such as, dynamic creation of content, separation between content and design, definition of different levels of access authority and flexible extension of their functionality.

Currently, a considerable number of different CMS platforms can be found on the market. This fact, along with the changes in the objectives of organizations, can cause organizations to see the necessity of migrating their CMS-based Web applications to other CMS platforms meeting better their needs. This migration process entails a complex, time-consuming and error-prone reengineering process.

The **Architecture-Driven Modernization (ADM)** is considered one of the most effective approaches to systemize the migration process and to mitigate the drawbacks mentioned. ADM advocates for the application of *Model-Driven Architecture (MDA)* techniques and tools (models as key artefacts and automated transformations) for the migration process. Furthermore, ADM proposes a set of standard metamodels to represent the information involved in this process. Two of these metamodels are: the *Abstract Syntax Tree Metamodel (ASTM)* represents at platform-specific level the syntax of the code implementing a legacy system, and the *Knowledge Discovery Metamodel (KDM)* represents at platform-independent level the semantics of this code.

We have carried out a literature review whose results demonstrate that there is not any model-driven reengineering method to shift from one CMS platform to another while maintaining the same features of the Web application.

To solve this gap, we propose the *ADMigraCMS* method, an **ADM-based method that defines standard guidelines to migrate CMS-based Web applications to other CMS platforms**. This method is supported by a toolkit that allows systemizing the migration process.

The *ADMigraCMS* method is composed of three reengineering stages defining a “horseshoe” process: *reverse engineering stage*, *restructuring stage* and *forward engineering stage*. During the *reverse engineering stage*, the knowledge from the code that implements the legacy CMS-based Web application is extracted and represented in a set of models at different abstraction levels. One of these models (CMS model) represents that knowledge within the CMS domain. The CMS model conforms to the **CMS Common Metamodel** that we have specifically defined for the *ADMigraCMS* method. In the *restructuring stage*, the CMS model is restructured to adapt the extracted knowledge to the features of the target CMS platform; finally, in the *forward engineering stage*, a set of models are generated from the CMS model. From one of these models defined at specific abstraction level, the code that implements the target CMS-based Web application is generated.

The **ADMigraCMS method** is structured in four different *modelling levels*: *Level 0* represents the code that implements the CMS-based Web application (PHP code); *Level 1* represents the code at platform-specific level using a model conforming to the ASTM metamodel (ASTM\_PHP model); *Level 2* specifies the code at platform-independent level with a model conforming to the KDM metamodel (Code model) and *Level 3* represents the code in the CMS domain with a the CMS model. For each modelling level, we have defined a *modelling language* in the form of *Domain Specific Languages* (DSL) such as, the PHP DSL (Level 0), the ASTM\_PHP DSL (Level 1), the KDM\_CODE DSL (Level 2) and the CMS DSL (Level 3). We have defined an abstract syntax with a metamodel and a concrete syntax with a customized graphical notation for each DSL.

Furthermore, the *ADMigraCMS* method systemizes the migration process with a set of *automated transformations* which allow the transition between the different modelling levels. The *ADMigraCMS* method defines three different types of automated transformations: text-to-model (T2M) transformation that extracts the knowledge from the code of a legacy CMS-based Web application, model-to-model (M2M) transformation that allows the transition from one model to another and model-to-text (M2T) transformation that generates the code that implements a target CMS-based Web application.

Up to now, the *ADMigraCMS* method is focused on the migration of Web applications based on open-source CMS platforms such as Drupal, Joomla! or Wordpress because of the their widespread use and relevant acceptance in the global market.

### Abstract III

This method is supported by a toolkit called ***ADMigraCMS toolkit*** that provides it with *graphical editors* for each DSL (tree-like and UML-like graphical editors) and the implementation of the whole set of automated transformations.

The *ADMigraCMS* method has been validated and refined by means of its application to the migration of two *case studies*: a Web application for the management of a coaching centre (*Coaching Web*) and a Web application for a wellness and nutrition centre (*Websana*). This validation allowed us to assess the correct specification of the metamodels, the proper performance of the graphical editors and the correct execution of the automated transformations. Furthermore, the research presented in this PhD Thesis has been validated by the scientific community with the presentation of partial results in renowned international and national conferences.



## Resumen

En la última década, las aplicaciones Web manejan grandes cantidades de contenido digital y soportan procesos cada vez más complejos. Por este motivo, las organizaciones se han visto con la necesidad de utilizar plataformas que les ayuden a mantener sus aplicaciones Web y a gestionar de una manera robusta y fiable el gran volumen de contenido que éstas manejan. Una de las soluciones más adoptadas ha sido la de basar las aplicaciones Web sobre los *Sistemas de Gestión de Contenidos (Content Management Systems, CMS)*. Estas **aplicaciones Web basadas en CMS** proporcionan algunas características tales como, la creación dinámica de contenido, la separación entre contenido y diseño, la definición de diferentes niveles de permisos para el acceso y la posibilidad de extender su funcionalidad de manera sencilla.

Actualmente, en el mercado podemos encontrar un número considerable de diferentes plataformas CMS. Este hecho, junto con los cambios de los objetivos que normalmente se dan en las organizaciones, causa que éstas se vean con la necesidad de migrar sus aplicaciones Web basadas en CMS a otras plataformas CMS más modernas o que satisfagan mejor sus necesidades.

Este proceso de migración implica un proceso de reingeniería complejo, largo y propensos a errores. En la actualidad, la **Architecture-Driven Modernization (ADM)** se considera una de las propuestas más efectivas para sistematizar este proceso y para mitigar todas sus desventajas. ADM aboga por la aplicación de técnicas y herramientas basadas en los principios de la *Model-Driven Architecture (MDA)* para llevar a cabo el proceso de migración. Además, desarrolla un conjunto de metamodelos estándar para representar la información involucrada en este proceso. Dos de estos metamodelos son: *el Abstract Syntax Tree Metamodel (ASTM)*, que permite representar mediante modelos específicos de plataforma la sintaxis de un código fuente, y *el Knowledge Discovery Metamodel (KDM)* que permite representar a nivel independiente de plataforma principalmente la semántica de este código.

Hemos llevado a cabo una revisión de la literatura cuyos resultados evidencian que no existe ningún método basado en ADM para la automatización del proceso de migración de aplicaciones Web basadas en CMS a otras plataformas CMS.

Por este motivo, proponemos el método *ADMigracMS*, **un método de migración basado en ADM que define una directrices para migrar aplicaciones Web basadas en CMS a otras plataformas CMS.**

El método *ADMigracMS* se compone de tres fases que definen un proceso de reingeniería en "herradura": la *fase de ingeniería inversa*, la *fase de restructuración* y la *fase de ingeniería directa*. Durante la *fase de ingeniería inversa*, se extrae el conocimiento a partir del código de la aplicación Web basada en CMS origen y se representa mediante un conjunto de modelos definidos a diferente nivel de abstracción. Uno de estos modelos es el modelo CMS que representa ese conocimiento dentro del dominio CMS. Este modelo está conforme al metamodelo **CMS Common Metamodel** que hemos definido e implementado específicamente para el método *ADMigracMS*. En la *fase de restructuración*, se reestructura el modelo CMS para adaptar ese conocimiento a las características de la plataforma CMS destino; finalmente, en la *fase de ingeniería directa*, un conjunto de modelos se generan a partir del modelo CMS reestructurado. A partir de uno de estos nuevos modelos definido a un nivel de abstracción más bajo, se genera el código que implementa la aplicación Web destino basada en CMS.

El método *ADMigracMS* se estructura en cuatro *niveles de modelado* diferentes. El *Nivel 0* representa el código que implementa la aplicación Web basada en CMS (la origen y la destino), el *Nivel 1* representa el código a un nivel de abstracción específico de plataforma mediante un modelo conforme al metamodelo ASTM (modelo ASTM\_PHP); el *Nivel 2*, representa el código a un nivel de abstracción independiente de plataforma mediante un modelo conforme al metamodelo KDM (Code model) y el *Nivel 3* representa el código en el dominio CMS con el modelo CMS. Para cada nivel de modelado, se ha definido un lenguaje de modelado mediante Lenguajes Específicos de Dominio (*Domain Specific Language*, DSL), tales como, el DSL PHP (Nivel 0), el DSL ASTM\_PHP (Nivel 1), el DSL KDM\_CODE (Nivel 2) y el DSL CMS (Nivel 3). Para cada DSL, hemos definido una sintaxis abstracta con un metamodelo y una sintaxis concreta con una notación gráfica específica.

Además, el **método *ADMigracMS*** sistematiza el proceso de migración mediante un conjunto de *transformaciones automatizadas* que permiten pasar de un nivel de modelado a otro. Este metamodelo define tres tipos de transformaciones automatizadas: transformación texto-a-modelo (*text-to-model*, T2M) para extraer el conocimiento del código fuente de la aplicación Web basada en CMS origen, transformación modelo-a-modelo (*model-to-model*, M2M) para pasar de un modelo a otro en el proceso de migración y transformación modelo-a-

texto (*model-to-text*, M2T) para generar el código que implementa la aplicación Web basada en CMS destino.

Hasta el momento, el método *ADMigraCMS* se centra en la migración de aplicaciones Web basadas en plataformas CMS de código abierto como *Drupal*, *Joomla!* o *Wordpress*, ya que a día de hoy son las plataformas CMS más utilizadas por las empresas para implementar sus aplicaciones Web.

El método *ADMigraCMS* está soportado por una herramienta llamada ***ADMigraCMS toolkit***. Esta herramienta provee al método de *editores gráficos* (al estilo de árbol y al estilo UML) para cada DSL e implementa el conjunto completo de transformaciones automatizadas.

El método *ADMigraCMS* se ha validado y redefinido mediante la migración de dos *casos de estudio*: una aplicación Web para la gestión de un centro de coaching (*Coaching Web*) y otra para la administración de un centro de bienestar y nutrición (*Websana*). Esta validación nos ha permitido evaluar la correcta especificación de los metamodelos, la ejecución sin errores de los editores gráficos y la correcta ejecución de las transformaciones automatizadas. Además, el trabajo presentado en esta Tesis Doctoral ha sido validado por la comunidad científica mediante la presentación de los resultados parciales en congresos nacionales e internacionales de renombrado prestigio.



## Acknowledgements

First of all, I would like to express my immense gratitude and acknowledgement to my Ph. D thesis directors, Valeria and Marcos, for their continuous support and the helpful advice that they gave me throughout this process. Thank you for all the things that I learned with you, from an intellectual and personal perspective. Valeria, thank you being so ready to help me and at the same so demanding of me. Also, for setting goals for me that have helped me over achieve, not just professionally, but personally. Marcos, thank you for always trusting me and for the love of a great job done well. The road to a PhD Thesis is not easy and now I understand why the relationship between a director and doctorate student is so important and intense. During these years, I have admired, hated, praised and cursed both of you. I have also cried, laughed, suffered and I have had a great time with you all. Taking all this into consideration, you have both become a reference point and I cherish both of you deeply. Thank you for everything. I will have you in my thoughts in my future. I would also like to give my sincere gratitude to Esperanza for having given me the opportunity to be part of her research group and for letting me discover the world of researching and teaching.

I would like to thank Professor Piero Fraternalli, as well as Professor Marco Brambilla, for their advice and their time dedicated to me when I was at the Universidad Politécnica de Milán. Their contributions where essential to carry out this thesis.

I would also like to thank the members of the Kybele research group. To Juancho, for sharing his knowledge with me. Each conversation with you was a milestone in the development of this thesis. To Verónica, for always being willing and able to give me a hand when I needed it. To Diana, for her great energy, wise comments and for transmitting her love and eagerness for teaching. To David, for always being a companion on the battlefield, always willing to help. To Álvaro, Jenifer, and Ángel, for all those moments shared in Department II. Also to Iván, the best office colleague anyone could ask for. A colleague that throughout the years became a friend, a brother. I certainly cannot express in just a few lines all the appreciation I have for you. You have been a key person in this process and this achievement is shared with you. I could not forget Karem, that without her help, all this would have been more difficult. Thank you for all the hours and

Feliu Trias X

hours that we worked elbow to elbow and for your willingness to always help me. Thank you. I will always be thanking you.

I would like to mention my university mates and friends from my technology and information degree, Edu, Laura and Javi for taking me in with open arms from the first day I got to Madrid. Thank you for introducing me into your lives. You all have been a great support.

I would like to mention all my friends that I have met in Madrid. To my university mates and friends from my audiovisual degree, Edurne, Rosa, Natalia and Juan, for all the indie film sessions and those great philosophical post-modernist discussions in the bars of Malasaña. Those times really helped me to disconnect from the thesis research. Also, to Billy, for his encouragement and his help on the English corrections and drafting for the summary part of the thesis. To Javi, Leire and Mar because they were always there when I needed them. Lastly, to Samer, my flatmate, who has lived first hand all my thesis process. I cannot sufficiently express my gratitude for all your help during this time. Thanks for listening and encouraging me. As well as for always pulling me off the edge when I did not see a solution to a problem. You are a great person. Thank you.

To my friends from Mallorca, Pep Toni, Àngels, Aina F. and Aina G., for being there unconditionally. Despite being in Madrid for many years, we have maintained our friendship and I have to thank you for that. Thank you for your surprise visits, for showing all your love and encouragement. I love you all.

Finally, my family. Thank you for sharing this amazing achievement with me. To my cousins, Xisca and Tià, which are like my brother and sister, whom I have grew up with. To my godmother, Assumpció, who is like my second mother. To my grandparents and my aunt Cati, who left us before I could write these words, I know they would be proud of my accomplishment. Lastly, of course, to my mother and father, Miquela and Tomeu, because you gave me everything I ever needed. Thank you for making me the person that I am. For passing along your values, showing me so much in life, for loving me unconditionally just the way I am and for always believing in me. I love and adore you.

---

En primer lugar, quería expresar mi inmensa gratitud y reconocimiento a mis directores de Tesis, Valeria y Marcos, por su continuo apoyo y los valiosos consejos recibidos durante todo este proceso. Gracias por todas las cosas que he aprendido con vosotros tanto desde el punto de vista intelectual como personal. Valeria, gracias por mostrarte tan cercana a la vez que exigente, y plantearme retos

que me han hecho superarme como persona y como investigador. A ti Marcos, gracias por confiar siempre en mí y por enseñarme el amor por el trabajo bien hecho. La realización de una Tesis no es un camino fácil y ahora entiendo por qué la relación director-doctorando es tan importante e intensa. Durante estos años os he admirado, os he odiado, os he maldecido, os he alabado; también he llorado, he reído, he sufrido y he disfrutado con vosotros. Por todo ello, os habéis convertido para mí en un referente que admiro y aprecio profundamente. Gracias por todo, os tendré muy presente en mi futuro. También quería mostrar mi sincero agradecimiento a Esperanza por haberme dado la oportunidad de haber formado parte de su grupo de investigación y haberme permitido descubrir el mundo de la investigación y la docencia.

Quiero agradecer al profesor Piero Fraternalli, así como al profesor Marco Brambilla, sus consejos y el tiempo dedicado durante mi estancia en la Universidad Politécnica de Milán. Sus aportaciones fueron esenciales para llevar a cabo esta Tesis.

Quería dar las gracias a mis compañeros del grupo de investigación Kybele. A Juancho por compartir su conocimiento conmigo. Cada conversación contigo se convertía en un hito en el desarrollo de esta Tesis. A Verónica por su siempre buena disposición a echar una mano en lo que nos hiciera falta. A Diana por su buena energía, sus sabios consejos y por transmitirme el amor e ilusión por la docencia. A David por ser un compañero de batalla dispuesto siempre a ayudar. A Álvaro, Jenifer y Ángel por los momentos compartidos en el Departamental II. Como no a Iván, el mejor compañero de despacho que me podía tocar. Un compañero que con los años se ha convertido en un amigo, en un hermano. No puedo expresar en unas líneas todo lo que tengo por agradecerte. Has sido una persona clave en este proceso y este logro es compartido contigo. Y no me podía olvidar de Karem, que sin su ayuda todo esto hubiese sido mucho más difícil. Gracias por la horas y horas que hemos pasado trabajando codo a codo y por tu mano siempre tendida a ayudarme; gracias, siempre te estaré agradecido.

Quería mencionar a mis compañeros y amigos de la carrera de informática Edu, Laura y Javi por acogerme con los brazos abiertos desde el primer día que llegué a Madrid y por haberme introducido en sus vidas. Habéis sido un gran apoyo.

También hacer mención al resto de amigos que he ido conociendo en Madrid. A mis compañeros y amigos de la carrera de comunicación audiovisual a Edurne, Rosa, Natalia y Juan por todas las sesiones de cine independiente y charlas filosófico-postmodernas en los bares de Malasaña que me han ayudado a

desconectar de mi investigación en la Tesis. También, a Billy por sus ánimos y su ayuda en la corrección y redacción de la memoria en inglés, a Javi, Leire y Mar por estar ahí siempre que los he necesitado. Por último, a Samer, mi compañero de piso, que ha vivido en primera persona todo mi proceso de la Tesis. No podré agradecerte todo lo que me has ayudado durante este tiempo. Gracias por escucharme, por animarme y por siempre estar dispuesto a sacarme del hoyo cuando todo parecía no tener solución; eres una gran persona, gracias.

A mis amigos de Mallorca, Mar, Pep Toni, Àngels, Aina F. y Aina. G. por estar ahí incondicionalmente. A pesar de llevar años en Madrid hemos mantenido nuestra amistad y os lo tengo que agradecer. Gracias por vuestras visitas sorpresa, por vuestras muestras de amor y cariño y por vuestros ánimos; os quiero.

Por último agradecer y compartir este logro con mi familia. A mis primos, Xisca y Tià que son como mis hermanos con los que me he criado y he crecido, a mi madrina, Assumpció, que es como mi segunda madre, a mis abuelos y mi tía Cati que aunque se han ido antes de que yo pudiera escribir estas palabras estoy seguro de que están muy orgullosos de mi logro. Y como no a mis padres, Miquela y Tomeu, porque lo habéis dado todo por mí, siempre. Gracias por haber hecho de mí la persona que soy, por haberme trasmitido tantos valores, por haberme enseñado tanto de la vida, por quererme tal y como soy y por creer en mí siempre; os quiero y os adoro.

## Table of Contents

<b>1</b>	<b>INTRODUCTION .....</b>	<b>1</b>
1.1	Problem Statement .....	1
1.1.1	<i>Reengineering Legacy Systems with ADM.....</i>	1
1.1.2	<i>The Relevance of Content Management Systems .....</i>	4
1.1.3	<i>Migration of CMS-based Web Applications .....</i>	5
1.2	Hypothesis and Objectives.....	6
1.3	Research Context .....	8
1.3.1	<i>Related Research Projects .....</i>	9
1.3.2	<i>External Research Stay .....</i>	10
1.4	Structure of the Dissertation .....	11
<b>2</b>	<b>RESEARCH METHOD.....</b>	<b>17</b>
2.1	Strategy .....	17
2.2	Identification of the Body of Knowledge.....	19
2.2.1	<i>Planning Phase.....</i>	20
2.2.2	<i>Execution Phase.....</i>	20
2.2.3	<i>Result Analysis Phase .....</i>	21
2.3	Definition of the Working Method .....	22
2.3.1	<i>Development Method .....</i>	22
2.3.2	<i>Validation Method .....</i>	24
2.4	Specification, Implementation and Validation.....	26
<b>3</b>	<b>STATE OF THE ART .....</b>	<b>30</b>
3.1	Previous Concepts.....	30
3.1.1	<i>Model-Driven Engineering .....</i>	30
3.1.2	<i>Model-Driven Reengineering (MDRE) .....</i>	31
3.1.3	<i>Architecture-Driven Modernization (ADM) .....</i>	33
3.1.4	<i>Modelling languages - Domain-Specific Languages (DSL)....</i>	38
3.1.5	<i>Content Management Systems .....</i>	39
3.2	Approaches in the Context of CMS-based Web Applications .....	41
3.2.1	<i>Research Questions Definition.....</i>	42

3.2.2	<i>Data Extraction Definition</i> .....	44
3.2.3	<i>Data Extraction</i> .....	45
3.2.3.1	Souer et al. (Web Engineering Method - WEM).....	46
3.2.3.2	Souer et al. (WebForm Diagram).....	48
3.2.3.3	Saraiva et al.....	50
3.2.3.4	Vlaanderen et al. ....	52
3.2.4	<i>Discussion</i> .....	54
3.3	Model-Driven Reengineering Approaches .....	57
3.3.1	<i>Research Questions</i> .....	58
3.3.2	<i>Data Extraction Definition</i> .....	59
3.3.3	<i>Data Extraction</i> .....	61
3.3.3.1	García-Rodriguez de Guzman et al. (Rational Web) .....	61
3.3.3.2	Blanco et al. ....	63
3.3.3.3	Perez-Castillo et al. (MARBLE).....	65
3.3.3.4	García-Rodriguez de Guzman et al. (PRECISO) .....	67
3.3.3.5	Perez-Castillo et al. (Schema Elicitation Method) .....	68
3.3.3.6	Rodríguez-Echevarría et al.....	70
3.3.3.7	Van Hoorn et al. ....	72
3.3.3.8	Sadovykh et al.....	73
3.3.3.9	Vasilecas et al .....	75
3.3.4	<i>Discussion</i> .....	76
3.4	Concluding Remarks.....	83
4	<b>CMS COMMON METAMODEL SPECIFICATION.....</b>	<b>89</b>
4.1	Process to Define the CMS Common Metamodel .....	89
4.2	CMS Platform Analysis .....	91
4.2.1	<i>CMS Market</i> .....	91
4.2.2	<i>Wordpress</i> .....	92
4.2.3	<i>Joomla!</i> .....	93
4.2.4	<i>Drupal</i> .....	93
4.3	CMS Metamodels Specification .....	95
4.3.1	<i>Wordpress Metamodel</i> .....	95

4.3.2	<i>Joomla! Metamodel</i> .....	100
4.3.3	<i>Drupal Metamodel</i> .....	106
4.4	Metamodel Modularization.....	111
4.5	CMS Common Metamodel Specification .....	112
4.5.1	<i>Navigation Concern</i> .....	117
4.5.2	<i>Presentation Concern</i> .....	121
4.5.3	<i>Behaviour Concern</i> .....	123
4.5.4	<i>Content Concern</i> .....	126
4.5.5	<i>User Concern</i> .....	130
4.6	Concluding Remarks.....	134
<b>5</b>	<b>THE ADMIGRACMS METHOD .....</b>	<b>137</b>
5.1	Method Definition.....	137
5.1.1	<i>Modelling Levels</i> .....	138
5.1.2	<i>Automated Transformations</i> .....	140
5.2	Specification of the Modelling Languages.....	142
5.2.1	<i>The PHP DSL</i> .....	142
5.2.1.1	The PHP Metamodel .....	144
5.2.2	<i>The ASTM_PHP DSL</i> .....	154
5.2.2.1	The ASTM_PHP Metamodel .....	155
5.2.2.2	Customized Graphical Notation .....	162
5.2.3	<i>The KDM_CODE DSL</i> .....	164
5.2.3.1	The KDM Metamodel .....	164
5.2.3.2	Customized Graphical Notation .....	172
5.2.4	<i>The CMS DSL</i> .....	173
5.2.4.1	Customized Graphical Notation .....	174
5.3	Specification of the Automated Transformations .....	175
5.3.1	<i>Between PHP Metamodel and ASTM_PHP Metamodel</i> .....	176
5.3.2	<i>Between ASTM_PHP Metamodel and KDM Metamodel</i> .....	186
5.3.3	<i>Between KDM Metamodel and CMS Common Metamodel</i> ..	196
5.4	Concluding Remarks.....	203
<b>6</b>	<b>THE ADMIGRACMS TOOLKIT .....</b>	<b>209</b>

6.1	Architecture of the <i>ADMigraCMS</i> Toolkit .....	210
6.1.1	<i>Conceptual Architecture of the ADMigraCMS Toolkit</i> .....	210
6.1.2	<i>Technical Design of the ADMigraCMS Toolkit</i> .....	211
6.2	Implementation of the Metamodels .....	213
6.2.1	<i>PHP Metamodel</i> .....	214
6.2.2	<i>CMS Common Metamodel</i> .....	227
6.3	Implementation of the Graphical Editors.....	229
6.3.1	<i>Tree-like Graphical Editors</i> .....	229
6.3.2	<i>UML-like Graphical Editor</i> .....	231
6.4	Implementation of the Automated Transformations .....	242
6.4.1	<i>L0-to-L1 T2M Transformation</i> .....	243
6.4.2	<i>L1-to-L2 M2M Transformation</i> .....	248
6.4.3	<i>L2-to-L3 M2M Transformation</i> .....	250
6.4.4	<i>L3-to-L2 M2M Transformation</i> .....	252
6.4.5	<i>L2-to-L1 M2M Transformation</i> .....	253
6.4.6	<i>L1-to-L0 M2T Transformation</i> .....	255
6.5	Concluding Remarks.....	258
<b>7</b>	<b>VALIDATION.....</b>	<b>263</b>
7.1	Validation Method .....	263
7.2	Case Study 1 and Case Study 2 Selection.....	265
7.3	Data Retrieval .....	266
7.4	Design and Execution of the Case Study .....	271
7.4.1	<i>Extraction of the ASTM_PHP Model</i> .....	274
7.4.2	<i>Generation of Code Model</i> .....	279
7.4.3	<i>Generation of CMS Model</i> .....	283
7.4.4	<i>CMS Model Restructuring</i> .....	286
7.4.5	<i>Generation of Target Code Model</i> .....	287
7.4.6	<i>Generation of Target ASTM_PHP Model</i> .....	291
7.4.7	<i>Generation of PHP Code</i> .....	297
7.5	Report and Conclusions .....	299
7.6	Concluding Remarks.....	301

Table of Contents XVII

<b>8</b>	<b>CONCLUSIONS AND FUTURE WORKS .....</b>	<b>305</b>
8.1	Analysis of Achievements .....	305
8.2	Main Contributions .....	312
8.3	Scientific Results .....	315
8.4	Future Works .....	317
<b>9</b>	<b>APPENDIX A .....</b>	<b>323</b>
<b>10</b>	<b>APPENDIX B.....</b>	<b>339</b>
<b>11</b>	<b>APPENDIX C .....</b>	<b>357</b>
<b>12</b>	<b>APPENDIX D .....</b>	<b>363</b>
<b>13</b>	<b>APPENDIX E.....</b>	<b>371</b>
<b>14</b>	<b>APPENDIX F.....</b>	<b>387</b>
<b>15</b>	<b>BIBLIOGRAPHY AND ONLINE RESOURCES .....</b>	<b>403</b>
<b>16</b>	<b>ACRONYMS.....</b>	<b>423</b>



## List of Figures

Figure 1-1. Research projects and research stays.....	9
Figure 1-2. Anatomy of the proposal in relation to the structure of the dissertation.....	12
Figure 2-1. Overview of the research method.....	17
Figure 2-2. Sistematic Review process proposed by Biolchini et al. (Biolchini et al., 2005).....	19
Figure 2-3. SLR phases applied to our research.....	20
Figure 2-4. Development method.....	23
Figure 2-5. Validation process by using case studies.....	25
Figure 2-6. Definition of the a) specification, b) implementation and c) validation stage.....	27
Figure 3-1. MDE disciplines.....	31
Figure 3-2. Reengineering stages.....	32
Figure 3-3. Reengineering stages in the context of ADM.....	34
Figure 3-4. KDM packages and layers.....	36
Figure 3-5. WCM four-hierarchy.....	40
Figure 3-6. Chart about the CMS market (BuiltWith, 2013).....	41
Figure 3-7. Web Engineering Method and Web engineering views.....	47
Figure 3-8. WebForm diagram definition and Web Engineering views.....	49
Figure 3-9. CMS-ML and CMS-IL modelling languages.....	51
Figure 3-10. Adaptation of OOWS to CMS domain.....	53
Figure 3-11. Relational Web method.....	62
Figure 3-12. Modernization process for Secure DW.....	64
Figure 3-13. MARBLE method.....	66
Figure 3-14. PRECISO method.....	67
Figure 3-15. Schema Elicitation method.....	69
Figure 3-16. Modernization of legacy Web applications to RIA.....	71
Figure 3-17. Dynamod method.....	72
Figure 3-18. Sadovykh et al. method.....	74

Figure 3-19. Vasilecas et al. method .....	75
Figure 4-1. Process to define the CMS Common Metamodel.....	90
Figure 4-2. MMWordpress.....	96
Figure 4-3. MMJoomla.....	101
Figure 4-4. MMDrupal .....	107
Figure 4-5. Intersection strategy.....	113
Figure 4-6. Intersection and intersecting pairs strategy.....	113
Figure 4-7. Union strategy.....	113
Figure 4-8. MMCMSCommon.....	116
Figure 4-9. Navigation concern in the MMCMSCommon.....	117
Figure 4-10. Presentation concern in MMCMSCommon.....	121
Figure 4-11. Behaviour concern in MMCMSCommon.....	123
Figure 4-12. Content concern in MMCMSCommon.....	127
Figure 4-13. User concern in MMCMSCommon.....	131
Figure 5-1. <i>ADMigracms</i> method and their modelling levels.....	139
Figure 5-2. Automated transformations between modelling levels.....	141
Figure 5-3. L0 modelling level.....	143
Figure 5-4. L1 modelling level.....	154
Figure 5-5. Hierarchy of GASTM elements.....	155
Figure 5-6. <i>Expression</i> , <i>Statement</i> and <i>DefinitionObject</i> elements of the GASTM domain .....	157
Figure 5-7. Specific PHP expressions defined in SASTM.....	159
Figure 5-8. Specific PHP binary operators defined in SASTM.....	160
Figure 5-9. Specific PHP unary operators defined in SASTM.....	160
Figure 5-10. Specific PHP statements defined in SASTM.....	161
Figure 5-11. <i>PHPCompilationUnit</i> element defined in SASTM.....	162
Figure 5-12. L2 modelling level.....	164
Figure 5-13. Code and action packages of the KDM metamodel.....	165
Figure 5-14. Hierarchy of the elements of the code and action packages of the KDM.....	165
Figure 5-15. Elements of the code package of the KDM. ....	166

Figure 5-16. Elements of the action package of the KDM .....	167
Figure 5-17. L3 modelling level .....	174
Figure 6-1. Conceptual architecture of the <i>ADMigraCMS</i> toolkit.....	211
Figure 6-2. Technical design of the <i>ADMigraCMS</i> toolkit.....	212
Figure 6-3. Classes of he PHP metamodel representing PHP Expressions. ....	215
Figure 6-4. ArrayType parser rule. ....	216
Figure 6-5. FunctionCallExpression parser rule. ....	216
Figure 6-6. ArrayAccess parser rule.....	217
Figure 6-7. NegateOrCast parser rule.....	218
Figure 6-8. LogicalNot parser rule. ....	218
Figure 6-9. Reference parser rule. ....	219
Figure 6-10. Assignment parser rule. ....	220
Figure 6-11 . Addition parser rule. ....	220
Figure 6-12. EqualityCheck parser rule.....	221
Figure 6-13 . BitewiseShift parser rule.....	221
Figure 6-14. LogicalAnd parser rule. ....	221
Figure 6-15. InstanceOf parser rule.....	222
Figure 6-16. Classes of the PHP metamodel representing PHP Statements. ....	222
Figure 6-17. VariableDefStatement parser rule.....	223
Figure 6-18. FunctionDefStatement parser rule. ....	224
Figure 6-19. IfStatement parser rule.....	226
Figure 6-20. ForStatement parser rule. ....	226
Figure 6-21. ReturnStatement parser rule.....	227
Figure 6-22. Ecore Model for the implementation of the CMS Common Metamodel. ....	228
Figure 6-23. Relation between the GenModel and the metamodel.....	230
Figure 6-24. Overview of the generation of the Java code. ....	230
Figure 6-25. a) tree-like graphical editor for ASTM_PHP DSL, b) tree-like graphical editor for KDM_CODE DSL.....	231
Figure 6-26. Process to define the UML-like graphical editor for the CMS DSL. .....	232

Figure 6-27. Extraction of the Emfatic implementation. ....	232
Figure 6-28. Emfatic implementation including GMF annotations. ....	233
Figure 6-29. Code to create the plugin in <i>FixGMFGen transformation</i> . ....	235
Figure 6-30. Physical structure of <i>CMSFigures</i> plugin. ....	236
Figure 6-31. Implementatio in Java of <i>TermFigure</i> class.....	237
Figure 6-32. Implementation in Java of the <i>Activator</i> class. ....	237
Figure 6-33. a) Customized graphical noation of the Term element b) EOL code to customize the label associated to the class Term.....	238
Figure 6-34. a) Customized graphical noation of the termContent link, b) EOL code to customize the reference termContent of the class Term. ....	239
Figure 6-35. a) Customized graphical noation of the Theme class, b) EOL code to customize the class Theme. ....	240
Figure 6-36. a) Customized graphical noation of the compartment of the Theme class, b) EOL code to customize the compartment of the class Theme. ....	241
Figure 6-37. Toolbar of the UML-like graphical editor. ....	241
Figure 6-38. UML-like graphical editor for the CMS DSL.....	242
Figure 6-39. Implementation of the model extractor. ....	243
Figure 6-40. a) Model extractor UI, b) Implementation of the <i>Principal</i> class. ..	244
Figure 6-41. Resulting window for the <i>JFileChooser</i> object. ....	245
Figure 6-42. Adding an <i>ActionListener</i> object to the buttons. ....	245
Figure 6-43. Exerpt of the EventHandler class implementation. ....	246
Figure 6-44. Windows displayed when ASTM_PHP model is created correctly. .....	246
Figure 6-45. Windows displayed when the Php files are not found. ....	246
Figure 6-46. Excerpt of <i>processStatement</i> method. ....	248
Figure 6-47. Astm2Kdm module.....	249
Figure 6-48. <i>functionDef2MethodUnit</i> matched rule in ATL. ....	250
Figure 6-49. Kdm2cms module. ....	251
Figure 6-50. <i>MethodUnit2Menu</i> matched rule in ATL. ....	251
Figure 6-51. cms2kdm module.....	252
Figure 6-52. <i>menu2ActionElement</i> matched rule in ATL. ....	253
Figure 6-53. kdm2astm module.....	254

List of Figures XXIII

Figure 6-54. <i>methodUnit2Statement</i> matched rule in ATL .....	254
Figure 6-55. <i>getFunctionDefinition</i> lazy rule in ATL .....	255
Figure 6-56. <i>GenerateStatement</i> template in Acceleo.....	256
Figure 6-57. <i>GenerateExpression</i> template in Acceleo.....	256
Figure 6-58. <i>GenerateBinaryExp</i> template in Acceleo.....	257
Figure 6-59. <i>GenerateIdRef</i> template in Acceleo.....	257
Figure 6-60. <i>GenerateOperator</i> template in Acceleo.....	257
Figure 6-61. <i>GenerateArray</i> template in Acceleo.....	258
Figure 7-1. Tasks of the validation method.....	263
Figure 7-2. a) Implementation of Coaching Web in Wordpress, b) Implementation of Coaching Web in Drupal.....	265
Figure 7-3. a) Implementation of Websana on Drupal, b) Implementation of Websana on Wordpress.....	266
Figure 7-4. PHP code implementing the <i>Block</i> in Wordpress.....	267
Figure 7-5. PHP code implementing the <i>Block</i> in Drupal.....	267
Figure 7-6. PHP code implementing the <i>Menu</i> in Drupal.....	268
Figure 7-7. PHP code implementing the <i>MenuItem</i> in Drupal.....	268
Figure 7-8. PHP code implementing the <i>Page</i> in Drupal.....	269
Figure 7-9. PHP code implementing the <i>Menu</i> , <i>MenuItem</i> and <i>Page</i> in Wordpress.....	269
Figure 7-10. Overview of the tasks executed in the case studies.....	271
Figure 7-11. Extraction of the ASTM_PHP model.....	274
Figure 7-12. The <i>FunctionDefinition</i> element within the ASTM_PHP model....	276
Figure 7-13. The <i>VariableDefinition</i> element within the ASTM_PHP model. ....	277
Figure 7-14. The <i>ExpressionStatement</i> element within the ASTM_PHP model.	278
Figure 7-15. The <i>ReturnStatement</i> element within the ASTM_PHP model.....	278
Figure 7-16. Generation of Code model.....	279
Figure 7-17. The <i>MethodUnit</i> element within the Code model.....	281
Figure 7-18. The <i>ActionElement</i> element ( <i>NewArray</i> ) within the Code model...	281
Figure 7-19. The <i>ActionElement</i> element ( <i>NewArray</i> ) within the Code model...	282
Figure 7-20. The <i>ActionElement</i> element ( <i>Return</i> ) within the Code model. ....	283

Figure 7-21. Generation of the CMS model .....	283
Figure 7-22. <i>Menu</i> , <i>MenuItem</i> , <i>Page</i> and <i>TextContent</i> elements within the CMS model .....	286
Figure 7-23. Restructuring CMS model .....	286
Figure 7-24. CMS model restructured .....	287
Figure 7-25. Generation of target Code model .....	288
Figure 7-26. Target Code model fragment extracted from the <i>MenuItem</i> element. .....	289
Figure 7-27. Target Code model fragment extracted from the <i>Page</i> and <i>TextContent</i> elements .....	291
Figure 7-28. Generation of target ASTM_PHP model .....	292
Figure 7-29. Excerpt of the target ASTM_PHP model generated .....	294
Figure 7-30. Excerpt of the target ASTM_PHP model generated .....	296
Figure 7-31. Generation of PHP code .....	297
Figure 9-1. Método de investigación .....	330
Figure E-9-2. Definición de la etapa de a) especificación, b) implementación and c) validación .....	331
Figure 14-1. AST_PHP model representing the function <i>customized_menu_websana_install</i> .....	388
Figure 14-2. ASTM_PHP model for the function call <i>websana_exercises</i> .....	389
Figure 14-3. Code model representing the function <i>customized_menu_websana_install</i> .....	391
Figure 14-4. Code model model for the function call <i>websana_diets</i> .....	391
Figure 14-5. CMS model representing the customized_menu_websana .....	393
Figure 14-6. Target Code model representing the function <i>created_menu_customized_menu_websana</i> .....	395
Figure 14-7. Target ASTM_PHP model representing the menu .....	397

## List of Tables

Table 3-1. Data extraction from the approaches related to CMS-based Web applications.....	54
Table 3-2. Data extraction from the model-driven reengineering approaches.....	77
Table 4-1. Market share yearly trends from 2010 to 2014. ....	91
Table 4-2. Analysis of the three CMS platforms. ....	94
Table 4-3. Elements of MMWordpress. ....	97
Table 4-4. Relationships of MMWordpress. ....	98
Table 4-5. Elements of the MMJoomla. ....	102
Table 4-6. Relationships of MMJoomla. ....	103
Table 4-7. Elements of the MMDrupal.....	108
Table 4-8. Relationships of the MMDrupal.....	109
Table 4-9. Union of the elements conforming the MMCMSCommon. ....	114
Table 5-1. Categorization of PHP elements.....	144
Table 5-2. Mappings from simple expressions to PHP elements. ....	145
Table 5-3. Mappings from PHP expressions to PHP elements. ....	146
Table 5-4. Unary operators in PHP. ....	147
Table 5-5. Mappings from unary expressions to PHP elements. ....	148
Table 5-6. Binary operators in PHP.....	149
Table 5-7. Mappings from binary expressions to PHP elements. ....	150
Table 5-8. Categorization of PHP Statements. ....	150
Table 5-9. Mappings from definition statements to PHP elements. ....	151
Table 5-10. Mappings from expression statements to PHP elements. ....	152
Table 5-11. Mappings from control statements to PHP elements.....	153
Table 5-12. Elements defined within the SASTM.....	158
Table 5-13. Customized graphical notation of the ASTM_PHP DSL. ....	163
Table 5-14. Definition of the micro action Assign. ....	169
Table 5-15. Definition of the Micro action Condition. ....	169
Table 5-16. Definition of the micro action Call.....	170

List of Tables XXVI

Table 5-17. Definition of the micro action MethodCall. ....	170
Table 5-18. Definition of the micro action Switch. ....	171
Table 5-19. Definition of the micro action Compound. ....	172
Table 5-20. Customized graphical notation for the KDM_CODE DSL. ....	172
Table 5-21. Customized graphical notation for the CMS DSL. ....	174
Table 5-22. Mappings from software assets to root elements in the ASTM_PHP metamodel. ....	176
Table 5-23. Mappings from simple expression to ASTM_PHP elements. ....	177
Table 5-24. Mappings from PHP expressions to ASTM_PHP elements. ....	178
Table 5-25. Mappings from unary expressions to ASTM_PHP elements. ....	179
Table 5-26. Mappings from binary expressions to ASTM_PHP elements. ....	180
Table 5-27. Mappings from definition statements to ASTM_PHP elements. ....	182
Table 5-28. Mappings from expression statements to ASTM_PHP elements. ....	184
Table 5-29. Mappings from control statements to ASTM_PHP elements. ....	185
Table 5-30. Mappings from the root elements of the ASTM_PHP to the KDM metamodel. ....	186
Table 5-31. Mappings from simple expressions to KDM elements. ....	187
Table 5-32. Mappings from PHP expressions to KDM elements. ....	189
Table 5-33. Mappings from unary expressions to KDM elements. ....	191
Table 5-34. Mappings from binary expressions to KDM elements. ....	192
Table 5-35. Mappings from definition statements to KDM elements. ....	192
Table 5-36. Mappings from expression statements to KDM elements. ....	194
Table 5-37. Mappings from control statements to KDM elements. ....	195
Table 5-38. Mappings from the root elements of KDM to the CMS Common Metamodel. ....	197
Table 5-39. Mappings from the KDM elements to the CMS elements of navigation concern. ....	198
Table 5-40. Mappings from the KDM elements to CMS elements of presentation concern. ....	199
Table 5-41. Mappings from the KDM elements to the CMS elements of behaviour concern. ....	200
Table 5-42. Mappings from the KDM elements to the CMS elements of content concern. ....	201

Table 5-43. Mappings from the KDM elements to the CMS elements of user concern .....	202
Table 6-1. Implementation of the simple expressions in Xtext .....	215
Table 6-2. Implementation of the PHP expressions in Xtext.....	216
Table 6-3. Implementation of the unary expressions in Xtext.....	217
Table 6-4. Implementation of the binary expressions in Xtext.....	219
Table 6-5. Implementation of the definition statements in Xtext .....	223
Table 6-6. Implementation of the expression statements in Xtext.....	224
Table 6-7. Implementation of the control statements in Xtext .....	225
Table 6-8. Implementation of L0-to-L1 T2M transformation.....	247
Table 6-9. Implementation of L1-to-L2 M2M transformation .....	249
Table 6-10. Implementation of the L2-to-L3 M2M transformation. ....	250
Table 6-11. Implementation of the L3-to-L2 M2M transformation. ....	252
Table 6-12. Implementation of L2-to-L1 M2M transformation. ....	253
Table 7-1. Coverage of the interesting points to validate with the Websana case study .....	273
Table 7-2. Transformations from the PHP code to the ASTM_PHP model.....	275
Table 7-3. Transformations from the ASTM_PHP model to the Code model. ...	279
Table 7-4. Transformations from the Code model to the CMS model. ....	284
Table 7-5. Transformations from the restructured CMS model to the target Code model.....	288
Table 7-6. Transformations from the structured CMS to the target Code model.	290
Table 7-7. Transformations from the target Code model to the target ASTM_PHP model.....	292
Table 7-8. Transformation from the target Code model to target ASTM_PHP model.....	294
Table 7-9. Generation of PHP code from the target ASTM_PHP model. ....	297
Table 10-1. QS adapted (for the SLR1) to the syntax of the digital library.....	348
Table 10-2. Result of the search process of the SLR1 .....	350
Table 10-3. QS adapted (for the SLR2) to the syntax of the digital library.....	350
Table 10-4. Result of the search process of the SLR2 .....	351
Table 10-5. Primary studies obtained in SLR1 .....	351

List of Tables XXVIII

Table 10-6. Primary studies obtained in SLR2.....	352
Table 11-1. Mappings from other PHP expressions to PHP elements. ....	357
Table 11-2. Mappings from Unary Expressions to PHP elements. ....	357
Table 11-3. Mappings from binary xxpressions to PHP elements. ....	358
Table 11-4. Mappings from definition statements to PHP elements. ....	359
Table 11-5. Mappings from Expression Statements to PHP elements. ....	359
Table 11-6. Mappings from control statements to PHP elements. ....	360
Table 12-1. Graphical notation of the elements of the user concern .....	363
Table 12-2. Graphical notation of the elements of the navigation concern .....	364
Table 12-3. Graphical notation of the elements of the content concern .....	364
Table 12-4. Graphical notation of the elements of the behaviour concern .....	366
Table 12-5. Graphical notation of the elements of the presentation concern .....	367
Table 13-1. Mappings from PHP expressions to ASTM_PHP elements.....	371
Table 13-2. Mappings from unary expressions to ASTM_PHP elements.....	372
Table 13-3. Mappings from binary expressions to ASTM_PHP elements.....	372
Table 13-4. Mappings from classDefStatement to ASTM_PHP element. ....	374
Table 13-5. Mappings from object access and member to ASTM_PHP elements.	374
Table 13-6. Mappings from control statements to ASTM_PHP elements. ....	375
Table 13-7. Mappings from expressions to KDM elements. ....	375
Table 13-8. Mappings from definition statements to KDM elements. ....	376
Table 13-9. Mappings from control statements to KDM elements. ....	377
Table 13-10. Mappings from KDM elements to CMS elements of the navigation concern.....	378
Table 13-11. Mappings from KDM elements to CMS elements of the presentation concern. ....	379
Table 13-12. Mappings from KDM elements to CMS elements of the behaviour concern. ....	380
Table 13-13. Mappings from KDM elements to CMS elements of the content concern. ....	382
Table 13-14. Mappings from KDM elements to CMS elements of the user concern. ....	383

Table 14-1. Transformation from PHP elements to ASTM_PHP model .....	387
Table 14-2. Transformations from the PHP elements to the ASTM_PHP model. .....	388
Table 14-3. Transformations from the ASTM_PHP model to Code model. ....	389
Table 14-4. Transformations from the ASTM_PHP model to Code model. ....	391
Table 14-5. Transformation from the Code model to the element Menu. ....	392
Table 14-6. Transformation from the Code model to the element TextContent..	393
Table 14-7. Transformation from the restructured CMS model to Target Code model. ....	394
Table 14-8. Transformation from the Target Code model to Target ASTM_PHP model. ....	395
Table 14-9. Transformations between the target ASTM_PHP model to the PHP code. ....	397
Table 14-10. Transformations between the target ASTM_PHP model to the PHP code. ....	398



## *Chapter 1: Introduction*

---



In this PhD Thesis, we propose the definition of a method based on the ADM to address the migration of CMS-based Web applications to other Content Management Systems as well as the construction of a toolkit which supports this method.

Section 1.1 introduces the context of this PhD Thesis and the problem to address. Section 1.2 presents the main hypothesis and the objectives covered by this research. Section 1.3 describes the research context in which this work has been developed, indicating the research projects in which this PhD Thesis has been framed. Finally, Section 1.4 provides a general overview on the rest of this dissertation.

## 1.1 Problem Statement

In this section, we describe the context of reengineering and how *Architecture Driven Modernization* (ADM) is presented as an initiative to provide solutions in this context. Then, we focus on the context of the *Content Management Systems* (CMS). We explain the advantages provided by these systems to the organizations for maintaining their complex Web applications as well as for supporting the management of their huge amount of digital content. At the end of this section, we emphasize the necessity of providing methods that address the migration of these CMS-based Web applications to other CMS platforms that meet the necessities of the organizations better.

### 1.1.1 Reengineering Legacy Systems with ADM

Information Systems (IS) used by organizations are continuously evolving. With this in mind, they can become obsolete in new real-world contexts (Lehman, 1998) as time progresses. These IS, considered to be legacy systems, are valuable assets for organizations since they embed important business knowledge (Ian Sommerville, 2002) but they become a problem at the time of their maintenance due to their high costs (high costs in re-documentary and in architecture restructuring) (Polo, Piattini, & Ruiz, 2003). The IEEE (IEEE, 1983) defines the software maintenance as: “*modification of software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment*”.

Legacy systems are usually maintained because it is cheaper to maintain them than to redevelop or to replace them (Sneed, 2005). The replacement with a new IS is risky since it has a great impact in technological, human and economic aspects (Sneed, 2005). From the technological point of view, specific functionalities can be lost due to the technological changes, otherwise the human aspect is affected since the replacement involves retraining the users in order for them to understand the new IS. Finally, the economic aspect is also affected since the replacement of an entire legacy system can exceed an organization's budget as well as it can imply the loss of the valuable business knowledge stored over time. On the other hand, the software maintenance is a better solution since it allows to preserve the business knowledge embedded in the legacy system as well as to control other costs (Bennett & Rajlich, 2000), (Bianchi, Caivano, Marengo, & Visaggio, 2003).

During the last few years, **Reengineering** (Chikofsky & Cross, 1990) has been the main tool for addressing the software maintenance of legacy systems (Bianchi et al., 2003) since it facilitates the reuse of the software artefacts and preserves the knowledge embedded in the system at a tolerable cost (Bennett & Rajlich, 2000). Nevertheless, Reengineering entails two main drawbacks (Pérez-Castillo, Guzmán, & Piattini, 2011); first, the reengineering for large complex legacy systems entails high maintenance costs (Canfora & Penta, 2007); second, these processes are carried out in a non-standardized and ad-hoc manner which hinders their automation (Kazman, Woods, & Carrière, 1998). For these two reasons, the software industry is demanding standardized and automated reengineering processes.

In the last few years, **Architecture-Driven Modernization** (ADM) (Khusidman & Ulrich, 2007), (Sadovskyh, Vigier, Hoffmann, et al., 2009a), (OMG, 2009), (Pérez-Castillo, Guzmán, et al., 2011) has proved to be an important solution for those problems. ADM advocates for carrying out reengineering processes taking into account the principles of the Model-Driven Development (MDD) (Miller & Mukerji, 2003), (Bézivin, 2004a). Concretely, it is based on the principles of the Model-Driven Architecture (MDA) (Frankel, 2002), (Miller & Mukerji, 2003), (Kleppe & Warmer, 2003), (Flore, 2003), (Mellor, Scott, Uhl, & Weise, 2004), (Guttman & Parodi, 2006), (Selic, 2008), (Watson, 2008). ADM is focused on formalizing the reengineering process by using models which represent the software artefacts of the legacy system at different abstraction levels –Platform Specific Model (PSM), Platform Independent Model (PIM) and Computation Independent Model (CIM)– and on automating it by the use of

automated transformations. Among the benefits provided by ADM we highlight the following ones (OMG, 2007): 1) it revitalizes the legacy system, 2) it reduces maintenance costs, 3) it extends the useful life of legacy systems, 4) it improves the ROI (Return On Investement) of these systems and 5) it eases the migration with other systems and other environments.

Apart from adopting MDA, ADM develops a set of standard metamodels (seven metamodels) to represent in a formalized manner the information involved in the reengineering process. Currently, out of the seven metamodels, there are three available: **Abstract Syntax Tree Metamodel** (ASTM) (OMG, 2005a), **Knowledge Discovery Metamodel** (KDM) (ISO/IEC, 2012) and **Structured Metrics Metamodel** (SMM) (OMG, 2012b). Besides supporting the software reengineering process, they allow developers to save time and effort in creating their own metamodels. Models conforming to ASTM (ASTM models) allow to represent the assets extracted from a legacy system in the form of Abstract Syntax Trees (AST) (Fischer, Lusiardi, & Wolff von Gudenberg, 2007) at a platform-specific level. On the other hand, models which conform to KDM (KDM models) allow to represent the semantics as well as the syntax of these assets at a platform-independent level. Finally, the models conforming to SMM are used to describe metrics and measurements in KDM models.

The key for automating a reengineering process is the definition of automated transformations. It is possible to define three types of transformations: text-to-model (T2M) transformations, the ones which allow to obtain a model from the software artefacts of the legacy system; model-to-model (M2M) transformations allow them to increase or decrease the abstraction level of the models; and model-to-text (M2T) transformations generate the target code from a model (Gerber, Lawley, Raymond, Steel, & Wood, 2002), (Selic, 2003), (Bézivin, 2004a), (Tratt, 2005).

These transformations make possible to link all the steps of the reengineering process, defining a so-called horseshoe process composed of three stages as originally defined by (Chikofsky & Cross, 1990): **the reverse engineering stage**, is the left-hand side stage of the horseshoe which analyzes the legacy system and their interrelationships by using models at different abstraction levels; **the restructuring stage**, is the curve upper part of the horseshoe and takes the models from the reverse engineering stage and transform them into others at the same abstraction level; and **the forward engineering stage**, is the right-hand side of the horseshoe that generates the physical implementation of the target system at a low abstraction level.

### **1.1.2 The Relevance of Content Management Systems**

The World Wide Web has evolved towards a platform for sophisticated enterprise applications and complex business processes, over the last few years (Souer, Kupers, Helms, & Brinkkemper, 2009). Organizations rely on the Web to support their business processes and use the Internet as a way to gain competitive advantage, global collaboration and integration with external partners (Turban, McLean, & Wetherbe, 1999), (Lee & Shirani, 2004). As a result, the number of Web applications (Isakowitz, Bieber, & Vitali, 1998) developed by industry and deployed by organizations to support their business models, has greatly increased.

In parallel, the volume of digital content managed by these organizations has also increased dramatically in the last decade. Thus, these organizations have experienced the necessity of using powerful management tools to consistently maintain their Web applications (McKeever, 2003), (Souer, Weerd, Versendaal, & Brinkkemper, 2007). In that sense, **Content Management Systems** (CMS) have become the most efficient tool allowing to collect, manage and maintain huge amounts of digital content in a robust and reliable manner (Boiko, 2001). Hence, many of the Web applications have been implemented above these CMS platforms (CMS-based Web applications). In comparison with traditional Web applications, they provide several benefits. Below, we can see a set of them:

- *Dynamic creation of content:* content is created and added dynamically by non-technical users of the Web, without requiring the intervention of the webmaster. It allows to overcome the bottlenecks that occur in the webmaster role.
- *Separation between content and design:* the graphical design of the Web page is stored in a template and the content is stored in a database or a separate document. It avoids possible inconsistencies in the look-and-feel of the Web page.
- *Different levels of access authority:* CMSs allow the definition of roles with different levels of permissions and access rights to the content managed by the Web application. It allows to protect more robustly this content access.
- *Functionality extension:* CMSs allow Web applications to extend their functionality by inserting existing new functional modules. It reduces the time in the development process.

The CMS market is evolving continuously, so we can find a large number of CMS platforms offering different facilities and meeting different domains, such

as blogs, e-commerce or e-learning (Shreves, 2011). It is possible to find proprietary CMS platforms such as SDL ACM (ACM, 2014), Content Verse (Verse, 2014) or Cascade Server (Hill, 2014) as well as open-source CMS such as Drupal (Drupal, 2014), Joomla! (Joomla!, 2014) and Wordpress (Wordpress, 2014). Therefore, organizations can opt for a wide range of new CMS platforms as well as for improved versions of the same CMS.

### **1.1.3 Migration of CMS-based Web Applications**

The continuous evolution of the CMS market along with the changes in the objectives of organizations, can cause them to see the necessity to migrate their legacy CMS-based Web applications to more modern CMS platforms meeting their needs better. This **migration process** entails a complex, time-consuming and error-prone reengineering process, so that it is necessary to support it by a method to mitigate all these drawbacks. Currently, the migration of legacy CMS-based Web application to other CMS platforms is not even addressed by any reengineering method according to the literature review performed in this publication (Trias, de Castro, López-Sanz, & Marcos, 2013a).

Therefore, to solve this gap, we considered interesting to propose a method to support this migration process. Furthermore, this method shoud be based on ADM, considering the advantages of this approach explained in Section 1.1.1.

From these two ideas, we propose the ***ADMigraCMS method, an ADM-based method that defines standard guidelines to migrate CMS-based Web applications to other CMS platforms***. This method is composed of three reengineering stages defining a horseshoe process: *reverse engineering stage*, *restructuring stage* and *forward engineering stage* and it is supported by a toolkit that allows systemizing the migration process.

The ***ADMigraCMS*** method is structured in four *modelling levels (Level 0, Level 1, Level 2 and Level 3)* that allow to represent the knowledge involved in the migration process from different points of view at different abstraction levels. Each modelling level is related to a *modelling language* that is implemented in the form of *Domain Specific Language (DSL)*. Hence, we have the PHP DSL (Level 0), the ASTM\_PHP DSL (Level 1), the KDM\_CODE DSL (Level 2) and the CMS DSL (Level 3). This last DSL allows representing the knowledge being migrated within the CMS domain using the *CMS Common Metamodel*.

The ***ADMigraCMS*** method defines a set of *automated transformations* between the different modelling languages that allow to systemize the migration

process. These automated transformations are classified in : *text-to-model (T2M)* transformation that extracts the knowledge from the code of a legacy CMS-based Web application, *model-to-model (M2M)* transformation that allows the transition from one model to another and *model-to-text (M2T)* transformation that generates the code that implements a target CMS-based Web application.

Our method is focused on the migration of CMS-based Web applications based on open-source CMS platforms. In particular, we consider platforms such as Drupal, Joomla! or Wordpress because of their extended use and relevant acceptance in the global industries and markets (Shreves, 2011). Most of these open-source CMS platforms are implemented in PHP so that we can pay special attention to this code.

## 1.2 Hypothesis and Objectives

The **hypothesis** formulated in this PhD Thesis is that — “*In an evolving world like the Web Engineering world and in a changing market like the CMS market, it is feasible to offer organizations a method which allows them to migrate their legacy CMS-based Web applications to other CMS platforms which meet their necessities better*”.

The **main objective** of the presented PhD Thesis, derived directly from the previous hypothesis, is, therefore: “*to specify an ADM-based method which automates the migration of legacy CMS-based Web applications to other CMS platforms*”.

Taking into account the previous requirements and desired features, in order to achieve the objective marked for this PhD Thesis, the next partial objectives are set:

**Obj 1.** Analysis and evaluation of previous research work and approaches related to the topic of this PhD Thesis. Having noted that this PhD Thesis leans on two clearly identified areas of concern within the Software reengineering field, this objective can be split as follows:

**Obj 1.2.**Detailed study of current initiatives in the scope of CMS-based Web applications focusing on the proposals addressing the development or migration of this type of Web applications.

**Obj 1.3.**Detailed study of current model-driven reengineering methods.

**Obj 2.** Specification of a metamodel for the CMS domain. This metamodel namely CMS Common Metamodel, should represent the key concepts of the CMS-based Web applications.

**Obj 3.** Specification of the *ADMigraCMS* method, presented in this PhD Thesis to carry out the migration of CMS-based Web applications. This objective can be split as follows:

**Obj 3.1.** Definition of the process of the *ADMigraCMS* method.

**Obj 3.2.** Definition of the modelling levels (Level 0, Level 1, Level 2 and Level 3).

**Obj 3.3.** Definition of the automated transformations between the modelling levels.

**Obj 4.** Specification of the modelling languages considered in the *ADMigraCMS* method. These modelling languages are defined in the form of Domain Specific Language (DSL) and are related to a modelling level.

**Obj 4.1.** Definition of the PHP DSL (Level 0).

**Obj 4.2.** Definition of the ASTM\_PHP DSL (Level 1).

**Obj 4.3.** Definition of the KDM\_CODE DSL (Level 2).

**Obj 4.4.** Definition of the CMS DSL (Level 3).

**Obj 5.** Specification of the automated transformations involved in the *ADMigraCMS* method.

**Obj 5.1.** Definition of the mappings between the PHP metamodel and the ASTM\_PHP metamodel.

**Obj 5.2.** Definition of the mappings between the ASTM\_PHP metamodel and the KDM metamodel.

**Obj 5.3.** Definition of the mappings between the KDM metamodel and the CMS Common Metamodel.

**Obj 6.** Implementation of the *ADMigraCMS* toolkit. This objective can be divided as follows:

**Obj 6.1.** Implementation of the metamodels that define the abstract syntax of the DSL.

**Obj 6.2.** Implementation of a graphical editor for each DSL.

**Obj 6.3.** Implementation of the automated transformations.

**Obj 7.** Validation of the *ADMigraCMS* method. We apply the *ADMigraCMS* method to two case studies to validate the methodological approach and the proper operation of the *ADMigraCMS* toolkit.

### 1.3 Research Context

In the last few years, the Kybele research group, in which the author of this dissertation is part of, has been working in the definition of MIDAS (Castro, Marcos, & Cáceres, 2004), (Vela, Fernández-Medina, Marcos, & Piattini, 2006), (Castro, Marcos, & López-Sanz, 2006), (Vara, Vela, Cavero, & Marcos, 2007), (López-Sanz, Acuña, Cuesta, & Marcos, 2008) an architectural-centric methodology for the model-driven development of IS; and recently, in the definition of reengineering methods such as PREMISA (Moratalla, 2012) and the *ADMigraCMS* method presented in this PhD Thesis, both based on ADM.

As part of MIDAS, a set of approaches have been defined, each of them focused on different aspects of the IS development: SOD-M (de Castro, 2007), a methodological approach based on MDA for service-oriented development; PISA (Acuña, 2007), an architectural approach for the integration of websites based on semantic Web services; MIDAS MDA Tool (M2DAT) (Vara, 2009) MDA tool supporting the methods proposed by MIDAS; MeTAGeM (Bollati, 2011) a meta-tool to semi-automate the development of model transformations for M2DAT; ArchiMeDeS (López-Sanz, 2011) a modelling framework for the specification of Software Architectures built upon the principles of the Service-Oriented Computing paradigm and MeTAGeM-Trace (Jiménez, 2012) a methodological approach to generate semi-automatically traces within a model-driven framework for developing model transformations.

As for the reengineering methods, PREMISA (Moratalla, 2012) is a process for the development and modernization of information systems, which is defined and validated by application to the Public Enterprise Red.es. The objective of this process is to update the business rules of the organization and the technology architecture. The key of this process is the model-driven modernization based on ADM, and the analysis of the gap between the business rule models at a high abstraction level.

### 1.3.1 Related Research Projects

As it is mentioned previously, the research conducted in this PhD Thesis has been carried out in the Kybele Research Group from Rey Juan Carlos University (URJC).

In this section, we introduce the research projects carried out during the research of this PhD Thesis. Most of them are national projects promoted and financed by public institutions.

As we can see in Figure 1-1, the research is framed in the context of two main national research projects, MODEL-CAOS and MASAI; and three national networks: Red Científico-Tecnológica en Arquitecturas y Desarrollo Orientado a Servicios (RedADOS), Red Científico Tecnológica en Desarrollo Industrial de Software (RedTDIS) and Red Científico-Tecnológica en Ciencias de los Servicios (RedCIS).

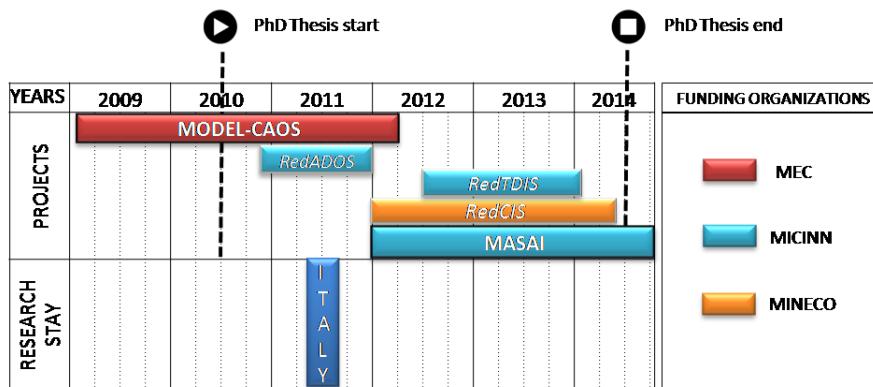


Figure 1-1. Research projects and research stays.

**MODEL-CAOS** [TIN2008-03582], was financed by the Ministry of Education and Culture of Spain, which started in 2009 and lasted for three years, ending in 2011. The main objective of this project was the specification of a framework for the semi-automatic development of information systems with a special attention to the use of Service Oriented Computing (SOC) (Papazoglou, 2003) as foundational paradigm. This last project inherited the work done in the previous projects and updated it by including the last trends in the development of information systems, such as the service-oriented paradigm, with an emphasis in the architectural aspect as central artefact guiding the methodological process. In the context of this project, the author of this PhD Thesis has defined a first

approach that allowed the specification and modelling of CMS-based Web applications within the MIDAS methodological framework.

**MASAI** [TIN-2011-22617] is financed by the Ministry of Science and Innovation of Spain, which started in 2012 and will last for three years, until the end of 2014. The main objective is to apply model-driven (MDE) development techniques to provide solutions to some of the problems found in the Engineering Services (IS), considering the knowledge and expertise gained in the context of previous research projects carried out by the Kybele Research Group. One of the main objectives of this project is the adaptation of legacy systems to modern service oriented systems. The author of this PhD has been working in the context of this project in the definition of the MDA-based modernization method.

It is worth noting that during this research, the PhD Student has taken part as a researcher in three research nets: the Red Científico-Tecnológica en Arquitecturas y Desarrollo Orientado a Servicios [TIN 2010-09669-E], financed by the Ministry of Science and Innovation of Spain, started in 2010 and lasting for one year, the Red Temática en Tecnologías para el Desarrollo Industrial de Software [TIN2011-15009-E] started in 2012 and financed also by the same ministry; and the Red Científico-Tecnológica en Ciencias de los Servicios [TIN2011-15497-E], financed by the Ministry of Economy and Competitiveness, started in 2012 and, again, lasting for two years.

The objective of these nets has been to ease the interchange and sharing of knowledge and experience between the different research groups composing the net. This cooperation has allowed it to refine the ADM-based method proposed in this PhD Thesis.

### 1.3.2 External Research Stay

A significant part of the research carried out to achieve the goals of this PhD Thesis was done during a 16-week period (from May, 2011 to September, 2011) working in the *Dipartimento di Elettronica e Informazione* at the *Politecnico di Milano* (Como – Italy), as we can see in Figure 1-1. This group is a specialist in the specification of formal approaches in the context of Model-Driven Engineering (MDE) (Stahl, Volter, & Czarnecki, 2006). They are the founders of *WebML* (Web Modelling Language) (Ceri, Fraternali, & Bongio, 2000), (Bozzon, Comai, Fraternali, & Toffetti, 2006a), (Bozzon, Comai, Fraternali, & Toffetti, 2006b), (Fraternali, Comai, Bozzon, & G., 2010), a DSL and a methodology for designing complex data-intensive Web applications. In 2013, *WebML* was adopted

as a standard by the Object Management Group (OMG, 1989). Moreover, they are the creators of the tool *WebRatio* (Piero Fraternali, 2000), (Acerbis, Bongio, & et al., 2004) which provides support for all the technical issues involved in the development of a Web application with *WebML*. During the last few years, *WebRatio* has consolidated its position in the software industry.

During this stay, and under the advice of Prof. Piero Fraternali, the PhD Student worked in the definition of a metamodel about the key concepts of CMS, in particular, the CMS Common Metamodel (Trias, 2012). This work was the first step in the construction of the *ADMigraCMS* method presented in this PhD Thesis.

## 1.4 Structure of the Dissertation

In order to have a better comprehension of the structure of the dissertation in relation to the work accomplished, Figure 1-2 dissects the anatomy of the proposal into its constituent parts and elements created for its completion. Associated to each building block is a reference to the chapter or subsection in which the issue is fully explained.

All in all, the remainder of this dissertation is organized in the following chapters:

- **Chapter 2** presents the **research method** chosen for this PhD Thesis. This research method is composed of a set of stages but we center the attention in three of them: the identification of the body of the knowledge (explained in Section 2.1), the definition of the working method (presented in Section 2.2) and finally the specification, implementation and validation stage (depicted in Section 2.4)
- **Chapter 3** provides a thorough review of the **state of the art**. Firstly, Section 3.1 presents some of the main previous concepts needed to approach the state of the art. In this chapter, we present two literature reviews to analyze the state of the art from two perspectives: 1) the first review (in Section 3.2) detects existing methods focused on the context of CMS-based Web applications and 2) the second review (in Section 3.3) finds existing reengineering methods based on ADM.

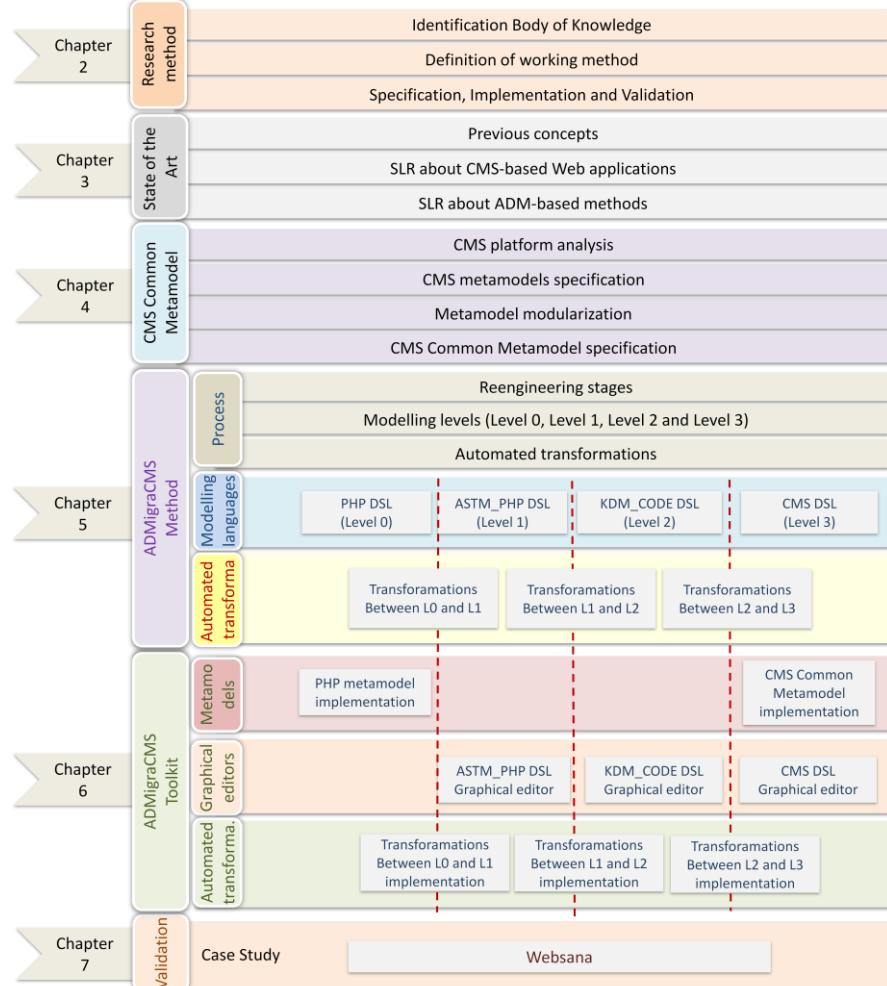


Figure 1-2. Anatomy of the proposal in relation to the structure of the dissertation.

- **Chapter 4** shows the process followed to define the **CMS Common Metamodel**, the cornerstone of the *ADMigracMS* method. This process is composed of four tasks: the CMS platform analysis (presented in Section 4.2), the CMS metamodels specification (explained in Section 4.3), the metamodel modularization (explained in Section 4.4) and the CMS Common Metamodel Specification (presented in Section 4.5). The CMS Common Metamodel defines the abstract syntax of the CMS DSL, a modelling language defined in the context of this PhD Thesis.

- **Chapter 5** explains in depth the foundations and main features of the *ADMigracMS method*. Section 5.1 presents the modelling levels (Level 0, Level 1, Level 2 and Level 3) and automated transformations that compose the *ADMigracMS* method. In Section 5.2.1, we define a PHP DSL. In Section 5.2.2, we define the ASTM\_PHP DSL to represent the models generated from the PHP code. To define this DSL we have defined the ASTM\_PHP metamodel and its graphical notation. Similarly, in Section 5.2.3 we present the KDM\_CODE DSL. This DSL is based on the action and code packages of the KDM metamodel. Moreover, we have defined the graphical notation for this DSL. In Section 5.2.4, we define the CMS DSL. This DSL is specified with the CMS Common Metamodel presented in Chapter 4 and a concrete graphical notation. Section 5.3.1 defines the automated transformations between Level 0 and Level 1 defining the mappings between the PHP metamodel and the ASTM\_PHP metamodel; Section 5.3.2 specifies the automated transformations between Level 1 and Level 2 specifying the mappings between the ASTM\_PHP metamodel and the KDM metamodel and finally, Section 5.3.3 defines the transformations between the Level 2 and Level 3 with the mappings between the KDM metamodel and the CMS Common Metamodel.
- **Chapter 6** shows the development of the *ADMigracMS toolkit*. In Section 6.1, we present the architecture of the *ADMigracMS* toolkit from the conceptual point of view and from the technical design point of view. In Section 6.2, we present the implementation of two of the metamodels (the PHP metamodel and the CMS Common Metamodel). Section 6.3 presents the implementation of the graphical editors for the ASTM\_PHP DSL, KDM\_CODE DSL and CMS DSL. Finally, Section 6.4 presents the implementation of the automated transformations.
- **Chapter 7** presents the **validation** of the work presented in this PhD Thesis according to the validation method presented in Chapter 2. In Section 7.4, we present the execution of the case study based on a CMS-based Web application called Websana for the management of a wellness and nutrition center. We migrate this Web application from Drupal to Wordpress.
- **Chapter 8** concludes with the main **conclusions and contributions** of this PhD Thesis. Moreover, **future research lines** derived from the work of this PhD Thesis are posed. Additionally, the contrasted results are presented in the form of the articles and paper published as result of this research work.

- **Appendices** we have defined a set of appendixes to extend the information presented in this dissertation. **Appendix A** presents a summary in Spanish of the main aspects of this PhD Thesis, **Appendix B** presents the complete process of the two Systematic Literature Reviews done to obtain the body of knowledge presented in the state of the art for this PhD Thesis, **Appendix C** presents the extended specification of the PHP metamodel, **Appendix D** shows the definition of the concrete syntax for the CMS DSL. We present a table with the CMS elements with their GMF annotations, EOL definitions and graphical notations, **Appendix E** presents the extended specification of the mappings between the modelling levels defined in the *ADMigraCMS* method, **Appendix F** presents part of the transformations considered during the Websana case study.
- **Bibliography and Online Resources** contains the bibliography and online resources.
- **Acronyms** contains the table of the most used acronyms in this dissertation.

## *Chapter 2: Research Method*

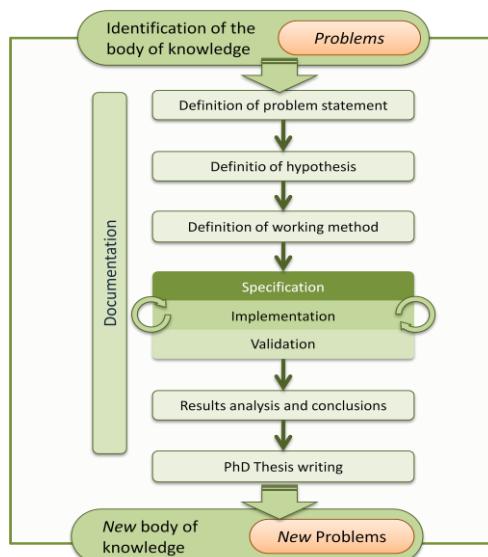


This Chapter presents the research method followed to carry out this PhD Thesis. Section 2.1 presents the stages composing this research method such as, the identification of the body of knowledge stage (presented in Section 2.2), the definition of the working method stage (explained in Section 2.3) and the specification, implementation and validation stage (explained in Section 2.4).

## 2.1 Strategy

The diverse nature of Engineering disciplines, among those considered empirical and formal, does not allow to directly apply classical research approaches to Software Engineering (Ian Sommerville, 2002).

The research method used in this PhD Thesis comprises an adaptation of the one defined by Marcos & Marcos (Marcos & Marcos, 1998) for investigation in the scope of Software Engineering. This method, in turn, is based on the hypothetic-deductive one by Bunge (Bunge, 1979) which is comprised of several stages, sufficiently general, to be applied to any kind of research activity. The main stages of the research process followed to complete the current PhD Thesis are depicted in Figure 2-1.



**Figure 2-1. Overview of the research method.**

The first stage of this method is the **identification of the body of knowledge** and a set of problems within this context required to be addressed. The result obtained is the state of the art of this PhD Thesis presented in Chapter 3; the theoretical basis on which we base our research.

The second stage, as we can see in Figure 2-1, is the **definition of the problem statement**. From the comprehensive *identification of the body of knowledge* and the potential problems discovered, we determine the problem to address. This stage has been depicted in Section 1.1.

The third stage is the **definition of the hypothesis**. After defining the problem, the hypothesis is formulated as well as a set of objectives which to fulfil to reach the solution. This solution must verify the hypothesis posed previously.

The fourth stage is the **definition of the working method**, which allows to carry out the specification, implementation and validation of the work of our research. This is a necessary aspect due to the adaptability of the method to different contexts. Each research project has its own intrinsic features and thus there is not a universal method to be applied to every kind of research.

The fifth stage is the corresponding to the **specification, implementation and validation**. This stage is of great interest as it represents the core of this PhD Thesis.

Once a proposal is derived from the previous research stages, it is the time to analyze the results obtained as consequence of the application of that proposal to concrete scenarios in a sixth stage called **result analysis and conclusions**. These analysis tasks serve to extract some conclusions from the work accomplished.

The seventh stage is the **PhD Thesis writing** stage to gather all this research experience in a final dissertation.

Though the final step of the research process may be embodied in the task of writing the PhD Thesis, any research activity establishes a new body of knowledge that is formed by incorporating all the research artefacts as part of it. This newly created situation derives new problems, **new body of knowledge**, that may be the object of future research tasks and initiatives.

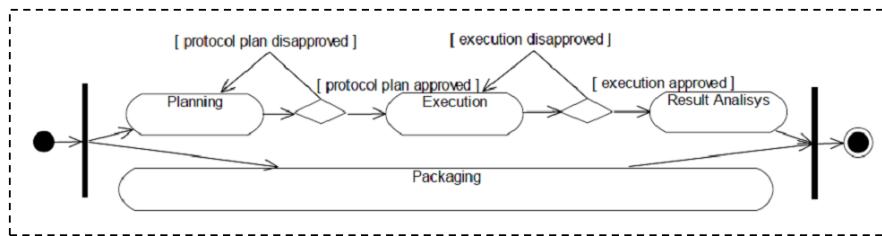
Three of the most relevant stages in the research method are: the **identification of the body of knowledge**, the **definition of the working method** and the **specification, implementation and validation** stage. According to their relevance, we explain them in the following sections.

## 2.2 Identification of the Body of Knowledge

As we explained in Section 2.1, the first stage of the research method is the identification of the body of knowledge. As we will see in Chapter 3, we use this body of knowledge as the base of the state of the art of this PhD Thesis.

To identify this body of knowledge, we have followed a Systematic Literature Review (SLR) based on the guidelines proposed by (Biolchini, Mian, Candida, & Natali, 2005) and (Barbara Kitchenham et al., 2009).

A SLR is a standardized and rigorous process to identify, assess and interpret all the information related to a research question (Biolchini et al., 2005), (B Kitchenham & Charters, 2007). One of the main motivations to perform a SLR is the huge amount of available information on, e.g. Software development and Software reengineering, due to the constant and quick evolution of technologies. Having so much information entails some drawbacks, such as to discard relevant information for the research. Unlike a traditional literature review, a SLR follows a strict and well-defined sequence of methodological phases ensuring the high scientific value of the results obtained (Biolchini et al., 2005). These methodological phases are explicitly defined, thus any researcher can reproduce them systematically as a protocol (Biolchini et al., 2005). The phases or the SLR followed in this PhD Thesis are shown in Figure 2-2.



**Figure 2-2. Sistematic Review process proposed by Biolchini et al. (Biolchini et al., 2005).**

As we can see in Figure 2-2, the SLR is composed of three consecutive phases (*Planning*, *Execution* and *Result Analysis*) and one in parallel (*Packaging*). In addition, we considered two control points (one defined among the *Planning* and *Execution* phases and the second one among the *Execution* and *Result Analysis* phases) which allow us to assess the correct performance of the SLR (Biolchini et al., 2005). The *Planning*, *Execution* and *Result Analysis* phases are explained in the following subsections. Figure 2-3 shows these phases and the tasks defined in them.

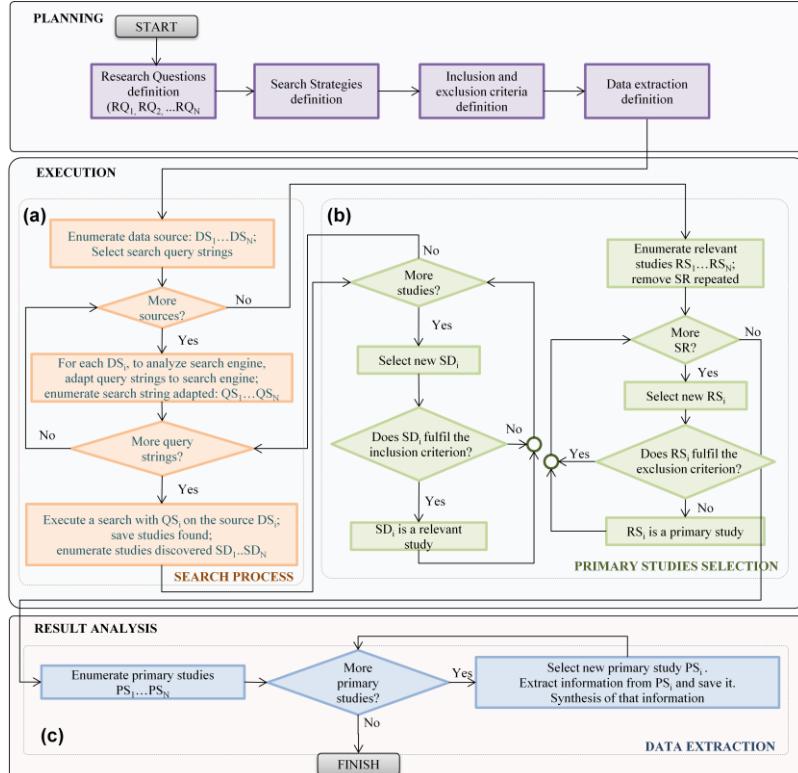


Figure 2-3. SLR phases applied to our research.

### 2.2.1 Planning Phase

The *Planning* phase aims for defining the objective of the SLR and the protocol conducting it (Biolchini et al., 2005). This protocol should specify the following information: **research questions definition**, which are answered with the information obtained in the SLR; **search strategies definition**, which define the sources and key terms to perform the SLR; **inclusion and exclusion criteria definition**, which filter the studies found in the SLR selecting those which are key and, finally, **data extraction definition**, which defines a set of criteria to extract the relevant information for our research (B Kitchenham & Charters, 2007).

### 2.2.2 Execution Phase

The *Execution* phase centres in the application of the protocol defined in the *Planning* phase. Therefore, it aims for the identification of the key studies for

our research. To define this protocol, we have adapted the process presented by Pino et al. in (Pino, García, & Piattini, 2007) which is mainly intended for searching in digital sources. The adapted process is basically composed of two tasks shown in Figure 2-3: *Search Process* (Figure 2-3.a) and *Primary Studies Selection* (Figure 2-3.b). In the following subsections we explain each of these tasks.

### **Search Process**

The first task in the search process is to enumerate the data sources ( $DS_i$  in Figure 2-3) from which the studies are retrieved; the second task is to define the query strings ( $QS_i$  in Figure 2-3), which will be launched in the data sources.

After enumerating the data sources and defining the query strings, we adapted these query strings to the syntax of each data source. Then, we start searching studies by launching the adapted query strings in the data source.

Once the study search has been executed, the studies discovered ( $SD_i$  in Figure 2-3) are enumerated and stored. To identify them more easily, we register their name and their authors before they are transferred to the *Primary Studies Selection* task.

### **Primary Studies Selection**

In this second task, we filter and select the most important studies discovered in the previous task for our research. To carry out this selection we use the two types of criteria defined during the *Planning* phase, the inclusion criteria and the exclusion criteria. Firstly, we apply the inclusion criteria, thus those studies meeting these criteria become **relevant studies**. Secondly, once we have identified all the relevant studies, we enumerate them ( $RS_1 \dots RS_N$  in Figure 2-3) and we eliminate the duplicated ones even they have been found in different data sources. Thirdly, we apply the exclusion criteria to discard uninteresting studies for our research defining the set of **primary studies** ( $PS_i$  in Figure 2-3). The primary studies are transferred to the *Data Extraction* task.

### **2.2.3 Result Analysis Phase**

The *Result Analysis* phase of the SLR involves writing up the results of the review and circulating the results to potentially interested parties.

According to Pino et al., this phase corresponds to the *Data Extraction* task of the protocol conducting the SLR, as we can see in Figure 2-3.c. The main

activities defined in this task are: the enumeration of all the primary studies and the extraction of the relevant information for our research by applying a set of criteria.

## 2.3 Definition of the Working Method

As we saw previously, the definition of the working method is the fourth stage in our research method. It defines the methods which allows to carry out the specification, implementation and validation stage. We consider the intrinsic features of our research to propose a development method to address the specification and implementation, otherwise one validation method to tackle de validation. These two methods are explained in next subsections.

### 2.3.1 Development Method

The resolution of this PhD Thesis involves the specification and implementation of a set of modelling languages. For the implementation of these modelling languages, we can opt for two approaches: using UML profiles or DSLs (Mernik, Heering, & Sloane, 2005), (Watson, 2008) and (Fowler, 2010). We opted for the use of DSLs considering the advantages that the use of DSLs provides over the use of UML profiles according to (Vara, 2009).

On the other hand, we define a development method for the implementation of the modelling languages as DSLs and the construction of automated transformations which allow to connect these DSLs. In this PhD Thesis, we have followed an adaptation of the method proposed by J.M. Vara (Vara, 2009). This method is the result of a comprehensive analysis of the state of the art about technological approaches supporting the model-driven development. The adaptation of this development method is shown in Figure 2-4.

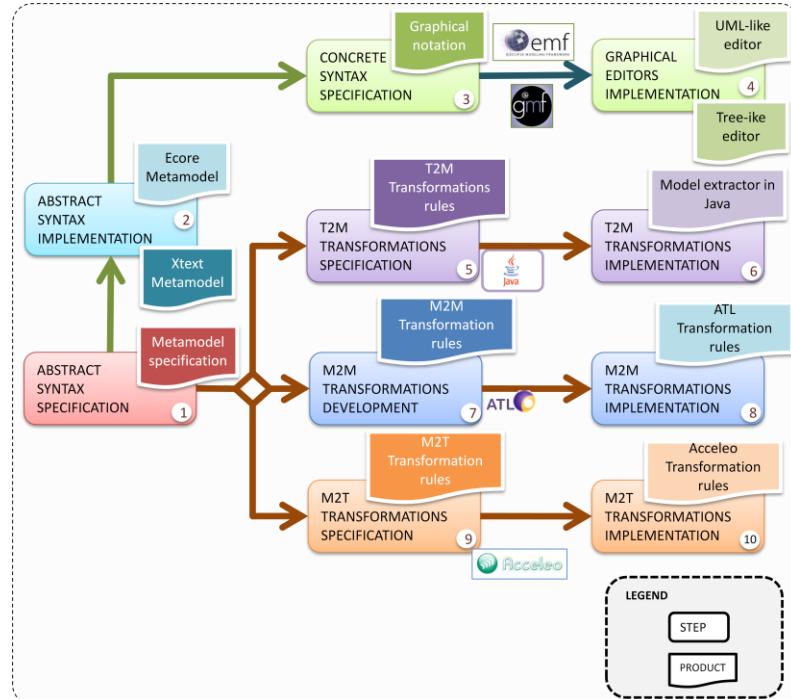


Figure 2-4. Development method.

As we can see in Figure 2-4, the development method is composed of ten steps interrelated. Below, we describe them:

1. **The definition of the abstract syntax** of the DSL which is specified as a metamodel. This metamodel captures the key elements of the modelling language and their relationships.
2. **The implementation of the metamodel** defined on the first step. It can be implemented with the metamodeling language Ecore by using Eclipse Modelling Framework (EMF) (Steinberg, Budinsky, Paternostro, & Merks, 2008), (Budinsky, 2008), (Gronback, 2009) or with the language Xtext by using the Xtext framework (German company Itemis, 2006), (Moritz & Behrens, 2010).
3. **The definition of the concrete syntax** of the DSL by defining a graphical notation. This graphical notation customizes and identifies uniquely each element captured in the metamodel.
4. **The implementation of graphical editors** for creating models conforming to the DSL. This graphical editor is based on the abstract syntax and the

concrete syntax we have defined previously. If the model is represented as a tree-like model, we implement the graphical editor by using EMF. Otherwise, if the model is represented as a UML model, we use the Graphical Modelling Framework (GMF) (Tikhomirov & Shatali, 2008) to implement the graphical model.

5. **The definition of T2M transformations** which allow to connect two DSLs. Concretely, those transformations centred in the transition from a grammaware to a modelware (Wimmer & Kramler, 2005) which are defined as a T2M transformation rules.
6. **The implementation of the T2M transformation rules** in the form of a model extractor implemented in Java.
7. **The definition of M2M transformations** which allow to connect two DSLs determined at modelware level. These transformations are specified as M2M transformation rules.
8. **The implementation of the M2M transformation rules** are with the Atlas Transformation Language (Czarnecki & Helsen, 2003), (Jouault, Allilaire, Bézivin, Kurtev, & Valduriez, 2006), (Jouault & Kurtev, 2006), (Jouault, Allilaire, Bézivin, & Kurtev, 2008), (Eclipse Foundation, 2013).
9. **The definition of T2M transformations** allowing the transition from modelware to grammaware. These transformations are specified as M2T transformation rules.
10. **The implementation of the M2T transformation rules** are by using Acceleo (Musset, Juliet, & Lacrampe, 2008).

The execution of this development method is presented in Chapter 5 and Chapter 6, where we explain in detail the steps taken for the specification and implementation of the *ADMigracMS* method and the toolkit that supports it.

### **2.3.2 Validation Method**

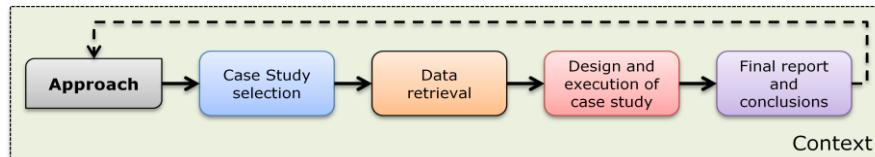
The scientific nature of Software engineering, like other disciplines such as physics, medicine or other engineering, requires of experimentation to verify and validate the proposed theories (Basili, Selby, & Hutchens, 1986b). Also, it is advisable to experiment with software techniques to know in which scenarios works correctly, which are its limitations and areas for improvement (Basili, 1996)

(Sjoeberg et al., 2005). To that end, Kish (Kish, 1959) defined three types of empirical research: surveys, experiments and simple research (case studies).

- **Surveys** are empirical research in which the subjects selected are a representative sample of the population under study (Pfleeger & Kitchenham, 2001).
- **Experiments** are research in which the possible disturbing variables have been randomized, but can be controlled (Basili, Selby, & Hutchens, 1986a).
- **Case studies** are those in which there is no randomness of those disturbing variables which cannot be controlled, and representativeness of the subjects composing the study sample (Yin, 2003).

Considering case studies allow to analyze the feasibility of an approach within real-world scenarios (Yin, 2003), in PhD Thesis we opt for this technique to validate our scientific proposal, to that end, we use two case studies for the validation of our *ADMigraCMS* method. One of the case studies is a simulated case study corresponding to a CMS-based Web application for a wellness and nutrition centre and the other one is a real-world case study related to a CMS-based Web application for a coaching centre.

According to the ideas posed in (Eisenhardt, 1989), (Yin, 2003), (Baxter & Jack, 2008), a validation by using case studies is defined as a process composed of four tasks. Figure 2-5 shows this process.



**Figure 2-5. Validation process by using case studies.**

- The **case study selection** task is the first task in the validation process where we define one or more case studies used to validate the approach. For the definition of these case studies, it is necessary to know thoroughly the approach as well as its context and the aspects to be analyzed (Baxter & Jack, 2008). Considering that covering all the possible scenarios for the approach is almost impossible, it is important to select comprehensive and representative case studies which provide us with interesting and valuable conclusions (Stake, 1995).

- The **data retrieval** task which consists in the elaboration of a protocol which provides with a global vision of the case study, the definition of a set of questions which allows the data retrieval and a guideline for documenting it (Yin, 2003). According to Yin, the data related to the execution of a case study can be retrieved from different sources: documentation, log files, meetings, direct observations, participant observations and physic artefacts, among others (Eisenhardt, 1989), (Baxter & Jack, 2008). To maximize the benefits provided by these sources, Yin recommends to follow three principles for the data retrieval: 1) to use multiple data sources confirming the same evidence, 2) to create a data base for the case study and 3) to keep a string of evidences so that an external observer is able to follow the track of the identified evidences in the case study.
- The **design and execution**, the design task provides with a global vision of the steps will be performed in the execution of the case study. As for the execution of the case study, it consists of performing the design of the case study and retrieving those data defined in the previous task which are required to carry out the next step (Yin, 2003).
- The **report and conclusions** task centres in elaborating a report related to the design and execution of the case study according to the guideline defined during the second step (data retrieval). To define the report of the case study we have to consider different aspects, such as the target audience, its scope and the type of the case study (Eisenhardt, 1989), (Stake, 1995), (Yin, 2003) (Baxter & Jack, 2008). Yin proposes six types of different structures to create the report: linear-analytic, comparative, chronological, theory-building, suspense and non-sequential (Yin, 2003), (Runeson & Höst, 2008). From the report, the results obtained during the design and execution task are analyzed. From this analysis we extract a set of conclusions which able to validate the previous proposal. If necessary, it is possible to correct and improve the weaknesses detected in the proposal by using the information extracted (Yin, 2003).

## 2.4 Specification, Implementation and Validation

The methods explained previously (development method and validation method) are applied in this specification, implementation and validation stage. They conform to an incremental and iterative process model. This stage iterates in the refinement of the elements defining the proposal and increments them by

completing the different steps. Moreover, each step provides a feedback to other related steps. A graphical overview of this process is shown in Figure 2-6. Figure 2-6.a shows the steps included in the specification task (specification of the DSLs and automated transformations); Figure 2-6.b presents the steps in the implementation task (implementation of the metamodels, the graphical editors and automated transformations); finally, Figure 2-6.c depicts the steps in the validation task (validated with two case studies).

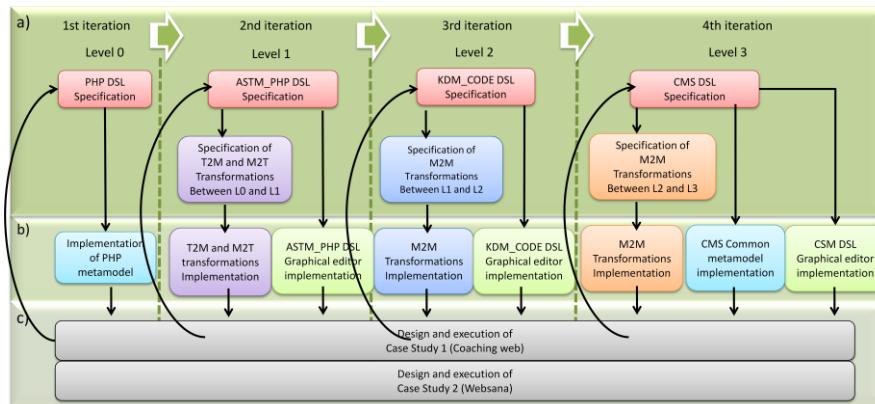


Figure 2-6. Definition of the a) specification, b) implementation and c) validation stage.

We chose this incremental and iterative process model because of the engineering nature of the problem to solve in this PhD Thesis. It is similar to those problems appearing in a real software development. The specification, implementation and validation do not represent a straightforward process, but an iterative one in which the information flows back and forth to improve each of the research tasks. As we can observe, this process is executed by four consecutive iterations. Below, we explain these iterations.

- **First iteration** centres its attention in the specification, implementation and validation of the PHP metamodel, the cornerstone of the PHP DSL defined in the modelling level *Level 0*. In the specification of the PHP metamodel, we capture the elements defining the syntax of this code. Finally, its validations is carried out by unit testing composed of PHP code fragments extracted from the two case studies defined for the validation of our research (Coaching Web and Websana).
- **Second iteration** is centred in the ASTM\_PHP DSL defined in the modelling level *Level 1*. In its specification we define its abstract syntax with the ASTM\_PHP metamodel. As for its concrete syntax, we define a

customized graphical notation for each element of this metamodel. From this DSL, we implement a tree-like graphical editor which allows to create and edit models conforming to this DSL. Moreover, we specify the automated transformations between the Level 0 and Level 1 (mappings between the PHP metamodel and the ASTM\_PHP metamodel). On the one hand, for the implementation of the T2M transformations between Level 0 and Level 1, we implement a model extractor. On the other hand, to implement the M2T transformations between Level 1 and Level 0, we implement a code generator. Finally, we validate the correct performance of the graphical editors and the proper execution of the automated transformations with the two case studies defined.

- **Third iteration** is focused on the KDM\_CODE DSL defined in the modelling level *Level 2*. We do not define any new metamodel for its abstract syntax, since we use the official implementation of KDM metamodel downloaded from (<http://adm.omg.org/>). As for the concrete syntax, we specify a customized graphical notation. For this DSL, we implement a tree-like graphical editor. We specify the M2M automated transformations between the Level 1 and Level 2 (mappings between the ASTM\_PHP metamodel and the KDM metamodel). Then, we implement the M2M transformations with a set of transformation rules. As occurred previously, the validation of the tree-like graphical editor and the M2M transformations are based on the two case studies.
- **Fourth iteration** is centred in the CMS DSL DSL defined in the modelling level *Level 3*. We define its abstract syntax with the CMS Common Metamodel. This metamodel is implemented. Moreover, we define the customized graphical notation of the elements of this metamodel. For this DSL, we implement a UML-like graphical editor. We specify the M2M automated transformations between the Level 2 and Level 3 (mappings between the KDM metamodel and the CMS Common Metamodel). Then, we implement these M2M transformations with a set of transformation rules. We validate the UML-like graphical editor and the M2M transformations are based on the two case studies.

## *Chapter 3: State of the Art*

---

In this Chapter, we present a comprehensive insight of the state of the art of the research addressed in this PhD Thesis. It is considered as a fundamental part of the dissertation as it helps to recognize the contributions that the PhD Thesis may add to the existent knowledge in the scientific area of influence and to position the proposal among others.

In Section 3.1, we provide the reader with the definition of the basic and main concepts to contextualize this PhD Thesis. Then, we explain the literature review we have performed to obtain all the relevant works that may have an influence in the scope of this PhD Thesis. As we explained in Section 2.2, to perform this literature review, we have followed a SLR process. In this Chapter we present the three tasks we have considered more relevant of this SLR, *research question definition task*, *data extraction definition task* and *data extraction task* (the complete SLR process is found in Appendix B). For this PhD Thesis, we have carried out two SLRs. In Section 3.2, we present the SLR which analyzes the existing approaches for the development or migration of CMS-based Web applications and in Section 3.3, we present the SLR focused on analyzing the existing model-driven reengineering approaches. Finally, in Section 3.4 we present the concluding remarks of this Chapter.

### 3.1 Previous Concepts

Throughout the history of Software Reengineering, it has been necessary to make great efforts in order to reach an agreement on the use of common terminology (Biggerstaff, 1989), (Chikofsky & Cross, 1990), (Arnold, 1993), (Müller et al., 2000). This problem is increased when it is a discipline such as Model-Driven Reengineering which are in the process of expansion and maturity (Wagner, 1959), (Rugaber & Stirewalt, 2004), (Fleurey, Breton, Baudry, & Nicolas, 2007), (Pérez-castillo, Garcia-Rodriguez de Guzman, & Piattini, 2012), (Wagner, 2014). Therefore, in the following sections, we present some of the definitions and previous concepts which are necessary to understand the body of knowledge of this PhD Thesis.

#### 3.1.1 *Model-Driven Engineering*

Model Driven Engineering (MDE) emerges as a natural step in the historical trend of increasing the abstraction level of the software development. In

particular, MDE focuses on the problem domain instead of the solution domain, the coding. Thus, models at different abstraction levels and model transformations are essential in the development process, becoming the basis of this paradigm (Selic, 2003), (J. Favre, 2004), (Bézivin, 2004b), (Schmidt, 2006), (Volter, 2011).

According to the final goal, it is possible to identify different disciplines which put into practice the MDE principles. Figure 3-1 presents a hierarchy of the different disciplines related to MDE.

On the left side of the Figure 3-1, we present those disciplines which apply the MDE principles to the *forward software engineering*: the Model-Driven Software Development (MDSD) (Stahl et al., 2006) and the Model-Driven Architecture (MDA) (Miller & Mukerji, 2003). On the right side, we present the disciplines which consider the MDE principles for the *software reengineering*: the Model-Driven Reengineering (MDRE) (Rugaber & Stirewalt, 2004) and the Architecture-Driven Modernization (ADM) (Khusidman & Ulrich, 2007). As it is marked in red in Figure 3-1, this PhD Thesis is framed in the context of MDRE and ADM. These two disciplines are explained in the following sections.

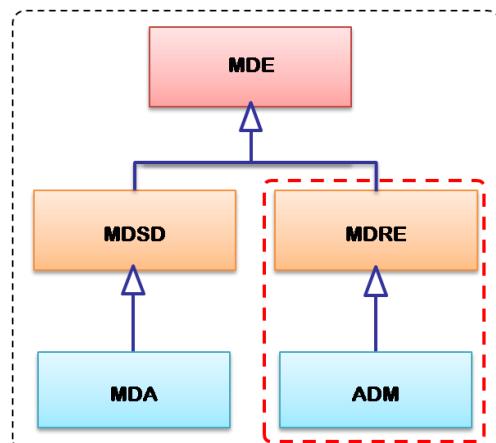


Figure 3-1. MDE disciplines.

### 3.1.2 Model-Driven Reengineering (MDRE)

The MDE's success and effectiveness to automate and formalize the top-down development, has expanded the MDE principles to the software reengineering. Thus, the Model-Driven Reengineering (MDRE) (Rugaber & Stirewalt, 2004), (Jean-marie Favre, 2004) has emerged as one of the most

effective approaches to automate the software reengineering process which consists of three stages (Chikofsky & Cross, 1990) presented briefly below:

- **Reverse engineering stage:** is the first stage in the software reengineering process. It analyzes the legacy system in order to identify the components of the system and their interrelationships. Then, the reverse engineering stage builds one or more representations of the legacy system at a higher level of abstraction.
- **Restructuring stage:** this stage takes the previous system's representation and transform it into another one at the same abstraction level. This stage preserves the external behaviour of the legacy system.
- **Forward engineering stage:** this stage generates the physical implementations of the target system at a low abstraction level from the previous restructured representation of the system.

As we mentioned in Section 1.1.1, the whole reengineering process can be viewed as a **horseshoe process**. The *reverse engineering stage* that increases the abstraction level represents the left side of the horseshoe; the *restructuring stage* preserves the same abstraction level, thus it represents the curve of the horseshoe; and the last part on the right side of the horseshoe is the *forward engineering stage* that instantiates the target system. Figure 3-2 shows graphically the horseshoe process considering the three reengineering stages.

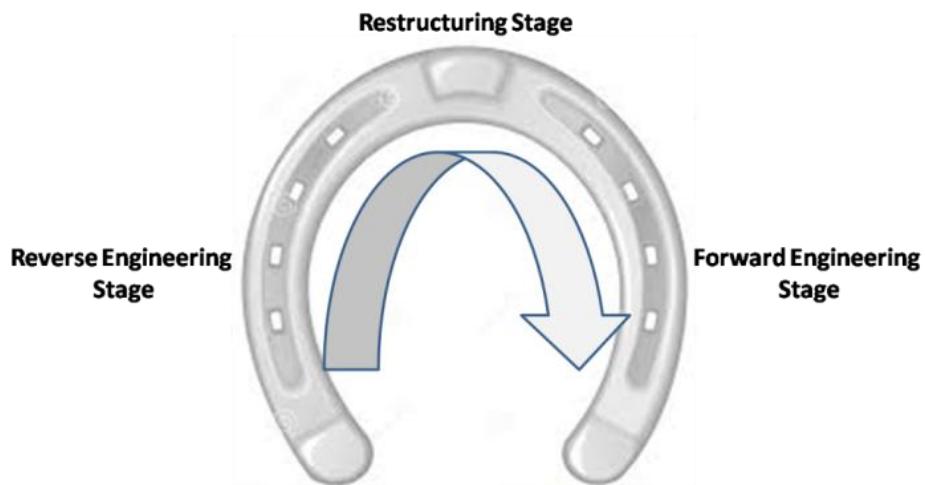


Figure 3-2. Reengineering stages.

Based on this idea, OMG proposes the Architecture-Driven Modernization (ADM), explained in the following subsection.

### 3.1.3 **Architecture-Driven Modernization (ADM)**

Architecture-Driven Modernization (ADM) (OMG, 2009) is an initiative proposed by the OMG to apply the MDRE principles to the model-driven reengineering process. Concretely, it is based on Model-Driven Architecture (MDA) (OMG, 2001), (Miller & Mukerji, 2003). Thus, ADM follows the following principles: i) modelling all the artefacts of a legacy IS as models at different abstraction levels (explained below); and (ii) establishing transformations to automate the reengineering process.

The abstraction levels used to specify these models are defined by MDA. We explained them in the following:

- **Computation-Independent Model (CIM):** it is the most abstract modelling level, which represents the context, requirements, and purpose of the solution without any binding to computational implications. It presents exactly what the solution is expected to be, but hides all the technological specifications, to remain independent to the implementation. The CIM is often referred to as a business model or domain model because it uses a vocabulary that is familiar to the subject matter experts.
- **Platform-Independent Model (PIM):** it is the level that describes the behaviour and structure of the application, regardless of the implementation platform. PIM is only for the part of the CIM that will be solved using a software-based solution and that refines it in terms of requirements for a software system. The PIM exhibits a sufficient degree of Independence so as to enable its mapping to one or more concrete implementation platforms.
- **Platform-Specific Model (PSM):** even if it is not executed itself, this model must contain all required information regarding the behaviour and structure of an application on a specific platform that developers may use to implement the executable code.

Considering the principles of ADM, we can contextualize the three stages involved in a reengineering process (*reverse engineering*, *restructuring* and *forward engineering*), as we can see in Figure 3-3. It shows the abstraction levels considered for each stage and the automated transformations defined to pass from one stage to another.

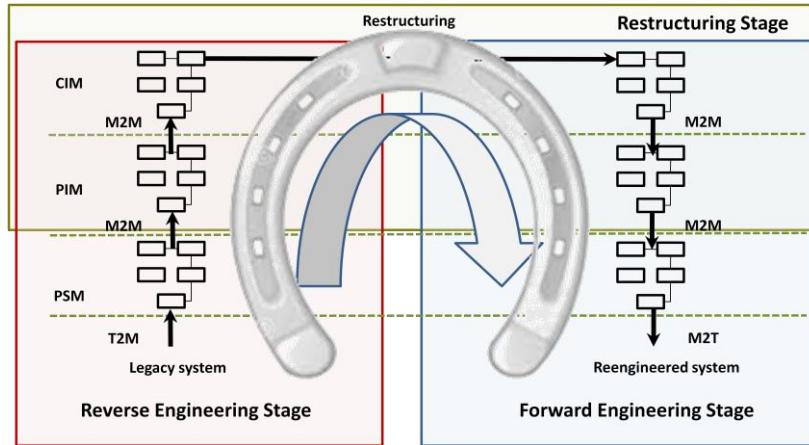


Figure 3-3. Reengineering stages in the context of ADM.

In the *reverse engineering stage*, the information is extracted by automated T2M transformations from software artefacts of a legacy IS. This information is represented usually in models defined at PSM level. Then, we obtain models at a higher abstraction level by means of automated M2M transformations. These models can be defined at PIM or CIM level.

In the *restructuring stage*, at this stage it is possible to include adjustments to the new platform in the models. For that, it is possible to use refactoring techniques, restructuring transformations or they can be included manually. Usually, the models restructured are defined at PIM or CIM level.

Finally, in the *forward engineering stage*, at this stage the abstraction level of the models is decreased by using M2M transformations to obtain a model from which we generate the code implementing the reengineered system. Usually, the code generation is performed from a model at PSM level by using M2T transformations.

Apart from the model-driven reengineering process propose, ADM provides with a set of standard metamodels which allow to represent in a formalized manner the information involved in the software reengineering process. Currently, the three available standard metamodels are (Brambilla, Wimmer, & Cabot, 2012): *Abstract Syntax Tree Metamodel* (ASTM), *Knowledge Discovery Metamodel* (KDM) and *Software Measurement Metamodel* (SMM). In next sections, we focus on the ASTM and KDM metamodels which are used in the *ADMigraCMS* method.

### **The Abstract Syntax Tree Metamodel (ASTM)**

ASTM is the standard metamodel representing a low-level view of the system. It allows to represent the source code of a system as an Abstract Syntax Tree (AST) (Fischer et al., 2007). Apart from allowing the representation of the syntax it allows to represent a basic semantic of this code by defining references and scopes (OMG, 2005a).

ASTM improves the interoperability between the approaches that generate ASTs from code since it allows to model the code by following a common formalism. This metamodel is considered as a standalone standard but it is complementary to KDM. Together, ASTM and the KDM provide high fidelity support for ADM scenarios.

ASTM is divided into two domains: the **Generic Abstract Syntax Tree Metamodel** (GASTM) and the **Specific Abstract Syntax Tree Metamodel** (SASTM). The former establishes a common core for language modelling gathering the common elements of the different existing programming languages and the latter represents the specific elements of a concrete programming language such as PHP, Java or SQL and allows the extension of the GASTM.

The elements from the GASTM are considered platform-independent, so that the models conforming to it are defined at PIM level. Otherwise, the elements from the SASTM are platform-dependent and the models conforming to this domain are considered PSM models.

For this PhD Thesis we extend the ASTM with specific elements of the PHP programming language which have been included within the domain of the SASTM. This extension of the ASTM is called **ASTM\_PHP** which is considered at PSM level.

### **The Knowledge Discovery Metamodel (KDM)**

KDM is the standard metamodel to define PIM models which represent the syntax and semantics of a system, ranging from source code to higher-level abstraction such as GUI events, platforms or business rules. It provides a common interchange format intended to represent existing software assets, allowing tool interoperability. KDM is composed of twelve packages (e.g. core, kdm, source, code or action) grouped in four layers to improve modularity and separation of concerns (infrastructure, program elements, runtime resource and abstractions). As we will see in Chapter 5, we use the packages code and action to define the

KDM\_CODE DSL. Figure 3-4 shows the packages and concerns conforming the KDM metamodel.

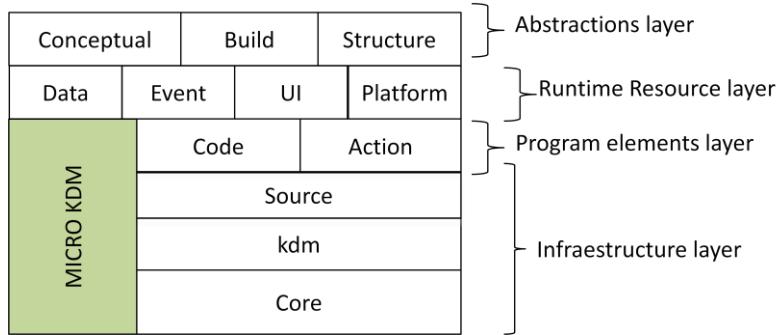


Figure 3-4. KDM packages and layers.

As for the packages defined at the **infrastructure layer**, we can say that the *code* and *kdm* packages are the basis of the rest of KDM packages since they define the generic elements. On the other hand, the *source* package allows us to depict the physical structure of the CMS-based Web application project in terms of directories and files. The model obtained from the *source* package is the *Inventory Model*.

As for the packages defined at the **program elements layer**, we can say that the *code* and *action* packages represent the syntax and semantics of the elements that compose the source code of the target CMS-based Web application and their relationships among them at PIM level. The resulting model is called Code model which has been taken into account in our method. We defined the packages included in this layer.

- **Code package** is composed of elements allow to represent the structure and syntax of the source code at a logical level.
- **Action package** is composed of a set of classes which allow the representation of behaviour descriptions and control-and-data-flow relationships between the code elements.
- **Micro KDM package** is a package defined between the infrastructure layer and program elements layer. It allows the refinement of the semantic of the action elements of the action package.

The packages belonging to the **runtime resource layer** enable the representation of high-value knowledge about legacy systems and their operating environment not contained within the code itself. We present each package:

- **Data package** allows the representation of the structure of complex data repositories, such as relational or XML databases. As for the relational databases, it is possible to define its tables and views. Otherwise, it is possible to define the XML schema of XML databases.
- **Event package** allows to model the dynamic aspect of a legacy system by representing its state and its state transitions by using an Event model. The extraction of knowledge from a legacy system can be performed by following a static or dynamic point of view. The former is focused on analysing the syntax of the source code and the later analyzes this source code at runtime.
- **UI package** defines a set of metamodel elements whose purpose is to represent facets of information related to user interfaces, including their composition, their sequence of operations, and their relationships to the existing software systems (Pérez-Castillo, de Guzmán, & Piattini, 2011).
- **Platform package** defines a set of elements whose purpose is to represent the runtime operating environments of existing software systems. Application code is not self-contained as it is determined not only by the selected programming languages, but also by the selected runtime platform. Platform elements determine the execution context for the application.

The packages of the **abstraction layer**, define a set of elements whose purpose is to represent domain-specific knowledge as well as to provide a business overview of the legacy information system. Below, we present them:

- **Structure package** defines a set of elements to represent the architectural components of a legacy system like subsystems, layers, packages and so on by using the Structural model.
- **Conceptual package** represents the domain-specific information of a legacy system in a Conceptual model, such as roles played by participants or some behaviour of the legacy system (Pérez-Castillo, de Guzmán, et al., 2011).
- **Build package** defines the artefacts related to the engineering view of a legacy system such as development activities or deliverables generated by the build process. By using this package, it is possible to generate the Build model.

### 3.1.4 *Modelling languages - Domain-Specific Languages (DSL)*

Modelling languages allow the construction of models in the MDE context. As for our *ADMigraCMS* method, we define graphical modelling languages to define the models consider. These modelling languages are defined as Domain Specific Languages (DSLs). These DSLs are tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use. Each domain has a set of specific features that distinguish it from the others and it has to be considered when specifying the problem. Hence, there is a need for Domain Specific Modelling (DSM) (S. Kelly & Tolvanen, 2008) in order to provide a vocabulary for each domain. This way, the problem could be expressed in a complete and reliable manner in order to automatically generate the solution.

The first step towards the definition of a DSL is the specification of its metamodel. In essence the metamodel collects the **abstract syntax** of the language, that describes the elements provided by the language and how they may be combined to create models. On the other hand, the **concrete syntax** provides the visual features of the elements of the metamodel that facilitates the presentation and construction of models by using the DSL.

There are two main types of concrete syntaxes typically used by languages: **textual syntax** and **graphical syntax**. That is, a model can be expressed using a textual notation (like coding a program) or a graphical notation, the most common being diagrams and tree-like representations. In fact, you can define several concrete syntax for the same abstract syntax, i.e. the set of concepts collected in a particular modelling language might be expressed using several notations. For instance, you may opt for a nodes & edges notation to provide an overview of the model and a textual one to provide a more detailed view.

We can identify two different trends in the use of DSLs. On the one hand, we can find DSLs defining modelling languages at a high abstraction level (Volter, 2009) which are closer to the problem domain than to the implementation domain. On the other hand, we can make a simile between DSLs and programming languages. A DSL might be shown as the grammar of the language and a model expressed with such DSL would be a program written in the corresponding grammar.

In this PhD Thesis, we have defined four DSLs: PHP DSL, ASTM\_PHP DSL, KDM\_CODE DSL and CMS DSL. The definition and implementation of these DSLs are explained in Chapter 5 and Chapter 6.

### 3.1.5 Content Management Systems

As we explain in the Introduction of this dissertation (Section 1.1.2), the CMS-based Web applications have become the most efficient tool allowing to collect, manage and polish huge amounts of digital content in a robust and reliable manner (Boiko, 2001). In this section, we present the concepts related to the Content Management (CM) to understand clearer the context of this type of Web applications.

Boiko (Boiko, 2001) defines the CM as *the process behind matching what “you” have with what “they” want. “You” are an organisation with information and functionality of value. “They” are a set of definable audiences who want that value.* CM can be seen as an overall process for collecting, managing, and publishing content:

- **Collection**, you either create or acquire information from an existing source. Depending on the source, you may or may not need to convert the information to a master format (such as XML). Finally, you aggregate the information into your system by editing it, segmenting it into chunks (or components), and adding appropriate metadata.
- **Management**, you create a repository that consists of database records and/or files containing content components and administrative data (data on the system's users, for example).
- **Publishing**, you make the content available by extracting components out of the repository and constructing targeted publications such as Web sites, printable documents, and e-mail newsletters. The publications consist of appropriately arranged components, functionality, standard surrounding information, and navigation.

Some form of CM is essential for organisations with a significant Web presence and a huge amount of digital content to maintain (Corporation, 2001). The CM in the Web context is so-called Web Content Management (WCM). McKeever defines WCM as *the process that considers the activities involved in the creation and deployment of digital content to Web based audiences, where these audiences may consist of customers, suppliers, partners and staff accessing Web content via extranet, Internet or intranet.* To support WCM activities there are software tools called Content Management Systems (CMS). The Web applications based on these CMS are so-called CMS-based Web applications.

McKeever depicts graphically the context of CMS-based Web applications using a four-layer hierarchy shown in Figure 3-5.

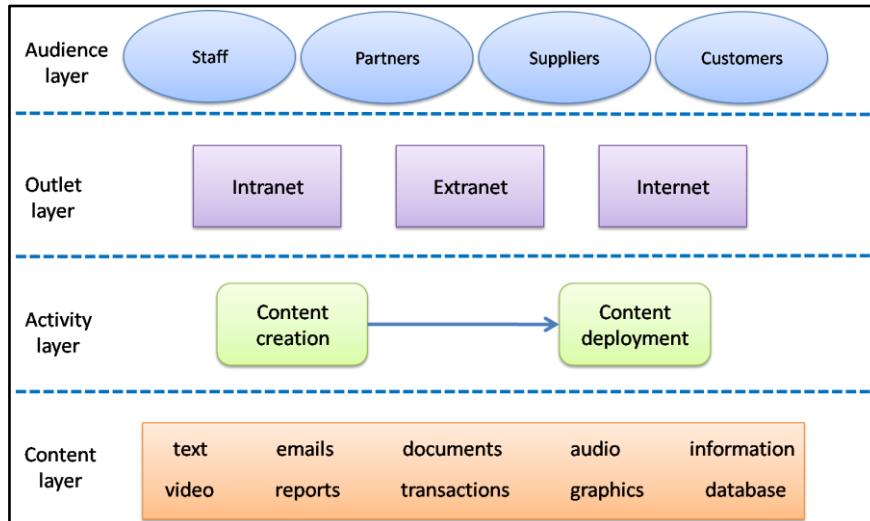
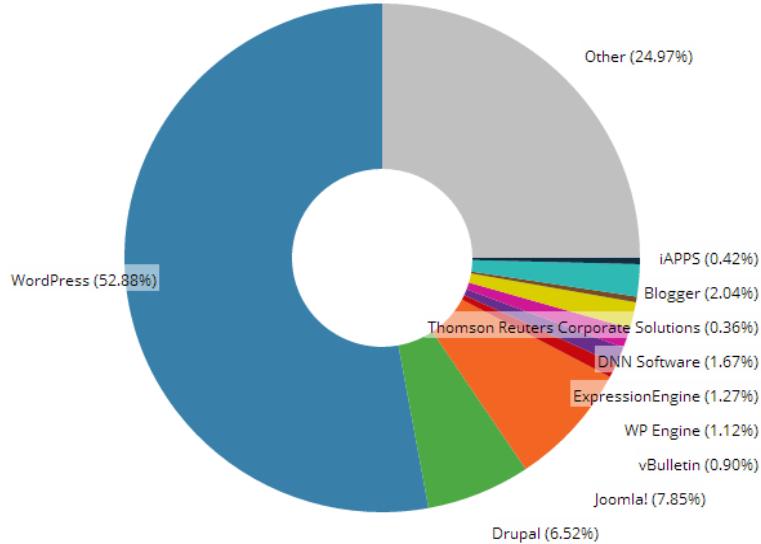


Figure 3-5. WCM four-hierarchy.

- **Audience layer:** this layer defines the groups of people who interact with the CMS-based Web application.
- **Outlet layer:** this layer defines the type of Web outlets through which content can be accessed: intranet, extranet or internet.
- **Activity layer:** this layer represents at a high level the activities involved in managing the content layer: the content is created and then deployed.
- **Content layer:** this layer consists of the types of content that require to be managed by CMS-based Web applications.

In the last decade, the number of available CMS in the market has growth very rapidly meeting different domains, such as blogs, e-commerce or e-learning (Shreves 2011). It is possible to find open-source CMS platforms such as Wordpress (Wordpress, 2014), Drupal (Drupal, 2014) and Joomla! (Joomla!, 2014) as well as proprietary ones, such as Ektron (Ektron & NH, 2001), SDL ACM (ACM, 2014), Content Verse (Verse, 2014) or Cascade Server (Hill, 2014). The use of open-source CMS is more spread. The chart presented in Figure 3-6 (BuiltWith, 2013) shows Wordpress (55.88%), Drupal (6.52%) and Joomla! (7.85%) as the most used open-source CMS platforms. It is worth noting that these CMS platforms are implemented in PHP. For this reason, up to now the proposal

of this PhD Thesis on migrating CMS-based Web applications to other CMS platforms is focused on open-source CMS platforms implemented in PHP.



**Figure 3-6. Chart about the CMS market (BuiltWith, 2013).**

### 3.2 Approaches in the Context of CMS-based Web Applications

In the last years the Web Engineering community has proposed new methods and methodologies to support the design and development of Web applications (Pressman, 1996), (Overmyer, 2000), (Lowe & Henderson-Sellers, 2001), (Gnaho, 2001), (Hoick, 2003), (Lee & Shirani, 2004), (van Berkum, M. Brinkkemper & Meyer, 2004). Some of these Web Engineering methods are WebML (Ceri et al., 2000), (Ceri, Frernali, Bongio, Brambilla, & Matera, 2003); OOH (Gómez, Cachero, & Pastor, 2000); OOWS (Valverde, Valderas, & Fons, 2007); UWE (Kraus, Knapp, & Koch, 2007), (Kroiss & Koch, 2008); OOHDMD (Schwabe & Rossi, 1998); HM<sup>3</sup> (Cáceres, de Castro, Vara, & Marcos, 2006) or W2000 (Baresi, Garzotto, & Paolini, 2001). All of them propose to systematize the complex task of a top-down Web application development by applying the MDSD perspective by using models and automated transformations.

The development process of CMS-based Web application is complex and time consuming. There are several reasons to substantiate this. First, CMS-based Web applications often involve customizations and integration with back office

systems. Secondly, these Web applications have a collaborative aspect since multiple users from different departments of the enterprise work simultaneously on the same content and functionality. A third reason is that CMS-based Web applications presents information over multiple channels (web, mobile, e-mail, print) for different purposes (sales, marketing, e-business, services, questionnaires, etc.), often in a personalized context. And fourth, implementing CMS-based Web applications is not just about the technology, but also about people and processes and therefore involve change management (Souer, Honders, Versendaal, & Brinkkemper, 2007), (Souer, Honders, Versendaal, & Brinkkemper, 2008), (Souer, Luinenburg, Versendaal, Weerd, & Brinkkemper, 2008).

Unfortunately, the existing Web Engineering methods do not cover the issues that arise when developing CMS-based Web applications (Souer, Honders, et al., 2007), (Souer, Weerd, et al., 2007), (Souer, Honders, et al., 2008), (Souer, Luinenburg, et al., 2008), (Souer & Mierloo, 2008), (Vlaanderen, Valverde, & Pastor, 2008), (Vlaanderen, Valverde, & Pastor, 2009), (Souer & Kupers, 2009), (Souer et al., 2009), (Souer & Joor, 2010), (Souer & Joor, 2010) and (Souer, Urlings, Helms, & Brinkkemper, 2011). Implementing a CMS-based Web application is different from implementing a traditional Web application or another information system (Weerd, Brinkkemper, Souer, & Versendaal, 2006). For instance, the development of a traditional Web application starts from scratch; however, the development of a CMS-based Web application starts from a preconfigured platform.

In this section, we carry out a literature review to identify and analyze those existing approaches that address the top-down development of CMS-based Web applications, as well as to identify those approaches addressing a reengineering process to migrate this type of Web applications. As we explained in Section 2.2, we followed a SLR to perform this literature review. The results of this SLR have been published in (Trias et al., 2013a).

From this SLR, we present the three most relevant tasks. In Section 3.2.1, we define the research questions which guide the SLR. In Section 3.2.2, we define the criteria used to extract the data from the existing approaches, and in Section 3.2.3, we present the analysis of each approach. Finally, in Section 3.2.4 we present the conclusions extracted from this SLR.

### ***3.2.1 Research Questions Definition***

In this Section, we define the Research Questions (RQ) that lead this SLR. These RQs are answered in Section 3.2.4.

- **RQ1. Which approaches do exist in the literature for addressing a development process of CMS-based Web applications?**

We are interested in finding all the existing methodological approaches within the scope of CMS-based Web applications. One branch of this scope is the Web Engineering methods following a typical top-down process for the development of CMS-based Web applications.

- **RQ2. Which approaches do exist in the literature for addressing a reengineering process for CMS-based Web applications?**

To complete the analysis of the scope of the CMS-based Web applications we try to find the existing reengineering methods focused on the automation and systematization of the migration of this type of Web applications.

- **RQ3. Has any existing method been adapted to the development of CMS-based Web applications?**

To date, the research field of Web Engineering has derived in many proposals supporting the complex tasks of designing and creating Web applications (Schwabe & Rossi, 1998), (Gómez et al., 2000), (Baresi et al., 2001), (Ceri et al., 2003), (Valverde, Valderas, & Fons, 2007), UWE (Kraus et al., 2007). Otherwise, the development of CMS-based Web applications entails complex and time consuming tasks which cannot be addressed by these current Web Engineering methods (Weerd et al., 2006), (Souer, Honders, et al., 2007), (Souer, Honders, et al., 2008), (Souer & Joor, 2010). Therefore, with this research question we are interested in identifying those Web Engineering methods that, undergoing some adaptations, have been capable to cope with the particularities of a CMS-based Web application.

- **RQ4. Which approaches can be framed within the context of Model-Driven Engineering (MDE)?**

Nowadays, there is an increasing tendency of traditional Web Engineering methods towards the use of a model-driven paradigm because of its benefits (Schmidt, 2006). Because of that, we are interested in testing if the approaches found in this SLR follow this paradigm. Our intention is to analyze aspects such as the processes proposed, the models defined, the transformations considered, the tools provided, the automatic generation of code, etc.

- **RQ5. Which is the interest of the Web Engineering community in researching about CMS-based Web applications?**

We consider the CMS-based Web applications a relevant topic to research because of its large acceptance and use in the market. Thus, we are interested in knowing the impact and the interest of this topic for the research community. Concretely, we want to know which research groups or organizations are leading the research in this area and which forums publish studies focused on this issue.

### **3.2.2 Data Extraction Definition**

Once we select the studies that allow us to answer the RQs, we define a set of criteria as a mechanism for extracting data from the different approaches found with the SLR. The extracted data will allow us to reach the conclusions presented in Section 3.2.4. Below we present the seven criteria defined for this goal.

- The **type of process** allows us to identify if the process defined by the approach is a development process or a reengineering process. The development process is characterized by starting with the definition of the IS at a high abstraction level and finishing with the generation of code at a low level. The reengineering process is defined as a horseshoe process which starts extracting knowledge from a legacy IS and finishes generating code for a target IS.
- The **Web Engineering views** allows us to know how complete is the method considering the coverage of the Web Engineering views. These views according to (Ginige & Murugesan, 2001a), (Ginige & Murugesan, 2001b), (Murugesan, Deshpande, Hansen, & Ginige, 2001), (Deshpande & Hansen, 2001), (Deshpande et al., 2002), (Ginige, 2002), (Kappel, Pröll, Reich, & Retschitzegger, 2006) are: 1) **content view**, is the view used to represent the business and data objects; 2) **navigation view**, is the view which expresses the composition of the interface in terms of container and the navigation map of the Web application; 3) **process view**, is the view that defines how the application reacts to the events raised by the user's navigation and 4) **presentation view**, is the view for specifying the layout and the look & feel of the interface. Furthermore, to complete this analysis we have considered two views belonging to the traditional Software Engineering (Pressman, 1996), (Ian Sommerville, 2002): the **requirements view** which is the view that reflects the functionality that Web applications offer to users and the

***implementation view*** which expresses Web applications in terms of artefacts implemented by code.

- The **models considered** allows us to know whether the approach uses models as a key artefact to carry out the development or reengineering process. With this criterion we can know if it meets with the MDE principles.
- The **MDA abstraction levels** allows us to know if the models used in the development/reengineering process are defined at some of the abstraction levels proposed by related by MDA (CIM, PIM, PSM). Thus, we can know if the method meets with one of the MDA principles.
- The **modelling languages** is related to the modelling language used to define the models. We consider interesting to know if the approach is supported by a standard modelling language such as UML or, otherwise, the approach uses its own DSL.
- The **automation level** refers to the level of automation of the approach. It allows us to know which type of automated transformation (T2M, M2M, M2T) are considered by the approach. If the approach automates all the development/reengineering process, it has a high level of automation. If the approach just automates some of the steps of the process, we consider it with a medium level of automation. Finally, if the approach does not consider any type of automated transformation it is considered to have a low level of automation.
- The **supporting toolkit** this criterion refers to the toolkit supporting the methodological approach. This toolkit can automate the T2M, M2M and M2T transformations as well as to assist the definition of the models considered within the approach.

### 3.2.3 Data Extraction

In this section, we present the analysis and the data extraction from the approaches considering the criteria defined in the previous section. We analyze the five approaches defined from the 15 primary studies obtained with this SLR. To know how we obtained the 15 primary studies is convenient to read the complete SLR process presented in Appendix B. Table 3-1 sums up the approaches presented in this section and the data extracted for each of them.

### 3.2.3.1 Souer et al. (Web Engineering Method - WEM)

The studies (Weerd et al., 2006), (Souer, Weerd, et al., 2007), (Souer, Honders, et al., 2007), (Souer, Luinenburg, et al., 2008), (Luinenburg, Jansen, Souer, & Weerd, 2008), (Souer, Honders, et al., 2008), (Souer et al., 2011) present the *Web Engineering Method* (WEM) a development process to implement CMS-based Web applications. This approach consists of six phases (Kappel, Präull, Reich, & Retschitzegger, 2006): Acquisition, Orientation, Definition, Design, Realization and Implementation. 1) the **Acquisition phase** focuses on outlining the customer's wish into a proper solution; 2) the **Orientation phase** defines the organizational aspects, such as participants, targets, products, scope and assumptions; 3) the **Definition phase** the analyst carries out the requirements analysis and the specification of the Web application (goals, scope, limiting conditions); 4) the **Design phase** determines how the requirements are realized. Based on the requirements, a suitable architecture is created; 5) the **Realization phase** creates the Web application and the graphical user interface design is integrated in the CMS-based Web application. Furthermore, the relevant functions in the Web application are configured to meet the customer's requirements; and 6) the **implementation phase** deploys the CMS-based Web applications to production.

WEM is adapted to three different types of project: standard, complex and migration projects. For each type, a *route* is defined including a set of activities. This way, the *standard route* addresses projects which are mostly based on existing standard functionalities of CMS-based Web applications. On the other hand, the *complex route* addresses projects based on existing functionalities with a significant amount of customization or build new functionalities. Finally, the *migration route* addresses migration projects that involve an upgraded of an older version of a CMS-based Web application to a newer version.

This method is defined following the Situational Method Engineering (SME) approach (Ralyté, Deneckère, & Rolland, 2003) that consists of taking parts of other existing engineering methods and customizing a new one for a certain domain. The two engineering methods on which the WEM method is based are: the UML-based Web Engineering (UWE) (Kraus et al., 2007) and the Unified Process (UP) (Jacobson, Booch, & Rumbaugh, 2000).

In the following, we analyze the proposal by Souer et al. according to the criteria defined in Section 3.2.2.

- **Type of process.** WEM method proposes three routes. Two of these routes correspond to a development process and one to a reengineering process. The studies found present the routes proposing development processes, leaving aside the reengineering process. Therefore, up to now, we considerer WEM as a method proposing a development process.
- **Web Engineering views.** WEM covers five Web Engineering views (requirements, process, navigation, presentation and implementation views) as we can see in Figure 3-7. The only view not addressed is the content view. The requirements view is addressed by the Acquisition, Orientation and part of the Definition phases. The presentation and navigation views, in turn, are addressed by the Definition phase, whereas, the process view is addressed by the Design phase. Finally, the implementation view of the CMS-based Web application is addressed during the Realization and Implementation phases.

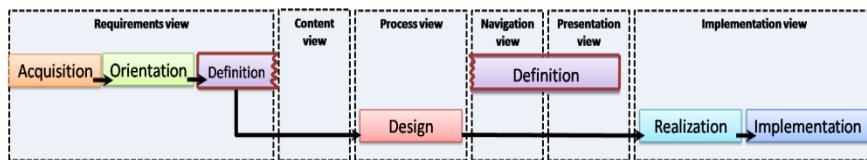


Figure 3-7. Web Engineering Method and Web engineering views.

- **Models considered.** WEM considers models as a key artefact for the development of CMS-based Web applications. WEM stands on two existing engineering methods, UP and UWE. The former does not follow the MDE principles, but considers UML (OMG, 2012a) models in its development process, and the latter is a model-driven Web engineering method. During the Definition phase, four models are defined: a use-case model (UP), a domain model (UWE), a user model (UWE) to identify the types of users and an application model (UWE) for the navigational description, user interface description and functional mapping.
- **MDA abstraction levels.** The domain model is the unique model that can be truly considered to be at CIM level. The rest of the models are defined at PIM level because they are platform-independent. WEM does not propose any model defined at PSM level.
- **Modelling language.** UML is the modelling language used by WEM. It is also the standard modelling language used by both UP and UWE.
- **Automation level.** WEM has a low level of automation since it does not consider any kind of transformation to automate the development process.

Even though, WEM steps on UWE (a model-driven Web engineering method) and considers models as a key aspect in its development process (as we proved with the criteria *models considered*), it is not considered to follow the MDE principles since it does not automate the development process, one of the key aspects of MDE.

- **Supporting toolkit.** This method does not have any supporting toolkit associated.

### 3.2.3.2 Souer et al. (WebForm Diagram)

The authors of the studies (Souer et al., 2009), (Souer & Kupers, 2009) think that customizing CMS-based Web applications is a difficult task and that there is a gap between the requirements analysis in this type of Web applications and its implementation. Accordingly, they propose a model-driven method to configure automatically a CMS-based Web application based on requirements. They expect this method to be easy and pragmatic to be used even by non-technical users, such as business users.

The authors of this approach are the same of WEM method, but we decided to take this approach in a separated manner because the idea proposed in the later studies differs significantly from the WEM method. It starts with the definition of a business model (henceforth WebForm Diagram) that is implemented in XML. Then, this model is transformed automatically to another XML model that contains the specific concepts from the CMS domain (CMS XML model). Finally, a specific XML model that configures the CMS-based Web application is generated automatically from the CMS XML model.

One of the key aspects of this method is the modelling language that they propose. It is an unambiguous visual language for the specification of online business processes. It is developed to cover all online form variables within a Web application. To define the elements of this language they also followed the SME approach. They took into consideration other Web application modelling languages such as WebML (Ceri et al., 2000), OOWS (Valverde, Valderas, Fons, & Pastor, 2007) and OOHDML (Schwabe & Rossi, 1995).

In the following, we analyze the proposal by Souer et al. according to the criteria defined in Section 3.2.2.

- **Type of process.** This method proposes a development process comprised by three steps: 1) definition of a business model, 2) transformation to a CMS

XML model and 3) generation of a XML model that configures the CMS-based Web application.

- **Web Engineering views.** This method covers three Web Engineering views (navigation, process and implementation views) as we can see in Figure 3-8. The navigation view is addressed because the WebForm Diagram considers concepts such as, forms, steps, form elements and pages, whereas the process view is addressed by concepts, such as action and handler. The implementation view considers the automatic generation of the XML model that configures the CMS-based Web application.

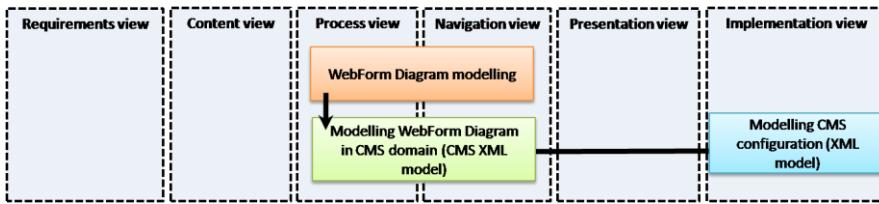


Figure 3-8. WebForm diagram definition and Web Engineering views.

- **Models considered.** The models considered in this method are a business model, called WebForm Diagram, and two XML models (CMS XML model and XML model configuring the CMS-based Web application).
- **MDA abstraction levels.** The WebForm Diagram and the CMS XML model are defined at PIM level since they represent platform-independent concepts. On the other hand, the XML model configuring the CMS-based Web applications is considered to be at PSM level.
- **Modelling language.** To specify the WebForm Diagram a concrete modelling language is defined. This modelling language is specified as a DSL and intends to be understandable and intuitive enough to be used by non-technical users such as business users. To define the elements of this DSL they follow the SME approach. They take features from other modelling languages such as WebML (Ceri et al., 2000), OOWS (Valverde, Valderas, Fons, et al., 2007) and OOHDM (Schwabe & Rossi, 1995).
- **Automation level.** We consider that this method has a high level of automation because it defines M2M and M2T transformations that generate automatically the configuration of the CMS-based Web application from the WebForm Diagram. The M2M transformation is implemented using Java. Whereas, the M2T transformation is implemented in *EXtensible Stylesheet Language Transformation* (XSLT) (W3C, 2012).

- **Supporting toolkit.** This method is supported by a modelling tool which allows the implementation of the WebForm Diagram. However, in some works they have defined this model by using the tool MetaEdit+ (Steven Kelly, Lyytinen, & Rossi, 1996).

### 3.2.3.3 Saraiva et al.

In the studies (Saraiva & Silva, 2008), (Saraiva & Silva, 2009a), (Saraiva & Silva, 2009b), (Saraiva & Silva, 2010) the authors present a model-driven method for the development of CMS-based Web applications. This method is based on two modelling languages situated at different abstraction levels; the CMS Modelling Language (CMS-ML) at a higher abstraction level and the CMS Intermediate Language (CMS-IL) at a lower one. These modelling languages are inspired on existing modelling languages such as WebML (Ceri et al., 2000), UWE (Koch, Knapp, Zhang, & Baumeister, 2008) and XIS2 (Rodrigues, Saraiva, Silva, & Martins, 2007).

This method is composed of two phases: the **modelling phase**, where the CMS-based Web application is modelled at two different abstraction levels, and the **deployment phase**, where the deployment artefacts that implement the CMS-based Web application are generated from the model defined at lower abstraction level in the previous phase.

The modelling phase is composed of three tasks: 1) the definition of the *Web-site templates* model by using the CMS-ML language, 2) the definition of *toolkits* model with CMS-ML language and 3) the automated transformation of these models into the CMS-IL language. The deployment phase consists of one task based on the automatic implementation of the CMS-based Web application from these models.

Below, we analyze the proposal by Saraiva et al. according to the criteria defined in Section 3.2.2.

- **Type of process.** This method proposes a development process for CMS-based Web applications. Such process is composed of four tasks: 1) the definition of the *Web-site templates* model by using the CMS-ML language, 2) the definition of *toolkits* model with CMS-ML language, 3) the automated transformation of these models into the CMS-IL language; and finally, 4) the automated implementation of the CMS-based Web application from these models.

- **Web Engineering views.** The method covers all the Web Engineering views we can see in Figure 3-9. The requirements, content, navigation and presentation views are addressed by the task of definition of the *Web-site templates* model. Moreover, the task of definition of the *toolkits* model addresses the process and navigation views. Finally, the implementation view is addressed by the automatic generation of the deployment artefacts for implementing the CMS-based Web application.

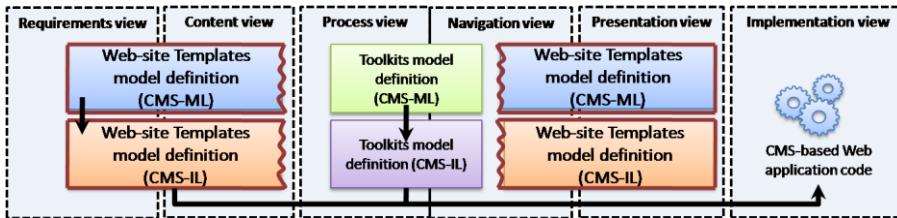


Figure 3-9. CMS-ML and CMS-IL modelling languages.

- **Models considered.** Two models are considered in this method. The *Web-site templates* model which reflects the structure, the content, the navigation and look and feel of the Web application and the *toolkits* model which allows the addition of task-oriented extensions and complex user interface elements to the elements of the *Web-site templates* model. These two models are defined at different abstraction level by using the two modelling languages proposed by this method: CMS-ML and CMS-IL.
- **MDA abstraction levels.** According to the semantics given to the elements modelled, we consider that, independently of the language used (CMS-ML or CMS-IL), the models defined in this method are at PIM level. The CMS-ML language is a high-level language but not so high as to be considered at CIM level. In turn, the CMS-IL language is considered as a low-level language but platform-independent. In any case, this method does not consider any model at CIM or PSM level.
- **Modelling language.** The two modelling languages defined in this method have been designed specifically for the CMS domain. On the one hand, the CMS-ML language (Saraiva & Silva, 2010) is a graphical modelling language defined as a DSL. It can be seen as a set of mnemonics hiding the details of the CMS-IL language. This language is inspired on WebML, UWE and XIS2. On the other hand, the CMS-IL language (Carmo, 2006) is a textual language that provides a mechanism which can be used by technical

developers to address low-level computation aspects, and deploy the CMS-based Web application.

- **Automation level.** This method has a high level of automation because it defines M2M transformations that generate automatically the CMS-IL models from the CMS-ML models and M2T transformations that implement automatically the CMS-based Web application from the CMS-IL models.
- **Supporting toolkit.** This method does not have any supporting toolkit associated.

### 3.2.3.4 Vlaanderen et al.

The studies (Vlaanderen et al., 2008), (Vlaanderen et al., 2009) present the adaptation of the existing model-driven Web Engineering method OOWS (Valverde, Valderas, Fons, et al., 2007), (Valverde, Valderas, & Fons, 2007) to the development of CMS-based Web applications.

Authors state that it is difficult to find a method that is the most suitable in all the different domains. Therefore, when a new domain needs to be supported, the method must be adapted, extended or redefined. All things considered, they decide to adapt the OOWS method to the CMS domain. They evaluate the suitability of the OOWS method in the context of the CMS-based Web applications and detect a list of current limitations. One of the most important limitations is that the models considered within OOWS method lack expressiveness to define CMS-based Web applications. Thus, OOWS is extended with new tasks (including new models) in order to address the modelling of this type of Web applications. It is worth noting that this extension has been carried out considering the principles of the SME approach.

The three tasks included in this adaptation of OOWS are: **session information specification**, redefines the object model. In this task, the analyst points out which information will be persisted in the session cache; **dynamic user information**, is used to extend the information represented in the user model. It defines the user information which cannot be decided at modelling time, such as the first user password or the language, and **detail navigational map**, models complex processes performed by the CMS-based Web application, such as the user-registration process in this kind of Web applications. The main purpose is to extend the Navigational model marking how data input must be grouped.

In the following, we analyze the proposal by Vlaanderen et al. according to the criteria defined in Section 3.2.2.

- **Type of process.** The proposed extension of the OOWS method defines a development process for CMS-based Web applications. It starts from a requirements modelling task and finishes generating code which implements the CMS-based Web application.
- **Web Engineering views.** The OOWS method addresses all the Web Engineering views as we can see in Figure 3-10, but the tasks included for the adaptation of OOWS covers two of them (content and navigation). As we can see in Figure 3-10, the session information specification covers the content view and the dynamic user information and detail navigational map tasks addresses the navigation view.

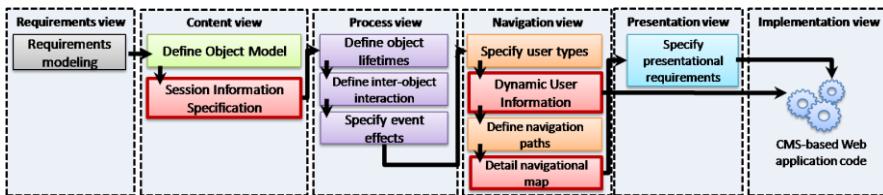


Figure 3-10. Adaptation of OOWS to CMS domain.

- **Models considered.** The three models that have been adapted for the development of CMS-based Web applications are object model, user model and navigational model.
- **MDA abstraction levels.** The three models adapted (object model, user model and navigational model) are specified at PIM level.
- **Modelling language.** OOWS method does not define any specific modelling language for modelling CMS-based Web applications. They propose the use of the OOWS modelling language and other standard modelling languages such as UML. The object model and the user model are defined with the OOWS language, whereas the navigational model is defined with UML.
- **Automation level.** OOWS has a high level of automation since it defines M2M transformations and the automatic generation of the code by M2T transformations. M2M transformations are implemented in XSLT, whereas the M2T transformations in *Xpand* language (OpenArchitectureWare, 2009).
- **Supporting toolkit.** OOWS is supported by the tool OlivaNova (Pedro J. Molina, 2004) that is the industrial implementation of the OOWS method.

### 3.2.4 Discussion

In the previous section, we analyze all the approaches related to the context of the CMS-based Web applications found in the literature. We use the information extracted in the previous section to compare them and extract the conclusions to answer the research questions (RQ) defined in Section 3.2.1 sums up the information extracted by considering criteria defined in Section 3.2.2.

**Table 3-1. Data extraction from the approaches related to CMS-based Web applications.**

Approach	Process	Web Engineering views	Models	MDA levels	Modelling language	Automation level	Toolkit
Souer et al. (WEM)	Development process	Requirements, Process Navigation, Presentation, Implementation	Use-case, Domain, User and Application models	CIM PIM	UML UWE	Not defined (n.d.)	n.d.
Souer et al. (WebForm diagram)	Development process	Process Navigation	WebForm Diagram	PIM	WebForm modelling language	M2M M2T	Graphical editor
Saraiva et al.	Development process	All	CMS-ML model and CMS-IL model	PIM	CMS-ML CMS-IL	M2M M2T	n.d.
Vlaanderen et al.	Development process	Content Navigation	Object model, User model, Navigation model	PIM	UML OOWS	M2M M2T	Oliva Nova
Trias et al. ( <i>ADMigrac MS</i> )	Reengineering process	All	ASTM_PHP model, Code model, CMS model	PIM PSM	ASTM_PHP DSL, KDM_COD E DSL, CMS DSL	T2M M2M M2T	<i>ADMigrac MS</i> toolkit

- **RQ1. Which approaches do exist in the literature for addressing a development process of CMS-based Web applications?**

With the performance of this SLR, we found four methods proposing a development process for CMS-based Web applications. In the following, we discuss the results regarding the seven criteria used for the data extraction.

According to the **type of process**, the four methods follow a top-down development process for the development of CMS-based Web applications. Methods to address a reengineering process for this type of Web applications have not been found. All the methods found have not been defined from scratch, but all

of them are based on other existing Engineering methods. Concretely, three of the methods have been defined following the SME approach. Considering the data gathered in Table 3-1, we can compare our *ADMigraCMS* method (inserted in the last row in the table) with these other method. We can state that *ADMigraCMS* method is the unique method addressing a reengineering process for CMS-based Web applications.

As for the **Web Engineering views** we can say that all the methods address at least one of the views proposed by the Web Engineering (Deshpande & Hansen, 2001). According to Table 3-1, the two most addressed views are the navigation and the process views, whereas the least addressed one is the content view. There are two approaches taking into account all the views: Saraiva et al. and Vlaanderen et al. On the contrary, Souer et al. (WebForm Diagram) is the approach considering less views, only the navigation and process views. As for the *ADMigraCMS* method, we can state that it addresses all the Web Engineering views reflected within the CMS Common Metamodel proposed in this PhD Thesis.

Regarding the **models considered**, we can say that all the methods use models as key artefacts within their development process, but not all of them follow the MDE principles (use of models + automated process). WEM, proposed by Souer et al., is not considered as a model-driven method, even being based on UWE and defining modelling tasks, because it does not automate the development process. On the contrary, Souer et al. (WebForm Diagram), Saraiva et al. and Vlaanderen et al. are considered as model-driven methods. *ADMigraCMS* is considered a model-driven reengineering method.

As for the **MDA abstraction level** of the models and according to Table 3-1, we can say that most of the methods consider models at PIM level. Only Souer et al. (WEM) bet for the use of models at CIM level and Vlaanderen et al. consider models at PSM level. It is interesting to say that most of them try to generate automatically code from PIM models instead of PSM models. AS for our *ADMigraCMS* method, the models considered for the reengineering process are defined mainly at PSM and PIM levels. The restructuring stage is addressed at PIM level and the code generation at PSM level. Models at CIM level are not considered.

Considering the **modelling languages**, we can say that most of the methods use generic languages which are not specific to the CMS domain. Thereby, two of the methods consider standard languages such as UML. As for new modelling languages proposed by the methods, we can consider three: WebForm proposed

by Souer et al. (WebForm Diagram), and CMS-ML and CMS-IL proposed by Saraiva et al. Except from CMS-IL which is considered a textual modelling language, the other two are graphical modelling languages. These three languages are focused on modelling the navigation and process views. WebForm's concepts are more generic, whereas CMS-ML and CMS-IL are more centred in the CMS domain. *ADMigraCMS* method provides with three graphical modelling languages, the ASTM\_PHP modelling language, the KDM\_CODE modelling language, and the CMS modelling language which allows to model the key aspects of CMS-based Web applications.

According to their **automation level**, all the methods considered as model-driven define M2M and M2T transformations to automate their development processes. The automation level of the *ADMigraCMS* method is high since the reengineering process is automated by T2M, M2M and M2T transformations.

As for the **supporting toolkits**, we can say that the two only methods supported by a toolkit are Souer et al. (WebForm Diagram) with a graphical editor and Vlaanderen et al. with OlivaNova, a comprehensive and industrial toolkit. *ADMigraCMS* method is supported by a toolkit composed of a set of graphical editors and automated transformations.

- **RQ2. Which approaches do exist in the literature for addressing a reengineering process for CMS-based Web applications?**

As we mentioned previously, none of the methods found is able to address a reengineering process. It is worth noting that Souer et al. (WEM) proposes a migration route, but we have not found any study deploying this route, thus we have discarded it.

- **RQ3. Has any existing method been adapted to the development of CMS-based Web applications?**

Most of the authors of these methods agree with the idea that undergoing the development process of CMS-based Web applications is complex and time consuming. Existing Engineering methods do not cover the issues that arise when developing this type of Web applications so that they need to be adapted to this development. In this SLR, we have found one existing Engineering method, OOWS, which tries to adapt its development process to the development of CMS-based Web applications.

- **RQ4. Which approaches can be framed within the context of Model-Driven Engineering (MDE)?**

All of the methods found are considered model-driven, except WEM. It is because WEM does not propose the automation of its development process.

- **RQ5. Which is the interest of the Web Engineering community in researching about CMS-based Web applications?**

We consider that the Web Engineering community has not dedicated much effort in the research about the CMS-based Web applications. It has surprised us considering the many advantages that they offer to organizations and considering that the existing Engineering methods do not address thoroughly the development of this kind of Web applications. Only the authors behind OOWS have shown interest in the adaptation of its process in the CMS domain.

We can find some active research groups such as the group in the Utrecht University and the group in the Technical University of Valencia. Regarding the forums where the primary studies were published, we conclude that they are recent forums (from 2006 to 2011) but they are not high-level such as CORE-A or JCR levels, but they are mainly CORE-C.

### 3.3 Model-Driven Reengineering Approaches

To complete the body of knowledge of the state of the art of this PhD Thesis, in this section we carry out another literature review to identify and analyze the existing model-driven reengineering approaches in order to have a more complete picture of the area of Model-driven Reengineering.

Like in Section 2.2, for this literature review we followed a SLR. Part of the results of this SLR have been published in (Trias, de Castro, López-Sanz, & Marcos, 2013b).

In the following Section 3.3.1, we present the objectives and research questions which guide this literature review. In Section 3.3.2, the criteria to analyze the approaches found in the literature are presented. In Section 3.3.3 we present the analysis of each approach, and finally, in Section 3.3.4 we present the discussion from the data extracted during this SLR.

### ***3.3.1 Research Questions***

To address this SLR, we propose the following research questions (RQ) which we will answer in Section 3.3.3, during the data extraction task.

- **RQ1. Which model-driven approaches do exist in the literature for addressing a reengineering process?**

We are interested in finding the existing reengineering methodological approaches based on the model-driven principles to know their features and operative steps.

- **RQ2. In which domain does the Software Reengineering Community dedicate more efforts to define model-driven approaches to automate a reengineering process?**

We are interested in knowing the domain where the approach is executed. This domain could be the databases, the legacy systems in general or Web applications, among others. We know that the CMS-based Web applications domain is not addressed by the reengineering approaches currently, but we want to know which is the domain most tackled by these approaches.

- **RQ3. Which kind of process is defined by the approaches? Which reengineering stages (reverse engineering, restructuring and forward engineering) do they consider in their process?**

We are interested in knowing which engineering stages are addressed by the approach. It is possible to find approaches addressing the three stages, defining a horseshoe process or just addressing just one of them. This issue allows us to know how to address comprehensively a reengineering process.

- **RQ4. Which is the level of automation of the approaches found in the literature? Are they supported by any toolkit?**

We want to know the level of automation of the approach to check the level of maturity of the approach as to avoid errors and to reduce time in the migration process. We analyze which type of transformations (T2M, M2M or M2T) is considered by the approach and whether it is supported by any toolkit allowing to automate the reengineering process.

- **RQ5. Which type of techniques are used by the approaches to extract and model the software artefacts from the legacy system? In which code**

**are the software artefacts implemented? Which is the technique to generate code?**

We are interested in knowing which techniques are used by the model-driven reengineering approaches to extract the models from the legacy code. Also, we want to know whether any of these approaches use any model extractor from PHP code. If so, we could assess the possibility of use it.

- **RQ6. Which type of techniques are used by the approaches to implement the M2M transformations? Which are the MDA level of the models transformed by these M2M transformations?**

We want to know from the approaches that implement the M2M transformations as part of their process, what transformation language they use to implement them. It can help us to choose the transformation language to implement the M2M transformation of our approach.

- **RQ7: Which is the use of the standard metamodels proposed by ADM among the reengineering approaches?**

We want to know the level of implementation of the ADM initiative. We are interested in knowing if the approaches use the standard metamodels proposed by ADM. Thereby, we can analyze the level of use and acceptance of this initiative.

- **RQ8. Which is the interest of the Web Engineering community in researching about CMS-based Web applications?**

We want to know which research groups or organizations are leading the research in this area and which forums publish studies focused on this issue. It allows to analyze the impact and interest of this topic in the research community.

### 3.3.2 Data Extraction Definition

To analyze any approach that may answer the previous questions, we have define a set of criteria. These criteria allow the extraction of data from any approach, which is used to organize discuss and reach the final conclusions presented in Section 3.3.3. In the following, we present these criteria used to analyze the different approaches.

- The **reengineering stage** analyzes which of the established reengineering stages are covered by the proposal: 1) *reverse engineering stage*, 2) *restructuring stage* or 3) *forward engineering stage*.

- The **scope of the approach** analyzes the aspect in which the approach is focused, so that it is possible to find approaches focused on: 1) the reengineering of data bases, 2) reengineering of Web applications, 3) the reengineering of Web services or 4) legacy systems (general purpose).
- The **ADM abstraction levels** checks the abstraction level at which the models are defined by the approach: 1) CIM level, 2) PIM level or 3) PSM level.
- The **metamodels considered** allows to know which metamodels the models defined by the approach conforms to. Accordingly, the approaches can: 1) define their own metamodel to represent the models at any abstraction level, 2) use existing standard metamodels and 4) use the ADM standard metamodels (ASTM, KDM or SMM).
- The **automation level** allows to analyze whether the transformations considered by the approach to systemize the reengineering process are implemented. If the approach implements all its automated transformations, we consider it with a high level of automation. If the approach implements some of its transformations we consider it with a medium level of automation. Finally, if the approach does not implement any type of transformation we consider it with a low level of automation.
- The **implementation of transformations** allows to know the technologies and techniques used to implement the automated transformations considered in the approach. In the case of the T2M transformations the techniques might be: 1) parser, 2) existing tool in the market or 3) implementation of transformation rules. As for the M2M and M2T transformations, the technologies are related to the transformation languages used to implement the transformation rules
- The **kind of code to migrate** is related to the source code implementing the legacy system from which the approach extracts the models and starts the reengineering process.
- The **supporting toolkit** is about the availability of tools supporting the tasks defined in each reengineering approach, such as graphical editors for creating models or frameworks allowing to run the automated transformations.

### 3.3.3 Data Extraction

In this section, we present the analysis and the data extraction from the approaches considering the criteria defined in the previous section. We analyze the nine approaches defined from the 27 primary studies obtained with this SLR. To know how we obtained the 27 primary studies it is possible to see the complete SLR process presented in Appendix B. Table 3-2 sums up the approaches presented in this section and the data extracted for each of them. In Section 3.3.4, we present the conclusions reached from this data extraction task.

#### 3.3.3.1 García-Rodríguez de Guzman et al. (Rational Web)

The work presented in (García-Rodríguez de Guzman, Polo, & Piattini, 2005), (Polo, García-Rodríguez de Guzmán, & Piattini, 2007), (Pérez-castillo, Sánchez-gonzález, Piattini, García, & Garcia-Rodriguez de Guzman, 2011) presents a method designed specially for the reengineering of relational databases. This method allows the automatic generation of different multilayer applications from relational databases.

Through this tool, you can create four types of applications from four types of databases (Oracle, Intersystems Cache, SQLServer and Microsoft Access). The method is defined as a horseshoe process and consists of the following steps: 1) the **physical schema** of the database is extracted from a dictionary of a database, 2) this *physical schema* is transformed into a **database model** and 3) translation of each table to a **class diagram** conforming with the metamodel OOS (Object Oriented System) (Schwabe & Rossi, 1998). The user may manually manipulate the classes of this last model, adding new operations. 4) from the *class diagram*, a **specific platform model** is generated from which the code is generated. It can generate a JSP application, a Java desktop application or a Web service-based application. In the following, we analyze the approach applying the criteria defined in 3.3.2.

- **Reengineering stages.** This approach addresses the three reengineering stages: 1) in the *reverse engineering stage*, the *physical schema* is extracted from the dictionary of the database, afterwards this *physical schema* is moved to a *database model*, finally it is translated into a *class diagram*, 2) in the *restructuring stage*, the user can manipulate manually the classes of this *class diagram*, adding new operations by means of state machines and 3) in the *forward engineering stage*, from the *class diagram* it is created a *specific platform model* from which the code is generated automatically for different

target applications. Figure 3-11 shows the matching between the reengineering stages and the steps conforming the Relational Web approach.

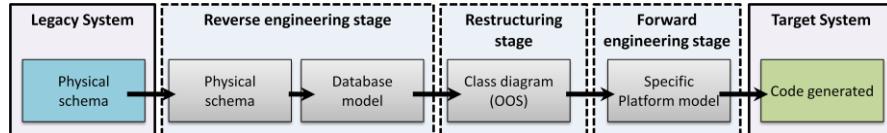


Figure 3-11. Relational Web method.

- **Scope of the approach.** This approach is focused on the reengineering of applications. The method extracts data from the relational database to generate multilayer applications following a complete process of reengineering.
- **MDA abstraction levels.** This approach defines models at different level of abstraction. The *physical schema* is defined at PSM level, whereas the *database model* and the *class diagram* are defined at PIM level. The restructuring stage is carried out at PIM level. Models at CIM level are not defined.
- Metamodels considered. **All the metamodels defined by this approach are based on existing metamodels so that they do not define their own metamodels. The database model conforms to the database metamodel, whereas the class diagram conforms to the OOS metamodel.**
- **Automation level.** To extract the *physical schema* from the database they define T2M transformation. To obtain the *database model* from the *physical schema* as well as to obtain the *class diagram* from the *database model* or the *specific platform model* from the *class diagram* a set of M2M transformations are defined. Finally, M2T transformations are defined to generate the code. Therefore, the automation level of this approach is high since all the reengineering process is automated.
- **Implementation of transformations.** The T2M transformations are implemented by a parser. As for the M2M transformations, we have not found any study presenting their implementation. Finally, the M2T transformations are implemented by a code generator.
- **Kind of code to migrate.** This approach is applied on legacy relational databases implemented in Oracle, Intersystems Cache, SQLServer or Microsoft Access. So the model extraction is performed over these three technologies.

- **Supporting Toolkit.** The automated transformations of this method are supported by a toolkit, whereas the management of the models is not supported by any toolkit such as a graphical editor.

### 3.3.3.2 Blanco et al.

The work presented in (Blanco, Pérez-castillo, Hernández, & Fernández-Medina, 2009) presents an ADM-based method focused on obtaining conceptual security models from legacy OLAP systems (Berson & Smith, 1997). It allows to analyze OLAP systems in order to include new security requirements to improve the architecture as well as to migrate to other platforms. The process is composed of the following steps: 1) from an OLAP system they extract automatically a **DataWarehouse model** which conforms to the Secure Multidimensional Logical Metamodel (SECMDDW) which considers the common structure of OLAP systems, 2) from this logical model they generate automatically a **conceptual model** which conforms to SECDW metamodel that allows the representation of structural aspects of DataWarehouses and the definition of several kinds of security rules. It is complemented by **an Access Control and Audit (ACA) model** focused on DataWarehouse confidentiality. The code is generated by executing M2T transformations. In the following, we analyze the approach applying the criteria defined in 3.3.2.

- **Reengineering stages.** This approach addresses the three reengineering stages 1) in *the reverse engineering stage*, they automatically extract *DataWarehouse model* from an OLAP system, afterwards from this *DataWarehouse model* they generate automatically a *conceptual model* that allows the representation of the structural aspect of DataWarehouses and the definition of several kinds of security rules, 2) in the *restructuring stage*, new security requirements are included within the *conceptual model* to improve the architecture of the OLAP systems, finally 3) in the *forward engineering stage*, a new model is considered, the *ACA model* focused on DataWarehouse confidentiality. From the set of models defined previously the code is generated. Figure 3-12 shows the matching between the reengineering stages and the steps conforming this reengineering approach.

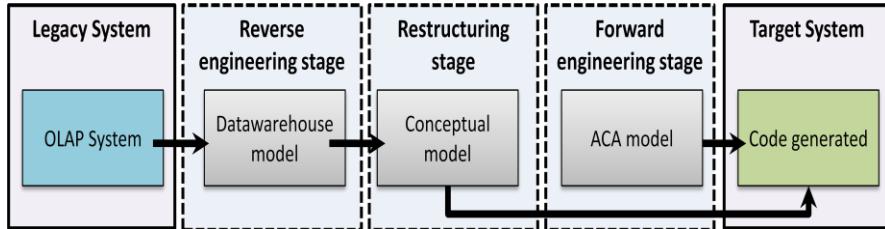


Figure 3-12. Modernization process for Secure DW.

- **Scope of the approach.** This approach presents an ADM process focused on obtaining conceptual security models from legacy OLAP systems. These models allow us to analyze OLAP systems in order to include new security requirements. These requirements improve the architecture of the OLAP systems and allow to migrate it to other platforms.
- **MDA abstraction levels.** This approach models different aspects of an OLAP system at different levels of abstraction. The *DataWarehouse model* is defined at PSM level, whereas the *conceptual model* and *ACA models* are defined at PIM level. The restructuring stage is carried out at PIM level. Models at CIM level are not defined.
- **Metamodels considered.** To define the models they define they own metamodels. The SECMDDW metamodel considers the common structure of OLAP systems and allows to represent a *DataWarehouse model* closer to OLAP platforms. This metamodel is based on a security improvement of the OLAP package from CWM (Common Warehouse Metamodel). On the other hand, SECDW metamodel allows the representation of structural aspects of DataWarehouses and the definition of several kinds of security rules. It is a UML profile created specifically for DataWarehouses.
- **Automation level.** To extract the *DataWarehouse model* they define T2M transformation. On the other hand, to obtain the conceptual model and ACA model, they defined M2M transformations. As for the generation of code, they define their own M2T transformations. Thus, the automation level of this approach is considered to be high.
- **Implementation of transformations.** The T2M transformations have been implemented by a parser. As for the M2M transformations have been defined by transformation rules implemented with QVT. Finally, the M2T transformations have been defined with a code generator.

- **Kind of code to migrate.** This approach is focused on the reengineering of legacy OLAP systems.
- **Supporting Toolkit.** They do not take into account any tool that addresses the automation of the approach. They do not consider graphical editors to define the models, either.

### 3.3.3.3 Perez-Castillo et al. (MARBLE)

The works (Perez-Castillo, Garcia-Rodriguez de Guzman, Avila-Garcia, & Piattini, 2009), (Pérez-castillo, Garcia-Rodriguez de Guzman, & Piattini, 2010) (Pérez-Castillo, García-Rodríguez de Guzmán, & Piattini, 2011), (Pérez-Castillo, García-Rodríguez de Guzmán, Piattini, Weber, & Places, 2011), (Pérez-Castillo, Weber, Garcia-Rodriguez de Guzman, & Piattini, 2011), (Pérez-Castillo, García-Rodríguez de Guzmán, Piattini, & Weber, 2012) present MARBLE (*Modernization Approach for Recovering Business Processes from LEGacy systems*), an ADM-based framework for recovering business processes from legacy systems.

MARBLE is organized in four abstraction levels related to four different kinds of models which are necessary to obtain the embedded business processes from a legacy system: *L0, legacy information system*, this level represents the legacy system in the real world; *L1, software artefacts models*, this level contains a set of models representing one or more software artefacts; *L2, KDM model*, this level represents the legacy system from a platform-independent point of view; *L3, Business process model*, this level represents the recovered business processes.

Moreover, MARBLE defines three transformations tasks which allow to transit from one abstraction level to another one: 1) *model extraction*, where a **code model** is obtained from the Java code which implements the legacy system. 2) *generation of a KDM model*, this **KDM model** is defined considering the code and action packages of the KDM metamodel, 3) *obtaining business process model*, the **business process model** which conforms to BPMN represents the business discovered and is manually restructured by business experts. In the following, we analyze the approach applying the criteria defined in 3.3.2.

- **Reengineering stages.** This approach addresses two of the reengineering stages: 1) in the *reverse engineering stage*, they extract automatically the *code models* from Java code, afterwards they raise the abstraction level generating a *KDM model* from that *code model* and finally they obtain the *business process model* from the *KDM model*, 2) in the *restructuring stage*, the experts can restructure manually the *business process model*. Figure 3-13

shows the matching between the three reengineering stages and the steps defined by MARBLE.

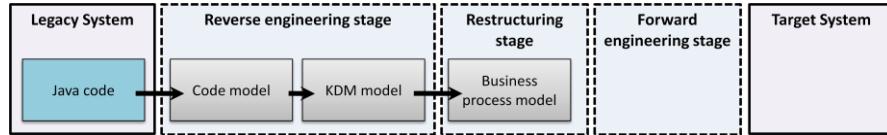


Figure 3-13. MARBLE method.

- **Scope of the approach.** This approach is focused on extracting business processes from legacy systems.
- **MDA abstraction levels.** In MARBLE we can find models defined at different abstraction levels. The *code model* is defined at PSM level, whereas the *KDM model* is defined at PIM level. Finally, the *business process model* is defined at CIM level. The restructuring stage is carried out at the CIM level.
- **Metamodels considered.** They define the Java metamodel to define the *code model*. This Java metamodel captures the syntax elements of the Java language. The KDM metamodel, one of the standard metamodels proposed by ADM, is used in this approach to define the *KDM model*. On the one hand, the standard metamodel BPMN is considered to base the *business process model*.
- **Automation level.** To obtain the *code model* they implement T2M transformations. To obtain the KDM model from the Java model and the business process model from the KDM model, they define M2M transformations. MARBLE automates all the reengineering process, thus its automation level is high.
- **Implementation of transformations.** The T2M transformations have been implemented by a syntactical parser. As for the M2M transformations, they have been implemented by defining transformations rules implemented with QVT.
- **Kind of code to migrate.** This method extracts business processes from legacy systems implemented in Java. For this reason, the Java model is extracted in the first step of this reengineering method.

- **Supporting Toolkit.** A complete tool supports the automation of the reengineering process of MARBLE although they do not mention the use of graphical editors to manage the models.

### 3.3.3.4 García-Rodriguez de Guzman et al. (PRECISO)

The works (Garcia-Rodriguez de Guzman, Polo, & Piattini, 2006a), (Garcia-Rodriguez de Guzman, Polo, & Piattini, 2006b), (Garcia-Rodriguez de Guzman, Polo, & Piattini, 2007), (García-Rodríguez de Guzmán, 2007), (Perez-Castillo & García-Rodríguez de Guzmán, 2009), (Perez-Castillo, Garcia-Rodriguez de Guzman, Caballero, Polo, & Piattini, 2009) present PRECISO a reverse engineering method to extract Web Services from relational databases in automatic manner. This process is based on ADM and it is composed of the four following stages: 1) extraction of a **SQL-92 model** according to the SQL-92 metamodel from a relational database, 2) transformation of this SQL-92 model into the **object model**, conforming to UML2 metamodel, which raises the abstraction level of the system, 3) generation of a **WSDL model** conforming to the WSDL metamodel (Christensen, Curbera, Meredith, & Weerawarana, 2001), (Chinnici, Moreau, Ryman, & Sanjiva, 2006) from the *object model* and 4) generation of the code from the *object model* and the *WSDL model*. This code is the basis for implementing the infrastructure of Web Services. In the following, we analyze the approach applying the criteria defined in 3.3.2.

- **Reengineering stages.** This approach addresses the three reengineering stages<sup>1</sup>) in the *reverse engineering stage*, they extract the *SQL-92 model* from a relational database, afterwards this model is transformed into a *object model* which is the basis to generate Web Services activities, 2) in the *restructuring stage*, the user can adapt this *object model* to refine the features of the Web service, finally 3) in the *forward engineering stage*, the *WSDL model* is generated from the *object model*. This model depicts Web Services; finally, from the *object model* and the *WSDL model* they generate C# code that implements the infrastructure of Web Services. Figure 3-14 shows the matching between the reengineering and the PRECISO method.

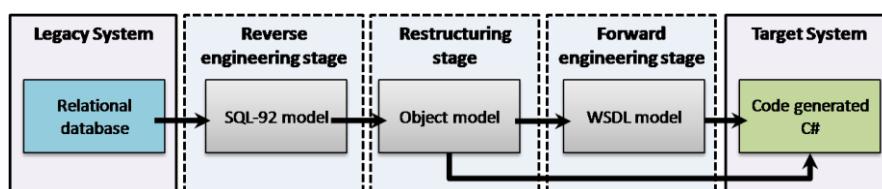


Figure 3-14. PRECISO method.

- **Scope of the approach.** This approach is focused on extracting Web Services from relational databases in automatic manner.
- **MDA abstraction levels.** The *SQL-92 model* and the *WSDL model* are defined at PSM level, whereas the object model is defined at PIM level. Models at CIM level are not considered in this approach. Finally, the restructuring phase is carried out at the PIM level.
- **Metamodels considered.** The *SQL-92 model* conforms to the SQL-92 metamodel which is specifically defined for this approach. The *object model* conforms to the existing UML2 standard metamodel, an existing standard metamodel. Finally, the *WSDL model* conforms to the WSDL metamodel which depicts the Web Services.
- **Automation Level.** To obtain the *SQL-92 model* they defined T2M transformations. To obtain the *object model* from the *SQL-92 model* as well as the *WSDL model* from the *object model* a set of M2M transformations are defined. To generate the C# code they implement M2T transformations. This method is completely automated, thus its automation level is high.
- **Implementation of transformations.** The T2M transformations are implemented by a parser. As for the implementation of the M2M transformations, we have not found any work explaining in detail it. Finally, the M2T transformations are implemented by a code generator.
- **Kind of code to migrate.** This method is centred in extracting automatically Web Services from relational databases implemented in SQL-92.
- **Supporting Toolkit.** This approach is supported by a comprehensive tool. This tool supports and automates all the process as well as proposes the use of editors to define the models.

### 3.3.3.5 Perez-Castillo et al. (Schema Elicitation Method)

The work presented in (Pérez-Castillo, García-Rodríguez de Guzmán, Caivano, & Piattini, 2012) proposes a method for modernizing the legacy source code of a relational database. As a sample modernization, this method removes “dead” parts of the database schema such as duplicated or unused tables and unused columns. This method considers two complementary sources of knowledge: 1) the schema of the legacy database, and 2) the SQL sentences embedded in the legacy source code. The method is composed of two steps: 1) It extracts by means of a syntactical analyzer, the specific-platform **SQL Statement model** which conforms to Data Manipulation Language (DML) metamodel

(Kunii, 1990), the **KDM Inventory model** and the platform-independent **KDM Code model**, 2) the platform-independent **KDM Data model** is generated from the *SQL Statement model*. Therefore, the *restructuring* and *forward engineering stages* are carried out in order to generate the modernized version of the legacy systems. In the following, we analyze the approach applying the criteria defined in 3.3.2.

- **Reengineering stages.** This approach addresses the three reengineering stages: 1) in the *reverse engineering stage*, a *SQL Statement model*, a *KDM Inventory model* and a *KDM Code model* are extracted from the *SQL* sentences embedded in the legacy source code. Afterwards, from the *SQL Statement model* the *KDM Data model* is generated, 2) in the *restructuring stage*, the engineer can restructure the *KDM data model* to modernize the code and finally 3) the *forward engineering stage*, is considered but not specified in the approach. Figure 3-15 shows the correspondence between the reengineering stages and the steps of this reengineering method.

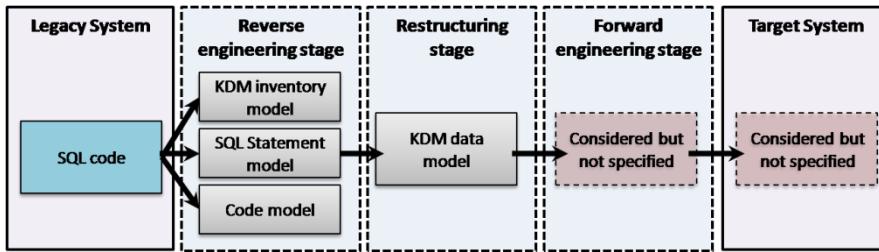


Figure 3-15. Schema Elicitation method.

- **Scope of the approach.** This approach is focused on the modernization of legacy source code together with legacy relational databases.
- **MDA abstraction levels.** This approach defines models at different abstraction levels. The *SQL Statement model* is defined at PSM level, whereas the *KDM Inventory model*, the *KDM Code model* and the *KDM Data model* are defined at PIM level.
- **Metamodels considered.** The *SQL Statement model* conforms to DML metamodel, an existing standard metamodel. On the other hand, the *KDM Inventory model*, the *KDM Code model* and the *KDM Data model* are conform to different packages of the KDM metamodel.
- **Automation Level.** To obtain the *SQL Statement model*, the *KDM Inventory model* and the *KDM Code model*, they define T2M transformations. On the

other hand, to generate the *KDM Data model*, they specify M2M transformations. They refers to the necessity of defining a forward engineering stage and a code generation task, but they do not define any transformations. Thus, the reengineering process provided by this approach is partially automated, so thus we consider it to have a medium automation level.

- **Implementation of transformations.** The T2M transformations are implemented by a syntactical parser. As for the implementation of the M2M transformations, they implement them by a set of transformation rules implemented with QVT.
- **Kind of code to migrate.** This approach is applied to legacy relational databases implemented in SQL.
- **Supporting Toolkit.** A toolkit is not implemented either to support the automation of this approach or to manage the models.

### 3.3.3.6 Rodríguez-Echevarría et al.

The works (Rodríguez-echeverría, Conejero, Linaje, Preciado, & Sánchez-Figueroa, 2010), (Rodríguez-Echeverría, Clemente, Preciado, & Sánchez-Figueroa, 2011), (Rodríguez-Echeverría, Conejero, Pedro J. Clemente, Pavón, & Sánchez-Figueroa, 2012), (Rodríguez-Echevarría, Clemente, Villalobos, & Sánchez-Figueroa, 2012), (Rodríguez-Echevarría, Conejero, Clemente, Villalobos, & Sánchez-Figueroa, 2012) propose a model-driven approach based on ADM to migrate traditional Web applications to Rich Internet Application (RIA) (Noda & Helwig, 2005), (Bozzon et al., 2006a), (Rossi, Urbina, Ginzburg, Distante, & Garrido, 2008), (Busch, Koch, & Unit, 2009). This approach is focused mainly on the migration of the presentation and navigation aspects of a Web application. This method is composed of four steps: 1) extraction of a **Java model**, a **JSP model** and a **XML model** by using MoDisco tool (Barbier, Bruneliere, Jouault, Lennon, & Madiot, 2010), (Bruneliere, Cabot, Jouault, & Madiot, 2010); 2) generation of a **KDM model** by M2M transformations; 3) manual restructuring of the *KDM model* taking into account RIA features and 4) projection into an existing RIA-extended model-driven Web engineering method to carry out the *forward engineering stage* and the code generation. In the following, we analyze the approach applying the criteria defined in 3.3.2.

- **Reengineering stages.** This approach addresses the three reengineering stages: 1) in the *reverse engineering stage*, they use MoDisco tool to extract *models* from the code (Java, JSP and XML) of a legacy Web application;

afterwards, *KDM model* are generated from these models, 2) in the *restructuring stage*, the *KDM model* is restructured taking into account RIA features and 3) in the *forward engineering stage*, they propose the selection of an existing model-driven Web engineering method to address this stage. Figure 3-16 shows the correspondence between the reengineering stages and the steps of this reengineering method.

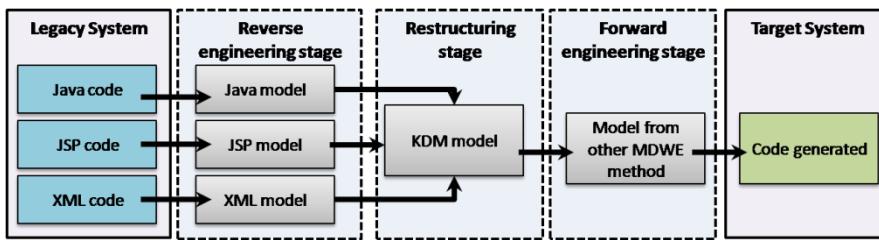


Figure 3-16. Modernization of legacy Web applications to RIA.

- **Scope of the approach.** This approach is focused on the migration of a traditional Web application to a RIAs.
- **MDA abstraction levels.** The *Java model*, *JSP model* and *XML model* extracted from the code are defined at PSM level, whereas the *KDM model* is defined at PIM level. The *restructuring stage* is carried out on the *KDM model* at PIM level.
- **Metamodels considered.** The *Java model* conforms to the Java metamodel as well as the *JSP model* is based on the JSP metamodel and the *XML model* on the XML metamodel. These metamodels have been defined specifically by this approach. On the other hand, they use the KDM metamodel to define the model at PIM level, concretely they are based on the ui and code packages.
- **Automation Level.** They automate the extraction of the *Java model*, the *JSP model* and the *XML model*. On the other hand, to obtain the rest of the models, they define M2M transformations. They do not address the code generation since they delegate this task to the model-driven Web engineering method selected. Thus, we can consider this approach having a medium automation level since they assure a partial automation.
- **Implementation of transformations.** To extract the *Java model*, the *JSP model* and the *XML model*, they rely on the MoDisco tool, so that they do not implement any parser. For the implementation of M2M transformations, they use the ATL language.

- **Kind of code to migrate.** This method is applied to legacy Web applications implemented in Java, JSP and XML.
- **Supporting Toolkit.** This automation of this approach is not supported by any toolkit, otherwise the edition of models is supported by graphical editors.

### 3.3.3.7 Van Hoorn et al.

The work (Van Hoorn et al., 2011) presents a reengineering method called *DynaMod* which addresses the model-driven modernization of software systems. This method is composed of three steps: 1) the extraction of a **Java architectural model** from Java which conforms to the KDM metamodel, 2) the *Java architectural model* is restructured with extra information provided by system experts, finally 3) the automatic generation of the **target architectural models** for the target system conforming to the KDM metamodel. From these *target architectural models*, the software artefacts are generated automatically. In the following, we analyze the approach applying the criteria defined in 3.3.2.

- **Reengineering stages.** This approach addresses the three reengineering stages: 1) in the *reverse engineering stage*, the *Java architectural model* conforming to the KDM metamodel is extracted, 2) in the *restructuring stage*, the *Java architectural model* is restructured manually with extra information, finally 3) in the *forward engineering stage*, the *target architectural models* are generated automatically from the restructured model and the software artefacts are generated by using a set of patterns. Figure 3-17 shows the matching between the reengineering stages and the steps of *Dynamod*.

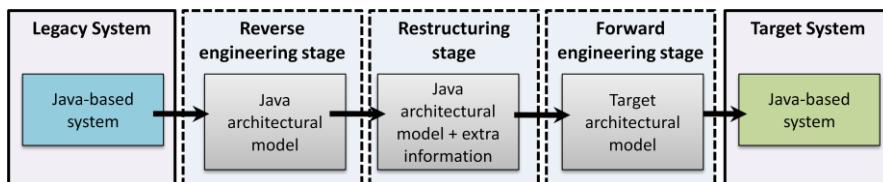


Figure 3-17. *Dynamod* method.

- **Scope of the approach.** *Dynamod* is not focused on any specific domain. It is defined as an approach to modernize legacy information systems.
- **MDA abstraction levels.** All the models used by *Dynamod* are defined at PIM level. It is noteworthy that, the models extracted from the code are not defined at PSM level and the code generation is run from a model at PIM level instead of one at PSM, as usually.

- **Metamodels considered.** All the models considered by *Dynamod* conforms to the KDM metamodel.
- **Automation Level.** To extract the models from the code, they implement T2M transformations. Furthermore, to generate automatically the *target architectural model*, they define M2M transformations. Finally, the code generation is specified by M2T transformations. We consider this approach having a high automation level since automates all the reengineering process.
- **Implementation of transformations.** The T2M transformations are implemented by a parser. As for the implementation of M2M transformations, they do not explain it in any work. Finally, the M2T transformations are implemented by using templates.
- **Kind of code to migrate.** *Dynamod* is applied to legacy Java-based systems.
- **Supporting Toolkit.** *Dynamod* is not supported by any toolkit for systematizing the reengineering process or editing the models.

### 3.3.3.8 Sadovsky et al

The work (Sadovskykh, Vigier, Gomez, & Hoffmann, 2009) presents a method for migrating legacy systems implemented in C++ to target systems implemented in Java. It addresses three steps: 1) the extraction of a platform-specific **UML model** from the C++ code conforming to the UML2 metamodel (Booch, Rumbaugh, & Jacobson, 1999), (OMG, 2005b), 2) the generation of a platform-independent **UML model** from the platform-specific *UML model*, 3) the restructuring of the generated platform-independent *UML model* eliminating platform dependencies and extracting business logic, finally 4) the generation of a platform-specific **Java UML model** from the platform-independent *UML model*. In the following, we analyze the approach applying the criteria defined in 3.3.2.

- **Reengineering stages.** This approach addresses the three reengineering stages1) in the *reverse engineering stage*, a platform-specific *UML model* is extracted from the C++ code. This model is subject of an abstraction transformation that aims at eliminating platform dependencies and extracting core business logic. Therefore, a platform-independent *UML model* is generated 2) in the *restructuring stage*, the generated platform-independent *UML model* is restructured and validated, finally 3) in the *forward engineering stage*, a platform-specific *Java UML model* is generated and validated from the platform-independent *UML model*. Figure 3-18 shows the

correspondence between the reengineering stages and the steps of this reengineering method.

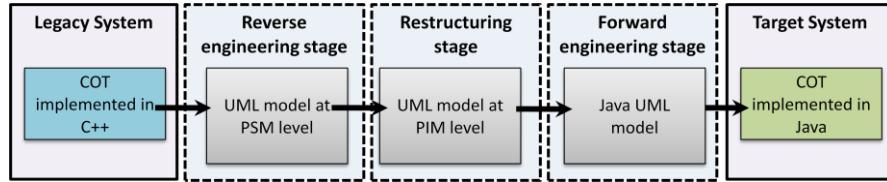


Figure 3-18. Sadovskykh et al. method.

- **Scope of the approach.** This approach is focused on Commercial-Off-The-Shelf Software (COTS).
- **MDA abstraction levels.** The UML models specified by this method are defined at PSM level and at PIM level. The *UML model* extracted from the code (the first one) as well as the *Java UML model* are defined at PSM level, whereas the *UML model* obtained from the first one is defined at PIM level.
- **Metamodels considered.** In accordance to the fourth criterion, we can say that all the models considered by this method, at PSM and at PIM levels, conform to a unique metamodel, the UML 2 metamodel.
- **Automation Level.** To extract the *UML model* from the code, they define T2M transformations. To generate automatically the platform-independent *UML model* and the *Java UML model*, they define M2M transformations. Finally, to generate the code they specify M2T transformations. The automation level of this approach is considered to be high since it automates comprehensively the reengineering process.
- **Implementation of transformations.** To implement the T2M transformations they use a parser. For the M2M transformations, they do not explain their implementation. As for the M2T transformations, they implement the generation of code by using templates.
- **Kind of code to migrate.** This method is applied to legacy COTS implemented in C++.
- **Supporting Toolkit.** This approach is supported by the *UML 2 Toolkit* (Eriksson, Penker, Lyons, & Fado, 2003) to generate the *UML model* at PIM level, to derive the *Java UML model* at PSM and to generate the Java code that implement the target system.

### 3.3.3.9 Vasilecas et al

The work (Vasilecas & Normantas, 2011a) presents a model-driven process for extracting business rules from existing IS. This method is composed of the following steps: 1) the extraction of an **ASTM model** conforming to the ASTM metamodel from the source code of a legacy, 2) the *ASTM model* is mapped to a **Code model** conforming to the KDM metamodel, 3) the application of software design recover techniques to the *KDM model* to extract business rules, this techniques are based on the *GUIDE Business Rule project* (Hay, Healy, & Hall, 2000) that classifies the business rules into four categories: business terms, facts, constraints, and derivations, finally 4) with the extracted and classified business rules a **KDM conceptual model** is created. This model conforms to the *Semantics of Business Vocabulary and Business Rules* (SBVR) (OMG, 2008b). In the following, we analyze the approach applying the criteria defined in 3.3.2.

- **Reengineering stages.** This approach addresses two of the reengineering stages: 1) in the *reverse engineering stage*, an *ASTM model* is extracted from the source code of a legacy IS. Then, from this *ASTM model*, a *KDM model* is generated automatically and 2) in the *restructuring stage*, the KDM model is analyzed by applying the design recover techniques to extract business rules which are captured in a *KDM conceptual model*. Figure 3-19 shows the matching between the reengineering stages and the tasks composing this method.

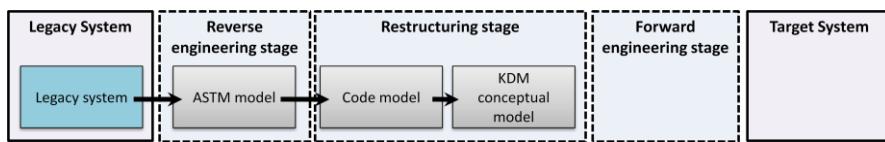


Figure 3-19. Vasilecas et al. method.

- **Scope of the approach.** This approach is not focused on any specific domain. It is defined as an approach to extract business rules from any legacy information systems.
- **MDA abstraction levels.** The *ASTM model* is defined at PSM level because describes the source code of the legacy system, whereas the *Code model* is defined at PIM level. The restructuring of the *Code model* to obtain the *KDM conceptual model* is carried out at PIM level, but the resulting *KDM conceptual model* is defined at CIM level.

- **Metamodels considered.** The *ASTM model* conforms to the ASTM metamodel whose SASTM domain captures the specific elements of the source code. The *Code model* conforms to the KDM metamodel, specifically to the code and action packages. Finally, the *KDM conceptual model* conforms to SBVR, the standard of OMG intended to be the basis for formal and detailed natural language declarative description of business rules.
- **Automation Level.** For the extraction of the *ASTM model*, they define T2M transformations, and for the automatic generation of the *Code model*, they specify M2M transformations. The reengineering process is partially automated since the obtainment of the *KDM conceptual model* is carried out manually, so that its automation level is medium.
- **Implementation of transformations.** The T2M transformations are implemented by a parser implemented in Xtext, whereas the M2M transformations are implemented by transformation rules defined with ATL.
- **Kind of code to migrate.** This approach is applied to legacy information systems without specifying a specific type.
- **Supporting Toolkit.** This method is supported by a set of tools based on EMF. The transformations are supported by the ATL IDE, whereas to edit, query and import/export to XMI format the KDM models, they use the KDM SDL toolkit.

### 3.3.4 Discussion

In the previous section, we have analyzed all the model-driven approaches that somehow tackle a reengineering process. In this section, we use the information extracted in the previous section to compare them and extract the conclusions to answer the research questions (RQ) defined in Section 3.3.1. Table 3-2 sums up the information extracted by considering the evaluation criteria.

**Table 3-2. Data extraction from the model-driven reengineering approaches.**

<i>Approach</i>	<i>Reengineering Stages</i>	<i>Scope</i>	<i>MDA levels</i>	<i>Metamodels</i>	<i>Automation level</i>	<i>Implementation transformations</i>	<i>code</i>	<i>Toolkit</i>
García-Rodríguez et al. (Rational Web)	Reverse, restructuring, forward	Databases	PSM, PIM	OOS	High	T2M (parser) M2M (not implemented) M2T (code generator)	SQL	Automating transformations
Blanco et al.	Reverse, restructuring, forward	Databases (OLAP)	PSM, PIM	SECMDDW, SECDW	High	T2M (parser) M2M (QVT) M2T (code generator)	--	Automating transformations
Perez-Castillo et al. (MARBLE)	Reverse, restructuring	Legacy systems	PSM, PIM, CIM	Java, KDM, BPMN	High	T2M (parser) M2M (QVT)	Java	Automating transformations
Perez-Castillo et al. (PRECISO)	Reverse, restructuring, forward	Databases	PSM, PIM	SQL-92, UML2	High	T2M (parser) M2M (not implemented) M2T (code generator)	SQL-92	Automating Transformations + model edition
Perez-Castillo et al. (Schema Elicitation Method)	Reverse, restructuring, forward	Databases	PSM, PIM	DML, KDM	Medium	T2M (parser) M2M (QVT) M2T (not implemented)	SQL	Not defined
Rodríguez-Echevarría et al	Reverse, restructuring, forward	RIA	PSM, PIM	Java, JSP and XML, KDM	Medium	T2M (Modisco) M2M (ATL) M2T (not implemented)	Java, JSP, XML	model edition
Van Hoorn et al. (Dynamod)	Reverse, restructuring, forward	Legacy systems	PIM	KDM	High	T2M (parser) M2M (not implemented) M2T (templates)	Java	Not defined

<i>Approach</i>	<i>Reengineering Stages</i>	<i>Scope</i>	<i>MDA levels</i>	<i>Metamodels</i>	<i>Automation level</i>	<i>Implementation transformations</i>	<i>code</i>	<i>Toolkit</i>
Sadovykh et al.	Reverse, restructuring, forward	COTS	PSM, PIM	UML2	High	T2M (parser) M2M (not implemented) M2T (templates)	C++	Automating transformations
Vasilecas et al.	Reverse, restructuring	Legacy systems	PSM, PIM, CIM	ASTM, KDM, SBVR	Medium	T2M (parser Xtext) M2M (ATL)	--	Automating Transformations + model edition
Trias et al. ( <i>ADMigraCMS</i> )	Reverse, restructuring, forward	CMS-based Web applications	PSM, PIM	ASTM_PHP, KDM, CMS Common Metamodel	High	T2M (parser Xtext, Java) M2M (ATL) M2T (Acceleo)	PHP	Automating Transformations + model edition

- **RQ1. Which model-driven approaches do exist in the literature for addressing a reengineering process?**

The SLR performed ended up in finding 9 approaches from the 27 primary studies found. In the following, we present them.

The García-Rodríguez de Guzmán et al. (Rational Web) approach proposes a method designed specially for the reengineering of relational databases. This method allows the automatic generation of different multilayer applications from relational databases.

Blanco et al. presents an ADM-based method focused on obtaining conceptual security models from legacy OLAP systems. It allows to analyze OLAP systems in order to include new security requirements to improve the architecture as well as to migrate to other platforms.

The approach presented by Pérez-Castillo et al. is MARBLE, an ADM-based framework for recovering business processes from legacy systems.

The García-Rodríguez de Guzmán et al. (PRECISO) approach presents a reverse engineering method to extract Web Services from relational databases in automatic manner.

García-Rodríguez de Guzmán et al. (Schema Elicitation Method) proposes a method for modernizing the legacy source code of a relational database.

Rodríguez-Echevarría et al. presents a model-driven approach based on ADM to migrate traditional Web applications to RIA.

Van Hoorn et al. presents a reengineering method called *DynaMod* which addresses the model-driven modernization of software systems.

Sadovych et al. presents a method for migrating legacy systems implemented in C++ to target systems implemented in Java.

Finally, Vasilecas et al. proposes a model-driven process for extracting business rules from existing IS.

- **RQ2. In which domain does the Software Reengineering Community dedicate more efforts to define model-driven approaches to automate a reengineering process?**

As for the scope addressed by the approaches, we can say that three out of nine are focused on databases (García-Rodríguez de Guzmán et al., Perez-Castillo et al. (PRECISO) and Perez-Castillo et al. (Schema Elicitation Method)); one

approach is centred in OLAP systems (Blanco et al.); three out of nine are generic reengineering processes without specifying its application to a specific type of information system (tagged as legacy systems in Table 3-2) (Perez-Castillo et al. (MARBLE), Hoorn et al. (Dynamod) and Vasilecas et al.); another approach is focused in COTS (Sadovskyh et al.) and finally, another one in RIAs (Rodriguez-Echevarria et al.). After the performance of this SLR, we can say that databases is the domain in which the different research groups have devoted more efforts to define model-driven approaches to automate the reengineering process. Also, we can find a set of approaches that are not defined for a specific domain and they are applicable to legacy systems, in general.

- **RQ3. Which kind of process is defined by the approaches? Which reengineering stages do they consider in their process?**

This SLR allows us to conclude that, most of the model-driven reengineering approaches found in the literature address a horseshoe process which include the three classical reengineering stages. Two of these approaches, Perez-Castillo et al. (MARBLE) and Vasilecas et al., address just two of reengineering stages, the reverse engineering stage and the restructuring stage.

- **RQ4. Which is the level of automation of the approaches found in the literature? Are they supported by any toolkit?**

The level of automation of the approaches found is high since practically all of them (six out of nine) consider automated transformations (T2M, M2M and M2T) to systemize the reengineering process. Transformations trying to automate the reengineering process. It worth noting that, if the approach considers the restructuring stage where the legacy system is restructured or redefined, it is usually carried out manually by experts. As for the toolkit supporting the approaches, we can say that most of the approaches are supported by toolkits (seven out of nine). We can classify the toolkits in those supporting the automated transformations and those which allow the edition of models involved in the reengineering process. Out of the approaches supported by a toolkit, we found four approaches which just consider the supporting of the automated transformations by a toolkit and one approach considering just the edition of models by a tool (Rodríguez-Echevarría et al.). Finally, two of them, Pérez-Castillo (PRECISO) and Vasilecas et al., consider both.

- **RQ5. Which type of techniques are used by the approaches to extract and model the software artefacts from the legacy system? In which code are the software artefacts implemented? Which is the technique to generate code?**

The most used technique to extract models from the software artefacts from the legacy systems is the definition and implementation of a parser. These parsers implements the T2M transformations involved in the approaches. Eight out of nine use parsers and one (Rodríguez-Echevarría et al.) uses the toll MoDisco to extract these models. The works found do not explain in detail the implementation of these parsers. Just Vasilecas et al. mentions that its parser is implemented using the Xtext framework. As for the code from which the models are extracted, we can see that the approaches focused on the databases scope extract models from SQL code. On the other hand, three of the approaches extract models from Java code. It is worth noting that the approach Rodríguez-Echevarría et al. apart from extracting models from Java, it extracts from JSP and XML codes. One approach, Sadovych et al. extracts models from C++. Most of these models extracted from the code are defined at PSM level. Just Van Hoorn et al. extracts models defined at PIM level. After the execution of this SLR, we realized that none of them extract models from the PHP code. For *ADMigracMS*, we have implemented a parser using Java and the Xtext framework to extract PSM models from PHP code.

As for the generation of code, the technique more used by the approaches to implement the M2T transformations is the code generator. The works found address the implementation of these code generators, either. Just two approaches, Van Hoorn et al. and Sadovykh et al., comment that they base this implementation on patterns without specifying the transformation language used. Usually, the code is generated from models defined at PSM level, but two of the approaches analyzed, Hoorn et al. and Blanco et al., generate the code from models at PIM level. For *ADMigracMS* method, we have used patterns implemented with Acceleo (Musset et al., 2008) to defined our parser.

- **RQ6. Which type of techniques are used by the approaches to implement the M2M transformations? Which are the MDA level of the models transformed by these M2M transformations?**

All the approaches found in the literature define M2M transformations to automate their reengineering process. Most of the approaches implement these transformations defining transformation rules implemented with a concrete transformation language. Three of them, Perez-Castillo et al. (MARBLE), Blanco

et al. and Perez-Castillo et al. (Schema Elicitation Method) use the transformation language QVT (OMG, 2008a) to implement these M2M transformations, otherwise two of the approaches Rodriguez-Echevarria et al. and Vasilecas et al. use ATL (Jouault et al., 2006).

The M2M defined in the reverse engineering stage are to arise the abstraction level of the models. All of the approaches define M2M transformations between PSM models and PIM models except the Van Hoorn et al. approach that defines transformations between models at PIM level. Otherwise, two approaches, Perez-Castillo et al. (MARBLE) and Vasilecas et al., define M2M transformations between PIM models and CIM models. All the approaches carry out the restructuring stage at PIM level, except these two ones which do it at CIM level. the restructuring stage in our *ADMigraCMS* method is performed at PIM level.

- **RQ7: Which is the use of the standard metamodels proposed by ADM among the reengineering approaches?**

Regarding the metamodels used by the approaches, we can say that the range of metamodels used by these approaches is wide. We present the metamodels used considering the MDA abstraction levels. The OOS metamodel is used by the approach presented by Garcia-Rodriguez et al. This metamodel allows to define models at PSM and PIM levels. The metamodels SECMDDW and SECDW are used by the approach proposed by Blanco et al. These metamodels allow to model the system's security configuration by using a role-based access control policy. The SECMDDW metamodel defines the models at PSM level and the SECDW metamodel specifies the models at PIM level. Otherwise, the Data Manipulation Language (DML) is used by the approach presented by Perez-Castillo et al. (Schema Elicitation Method) to define the PSM models. The Java metamodel is used by two approaches Perez-Castillo et al. (MARBLE) and Rodriguez-Echevarria et al. to model at PSM level the Java code implementing the software artefacts of the legacy system. On the other hand, there are three other specific-platform metamodels which allow to define models at PSM level. These metamodels are the JSP metamodel and the XML metamodel used by the approach presented by Rodriguez-Echevarria et al. and the SQL metamodel considered by the Perez-Castillo et al. (PRECISO). The UML 2 metamodel is considered by three approaches (Perez-Castillo et al. (PRECISO), Perez-Castillo et al. (Schema Elicitation Method) and Sadovskykh et al. These three approaches use the UML 2 metamodel to define their models at PIM level. Moreover, the approach presented by Sadovskykh et al uses the UML 2 metamodel to define the

models at PSM level. The BPMN metamodel is used to define the CIM models considered by MARBLE.

As for the standard metamodels proposed by ADM, we can say that KDM is used in four approaches Perez-Castillo et al. (MARBLE), Rodriguez-Echevarria et al., Van Hoorn et al. and Vasilecas et al. to define their PIM models. Otherwise, the ASTM standard metamodel to define PSM models is used by Vasilecas et al. approach. We conclude that the use of the standard metamodels proposed by ADM is quite spread. For *ADMigraCMS* method, we use the KDM metamodel to define the PIM metamodels and an adaptation of the ASTM metamodel (which we call ASTM\_PHP) at the time of extracting the PSM models from the legacy software artefacts implemented in PHP.

- **RQ8. Which is the interest of the Web Engineering community in researching about CMS-based Web applications?**

The model-driven reengineering is a very interesting topic for the research community since we have found many model-driven approaches focusing on the automation of the reengineering process. We can state that the most active research group in this area is Alarcos from the Castilla-la-Mancha University.

Regarding the forums where the primary studies were published, we conclude that they are recent forums (from 2004 to 2012). Moreover, most of them are rated as high-level contributions since they have been published in CORE-A and CORE-B conferences.

Finally, we would like to comment that with this SLR we reaffirm that there is not any approach framed in the scope of CMS-based Web applications to carry out an automatic reengineering process. Therefore, with the work presented in this PhD Thesis, we intend to solve this gap.

### 3.4 Concluding Remarks

In this section, we extract some concluding remarks from the two SLRs executed in this chapter.

From the first SLR about **approaches related to the development of CMS-based Web applications**, we can say that the number of approaches focused in this issue is scarce (four approaches) despite the manifest necessity for a specific method to support the development of CMS-based Web applications.

All the approaches found define a top-down development process. *We have not found any approach defining a reengineering process to migrate CMS-based Web applications.*

All the approaches use models as the key artefact to carry out their development processes. All of them consider models at PIM level to design and represent the CMS-based Web application being developed. Only the approach Souer et al. (WEM) considers, in addition, models at CIM level and none of them considers models at PSM level.

The modelling languages used to build these models are specific modelling languages defined by each approach, i.e. Souer et al. (WebForm diagram) defines the WebForm modelling language and Saraiva et al. defines the CMS-ML and CMS-IL languages. Only, two approaches use the standard modelling language UML to define some of their models.

Apart from the use of models, most of them, except the Souer et al. (WEM), defined automated transformations to systemize the development process. Thus, we state that most of them are considered model-driven approaches. Being approaches centred in the top-down development, all of them take into account only M2M and M2T transformations.

The toolkit support level of these approaches is low since just one approach Vlaanderen et al. is supported by a thorough toolkit called OlivaNova. The other ones are not supported by any toolkit, such as Souer et al. (WEM) and Saraiva et al., or they are partially.

From the second SLR about **model-driven reengineering methods**, we can state that most of the approaches found are framed within the domain of databases, four out of seven. *We have not found any approach focused on the migration of CMS-based Web applications.*

Moreover, most of them cover the three reengineering stages proposed by Chikofsky: reverse engineering, restructuring and forward engineering. Only Pérez-Castillo et al.(MARBLE) and Vasilecas et al. consider two of these stages: reverse engineering and restructuring.

Being model-driven reengineering approaches, they use models as the key artefacts to represent the knowledge involved within the reengineering process and automated transformation (T2M, M2M and M2T transformations) which systemize it. The models are defined at different abstraction levels. Most of the models obtained by the T2M transformations during the reverse engineering stage

are defined at PSM level. These models conform to platform-specific metamodels such as OOS, Java, SQL-92, JSP or XML. Only one approach, Vasilecas et al., uses the standard metamodel ASTM to represent its PSM model.

Most of the approaches carry out the restructuring stage with models defined at PIM level. Only two approaches, Pérez-Castillo et al.(MARBLE) and Vasilecas et al., execute their restructuring stage with models at CIM level. Five out of nine approaches, use the standard metamodel KDM to define their models at PIM level, whereas the approaches that define models at CIM level use the standard metamodels BPMN and SBVR. We can remark that the use of the standard metamodels proposed by ADM (KDM, ASTM and SBVR) is quite extended, mainly the KDM metamodel.

As for the automated transformations, most of the approaches implement the T2M transformations by means of parsers that read a file written in a specific code and generate a model. Some approaches extract models from the following codes: SQL-92, Java, JSP, XML or C++, but we did not find any approach extracting from PHP code.

Otherwise, the M2M transformations are implemented by transformation rules defined in a transformation language, QVT or ATL. As for the M2T transformations, the approaches define them by code generators without specifying their implementation. Only two approaches, Van Hoorn et al (Dynamod) and Sadovykh et al. implement them by templates.

Most of these approaches are supported by toolkits that mainly support the automated transformations rather than the model edition.

After the execution of these two SLRs, we think that this PhD Thesis covers a necessity that currently is not tackled by another approach in the literature. We propose a method for migrating CMS-based Web applications to other CMS platforms using the standard metamodels proposed by ADM. Moreover, we extract the knowledge involved in this migration process from PHP code.



*Chapter 4: CMS Common  
Metamodel Specification*

---



After executing the two SLRs in the previous chapter, we did not find any model-driven engineering methods providing developers with a comprehensive modelling language to model CMS-based Web applications in the CMS domain. Therefore, in the context of this PhD Thesis we propose the CMS DSL, a modelling language for modelling CMS-based Web applications.

In this chapter, we present the CMS Common Metamodel, the metamodel that specifies the abstract syntax of the CMS DSL proposed in the *ADMigraCMS* method. It captures the key elements of the CMS domain required to define the CMS model.

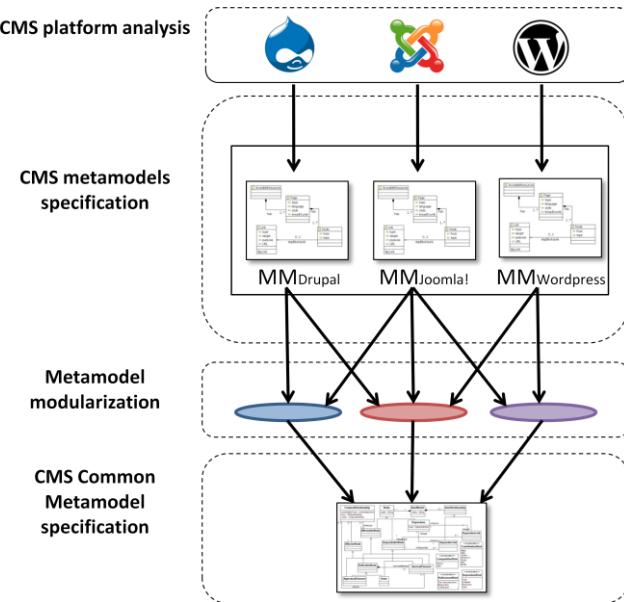
The process to define this metamodel is composed of four steps: 1) *CMS platform analysis*, where we study the CMS market and we choose the three most used CMS platforms (explained in Section 4.2); 2) *CMS metamodels specification*, specification of a metamodel for each of those three CMS platforms capturing their key aspects (described in Section 4.3); 3) *Metamodel modularization*, the definition of a set of concerns in order to classify the elements of those CMS metamodels (depicted in Section 4.4) and 4) *CMS Common Metamodel specification*, specification of the CMS Common Metamodel making the union of the elements of the previous three metamodels (defined in Section 4.5). Finally, in Section 4.6, we present the concluding remarks of this chapter.

## 4.1 Process to Define the CMS Common Metamodel

In this section, we present the four steps we followed to define the CMS Common Metamodel. In Figure 4-1 presents graphically the process.

- 1. CMS platform analysis:** We analyze the CMS market to discover the most used CMS platforms by organizations. After the study of the CMS market, we select the three most used CMS platforms (Wordpress, Joomla! and Drupal) and we analyze their main features applying seven of the criteria defined in (Plain Black Corporation, 2010): 1) *databases*, indicates the database associated with the CMS platform, 2) *programming language*, indicates the programming language implementing the CMS platform, 3) *security requirements*, specifies the security mechanisms supported by the CMS platform, 4) *support*, presents the ways to support developers in the use of the CMS platform, 5) *ease of use*, indicates the tools provided by the CMS

platform to ease its use, 6) *management requirements*, presents the aspects which can be managed by the CMS platforms and 7) *flexibility features*, prove the flexibility of the CMS platform to cope with different contexts. The analysis of the CMS platforms using these criteria allows us to know thoroughly the features of each CMS platform.



**Figure 4-1. Process to define the CMS Common Metamodel.**

2. **CMS metamodels specification:** After the analysis of the CMS platforms, we specify the metamodels for each CMS platform (CMS metamodels) to capture the key elements and their relationships of the CMS domain. We need to define a common language to capture the features of each CMS platform so that we defined a CMS metamodel for each one. Therefore, we defined a Wordpress metamodel (MMWordpress), a Joomla! metamodel (MMJoomla) and a Drupal metamodel (MMDrupal).
3. **Metamodel modularization:** After the identification and definition of the key elements and their relationships of each CMS platform by defining the CMS metamodels, we specify a set of concerns which allow us to modularize the CMS metamodels. According to the complexity of the CMS domain, we experience the necessity of modularizing the CMS metamodels. Therefore, we define five concerns based on the Web Engineering views proposed by

(Deshpande et al., 2002), (Ginige, 2002), (Kappel, Präüll, et al., 2006): content, navigation, presentation, behaviour and user.

4. **CMS Common Metamodel specification:** After defining the three CMS metamodels, we built the CMS Common Metamodel. To build it we decided to create the union of the three previous metamodels. Thus, the resulting CMS Common Metamodel contains the common elements as well as the specific elements of each platform to assure that our metamodel allows the modelling and the implementation of a CMS-based Web application based in any of these CMS platforms.

## 4.2 CMS Platform Analysis

In this section, we present the first step in the process of specifying the CMS Common Metamodel. In this step we carry out an analysis of the CMS market to know which three CMS platforms are most used by organizations. Then, we analyze these three CMS platforms applying the criteria defined in the previous section.

### 4.2.1 CMS Market

In this section, we study the market of the current existing CMS platforms to discover the most used ones by organizations. This study allowed us to select the three main CMS platforms to base a proposal for CMS Common Metamodel. For the study of this market we considered the two surveys carried out by Ric Shreves in (Ric Shreves, 2008) and (Shreves, 2011) which conclude that for 4 years in a row the top three leaders in the CMS market were: Wordpress (Wordpress, 2014), Joomla! (Joomla!, 2014) and Drupal (Drupal, 2014), called the *Big Three* (Shreves, 2011). Moreover, the study found in the Web page (W3Techs, 2014) also reveals that the three most spread CMS platforms from 2010 to 2013 were, again, Wordpress, Joomla! and Drupal. It is worth noting that these CMS platforms are under the GNU/GPL licence, i.e. they are open-source. Table 4-1 shows the market share from 2010 to 2014 extracted from (W3Techs, 2014).

**Table 4-1. Market share yearly trends from 2010 to 2014.**

	2010		2011		2012		2013		2014	
<b>WORDPRESS</b>	51%		55.3%		54.3%		54.8%		60.3%	
<b>JOOMLA!</b>	12%		10.9%		9.5%		8.7%		8.2%	
<b>DRUPAL</b>	7.1%	70.1%	6.1%	72.3%	6.5%	70.3%	7.2%	70.7%	5.2%	73.7%
<b>BLOGGER</b>	--		2.7%		3%		3.5%		3.0%	

<b>MAGENTO</b>	--	--	--	2.7%	2.7%
<b>TYPO3</b>	4.2%	2.6%	2%	2.1%	1.6%
<b>VBULLETIN</b>	8.4%	5.9%	4.4%	3.5%	1.2%
<b>Others</b>	17.3%	16.5%	20.3%	17.5%	17.8%

As we can see in Table 4-1 the highest percentages of market share from 2010 up to 2014 are assigned to Wordpress, Joomla! and Drupal, with over a 70% market share. Furthermore, Table 4-1 shows an increase in the use of these open-source CMS platforms from 2010 to 2014 (70.1% to 73.7%).

For all these reason, we selected Wordpress (Wordpress, 2014), Joomla! (Joomla!, 2014) and Drupal platforms to define the CMS Common Metamodel. In the following sections, we analyze the features of each CMS platform.

#### 4.2.2 Wordpress

In this section we analyze the Wordpress platform considering the criteria defined in Section 4.1. This analysis is sum up in Table 4-2.

- **Databases:** Wordpress uses MySQL as the database to store the content
- **Programming language:** Wordpress is implemented with PHP code.
- **Security requirements:** Wordpress allows to perform an email verification, to carry out a content approval, to define a set of privileges in order to control the access of the users as well as the management of different sessions by installing a new plugin.
- **Support:** Wordpress provides with an active developer community, an online help, public forums and user conferences supporting users at the time of developing a Web application by using Wordpress.
- **Ease of use:** Wordpress provides users with WYSIWYG editors to add or edit new content. Also, it allows users to displace content by using the drag-and-drop technique, to use of friendly urls and to resize images.
- **Management requirements:** Wordpress does not permit the management of advertising, the content staging or the package deployment. It provides with a workflow engine neither. On the contrary, it allows the management of different assets and the management of the look and feel of the Web application. Finally, it allows a limited translation management.

- **Flexibility features:** Wordpress allows the content reuse, the use of metadata to tag the content and the multisite deployment. Also, it allows the multi-lingual content by adding a plugin.

#### **4.2.3 Joomla!**

As occurred with Wordpress, in this section we present the analysis of Joomla! considering the criteria defined in Section 4.1. Table 4-2 sums up this analysis.

- **Databases:** Joomla! uses MySQL as the database to store the content
- **Programming language:** Joomla! is implemented with PHP code.
- **Security requirements:** Joomla! allows to perform an email verification, to carry out a content approval, to define a set of privileges in order to control the access of the users as well as the management of different sessions.
- **Support:** Joomla! provides with online help, public forums and user conferences supporting users at the time of developing a Web application by using Joomla!.
- **Ease of use:** Joomla! provides users with WYSIWYG editors to add or edit new content, the use of friendly urls and image resizing, but it does not allow the drag-n-drop content.
- **Management requirements:** Joomla! allows the management of advertising and other assets as well as the management of the look and feel of the Web application and a translation management by installing a new plugin.
- **Flexibility features:** Joomla! allows the content reuse and the use of metadata to tag the content. Also it allows the multi-lingual content and multi-site deployment by adding a plugin.

#### **4.2.4 Drupal**

Finally, in this section we present the analysis of Drupal considering the criteria defined in Section 4.1. Table 4-2 sums up this analysis.

- **Databases:** Drupal uses MySQL as the database to store the content
- **Programming language:** Drupal is implemented with PHP code.

- **Security requirements:** Drupal allows to perform an email verification, to carry out a content approval, to define a set of privileges in order to control the access of the users as well as the management of different sessions.
- **Support:** Drupal provides with a developer community, an online help, public forums and user conferences supporting users at the time of developing a Web application by using Drupal.
- **Ease of use:** Drupal provides users with WYSIWYG editors to add or edit new content by installing a plugin. Also, by installing a plugin, it allows the drag-n-drop and the content image resizing. Otherwise, by default, it allows the use of friendly urls.
- **Management requirements:** Drupal allows the management of advertising by installing a plugin and the management of other assets as well as the management of the look and feel of the Web application and a translation management.
- **Flexibility features:** Drupal allows a limited content reuse and the use of metadata to tag the content. Also it allows the multi-lingual content and multi-site deployment.

**Table 4-2. Analysis of the three CMS platforms.**

	<i>WORDPRESS</i>	<i>JOOMLA!</i>	<i>DRUPAL</i>
<i>DATABASE</i>	MySQL	MySQL	MySQL
<i>PROG. LANGUAGE</i>	PHP	PHP	PHP
<i>SECURITY REQUIREMENTS</i>	Email verification Content approval Granular privileges Multi-session	Email verification Content approval Granular privileges Multi-session	Email verification Content approval Granular privileges Multi-session
<i>SUPPORT</i>	Developer community Online help public forums	Developer community Online help public forums user conferences	Developer community Online help public forums user conferences
<i>EASE OF USE</i>	WYSIWYG editors drag-n-drop content friendly urls image resizing.	WYSIWYG editors friendly urls image resizing.	WYSIWYG editors (plugin) drag-n-drop content (plugin) friendly urls
<i>MANAGEMENT REQUIREMENTS</i>	Assets Look and feel translation	Advertising Assets Look and feel translation	Advertising Assets Look and feel translation
<i>FLEXIBILITY FEATURES</i>	Content reuse Metadata use Multi-lingual Multi-site	Content reuse Metadata use Multi-lingual (plugin) Multi-site (plugin)	Content reuse (limited) Metadata use Multi-lingual Multi-site

### 4.3 CMS Metamodels Specification

In this section, we present the CMS metamodels we have defined for each CMS platform. These CMS metamodels are the common language used to define the features of each of them. Therefore, these metamodels capture the main elements and their relationships of each CMS platform.

In the following, for each CMS metamodel, we present a graphic of the metamodel and two tables one explaining its elements and the other one its relationships between these elements.. This former describes the name of the element, its attributes and a description, whereas the latter presents the name of the relationship, its source element, its target element, and the explanation of its semantic within the metamodel.

#### 4.3.1 *Wordpress Metamodel*

In this subsection, we present the Wordpress metamodel (MMWordpress) which captures the key elements and of this platform. Figure 4-2 shows the elements that compose this metamodel, their attributes and the relationships among them. In Table 4-3 we explain the elements of MMWordpress and in Table 4-4 we present its relationships.

96 Feliu Trias

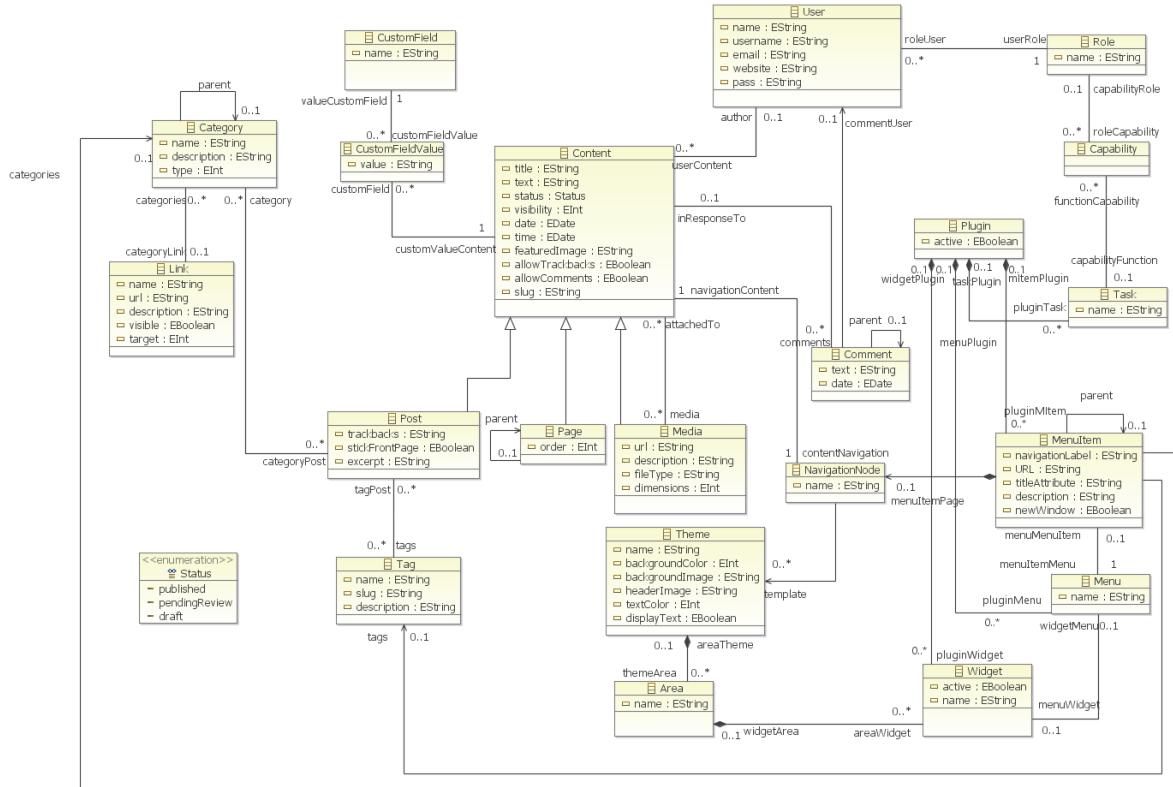


Figure 4-2. MMWordpress

**Table 4-3. Elements of MMWordpress.**

ELEMENTS OF THE MMWORDPRESS		
NAME	ATTRIBUTES	DESCRIPTION
Content	Title, status, visibility, date, time, featuredimage, allowTrackbacks, allowComments, slug	This element represents the information displayed on a Web page.
Post	Trackbacks, stickFrontPage, excerpt, page, order	This element is a specialization of the element <i>Content</i> which represent a specific type of content within Wordpress. A <i>Post</i> is a blog entry with dynamic content.
Media	Url, description, fieldType, dimensions	This element is a specialization of the element <i>Content</i> and represents the multimedia content which can be added to the pages of the Web application.
Theme	Name, backgroundColour, backgroundImage, headerImage, text colour, displayText	This element represents the presentation aspect of a page. A <i>Theme</i> defines the structure and look and feel the page.
Navigation Node	Name	This element represents a page belonging to the Web application.
Tag	Name, slug, description, Area, name, Role, name	This element represents the extra information related to a content. A tag is a term which is used to classify the information and to ease the search of content within the Web application.
Capability	Name	This element represents the functions associated with a certain role. Depending on this capabilities the user is able to perform some functions or other ones.
Task	Name	This element represents a task or functionality which a plugin is able to perform.
Plugin	Active	This element represents Wordpress plugins. A plugin provides an extension of the functionality of the Web application.
User	Name, username, email, website, pass	This element represents the users which interact with the Web application.
Comment	Text, date	This element represents the comments inserted by the users of the Web applications and they are related to the content displayed on the pages.
MenuItem	NavigationLabel, URL, titleAttribute, description, newWindow	This element represents the links which allow users to navigate through the Web application.
Menu	Name	This element represents the menus of Wordpress. <i>Menus</i> are a set of navigational links to facilitate the content access.
Widget	Active, name	This element represents the widgets in Wordpress. They offer a additional functionality and they can be placed within the different areas of a <i>Theme</i> .
CustomField	Name	This element defines a field which is associated with a content to add metadata.

ELEMENTS OF THE MMWORDPRESS		
NAME	ATTRIBUTES	DESCRIPTION
CustomFieldValue	Value	This element defines the value associated to a field.
Category	Name, description, type, link, name, url, description, visible, target	This element represents a category in Wordpress. A <i>Category</i> allows to classify the content of the Web application.

Table 4-4 presents the main relationships established among the elements of the MMWordpress.

**Table 4-4. Relationships of MMWordpress.**

RELATIONSHIPS OF MMWORDPRESS			
NAME	SOURCE	TARGET	DESCRIPTION
parent	Page	Page	This reflexive relationship allows to associate a page with another one to create hierarchies of pages.
attachedTo	Media	Content	This relationship determines that a multimedia content is related to a content. There is an opposite relationship called <i>media</i> .
navigationContent	Navigation Node	Content	This relationship determines that a page have to be associated to a content. There is an opposite relationship called <i>contentNavigation</i> .
theme	Navigation Node	Theme	This relationship defines the theme of a page of the Web application.
themeArea	Theme	Area	This relationship allows to define the areas in which the theme is composed of. There is an opposite relationship called <i>areaTheme</i> .
tagPost	Tag	Post	This relationship allows to assign one or more tags to a post. There is an opposite relationship called <i>postTag</i> .
tags	Tag	Menu Item	This relationship determines that a tag is related to a menu item.
capabilityRole	Capability	Role	This relationship represents that a capability is associated to a unique user role. There is an opposite relationship called <i>roleCapability</i> .
roleUser	Role	User	This relationship defines that a role can be related to different users. There is an opposite relationship called <i>userRole</i> .
functionCapability	Task	Capability	This relationship defines that a task can be related to different capabilities. There is an opposite relationship called <i>capabilityFunction</i> .
pluginTask	Plugin	Task	This relationship defines that a plugin is composed of a set of functionalities. Moreover, there is an opposite relationship called <i>taskPlugin</i> .
autor	User	Content	This relationship indicates that a user is the creator of a content. Moreover, there is an opposite relationship called <i>userContent</i> .
commentUser	Comment	User	This relationship allows to associate a comment with an user indicating who is the creator.
parent	Comment	Comment	This reflexive relationship is used to represent a hierarchy of comments. It is possible to relate a comment with another one.

RELATIONSHIPS OF MMWORDPRESS			
NAME	SOURCE	TARGET	DESCRIPTION
inResponseTo	Comment	Content	This relationship indicates that a comment is related to a certain content. Moreover, there is an opposite relationship called <i>comments</i> .
widgetArea	Widget	Area	This relationship allows to assign the area where the widget will be located and displayed. An area can be composed of a set of different widgets. Otherwise a widget can be located just to one area.
widgetMenu	Widget	Menu	This relationship allows to relate a widget to a menu. Moreover, there is an opposite relationship called <i>menuWidget</i> .
menuMenuItem	Menu	Menu Item	This relationship determines that a menu is composed of a set of menu items. Moreover, there is an opposite relationship called <i>menuItemMenu</i> .
parent	Menu Item	Menu Item	This reflexive relationship allows to associate a menu item with another one creating a hierarchy of menu items.
menuItemPage	Menu Item	Navigation Node	This composition relationship defines the page to which a menu item is linked.
tags	Tag	Menu Item	This relationship associates a set of different tags to a menu item.
menuItemPage	Menu Item	Page	This composition relationship defines that a menu item is the link to access a page.
valueCustomField	Custom Field Value	Custom Field	This relationship allows to associate a value with its corresponding field. A field can have only one associated value. There is an opposite relationship called <i>customFieldValue</i> .
customField	Content	Custom Field Value	This relationship allows to associate a Content with the customized values of their fields. There is an opposite relationship called <i>customValueContent</i> .
categoryPost	Category	Post	This relationship determines that a category is associated with a set of posts. There is an opposite relationship called <i>category</i> .
parent	Category	Category	This reflexive relationship allows to create hierarchies of categories relating one category with another one.
categoryLink	Category	Link	This relationship specifies that a category can be associated with a set of links. There is an opposite relationship called <i>categories</i> .
pluginWidget	Plugin	Widget	This composition relationship specifies that a plugin can define by code a set of widgets. There is an opposite relationship called <i>widgetPlugin</i> .
pluginMItem	Plugin	Menu Item	This composition relationship specifies that a plugin can define by code a set of menu items. There is an opposite relationship called <i>mItemPlugin</i> .
pluginMenu	Plugin	Menu	This composition relationship specifies that a plugin can define by code a set of menus. There is an opposite relationship called <i>menuPlugin</i> .

#### **4.3.2 Joomla! Metamodel**

In this subsection we present the Joomla! metamodel (MMJoomla) which captures the key concepts of the Joomla! platform. Figure 4-3 presents the main elements with their attributes and relationships which compose the MMJoomla. We can find elements such as, *Template*, *Content*, *Category* or *Language*. In Table 4-5, we explain the main elements of MMJoomla, whereas in Table 4-6 presents its relationships.

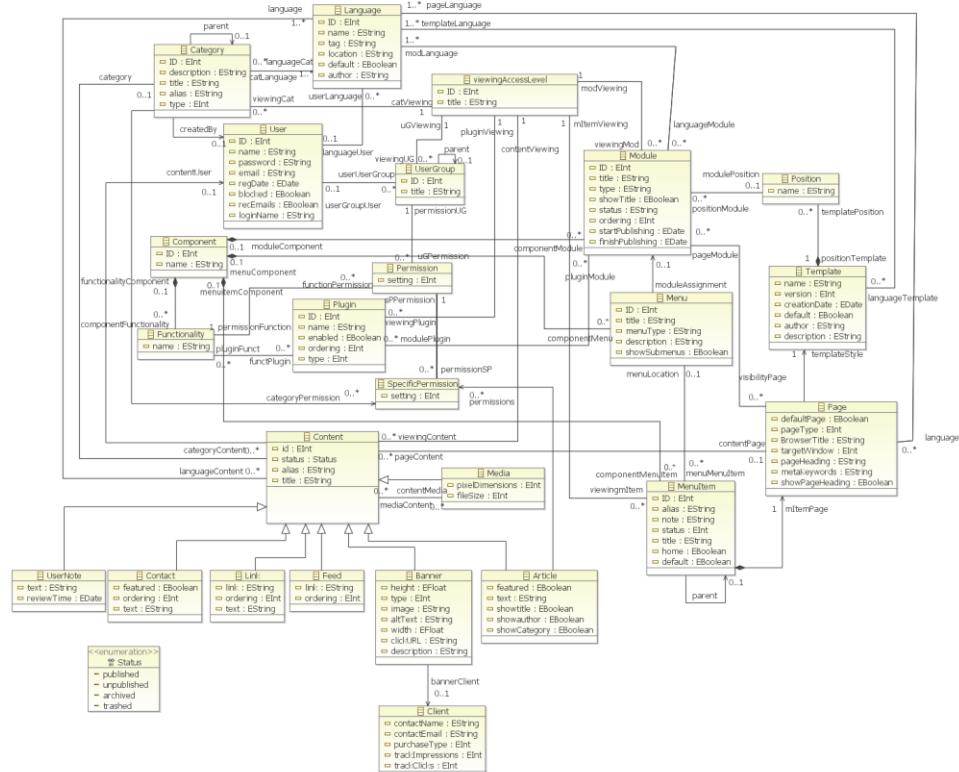


Figure 4-3. MMJoomla

**Table 4-5. Elements of the MMJoomla.**

ELEMENTS OF THE MMJOOMLA		
NAME	ATTRIBUTES	DESCRIPTION
Template	Name, version, creationDate, default, author, description	This element represents the templates which are the elements that define the presentation aspect (look and feel) of the Web application.
Position	Name	This element defines a position which is part of the <i>Template</i> . Each <i>Template</i> will define their own positions to display the information on the page of the Web application.
Page	DefaultPage, pageType, browserTitle, targetWindow, pageHeading, metaKeywords, showPageHeading	This element represents each page that compose the Web application.
Menu	Id, title, menuType, description, showSubmenus	This element represents links which can be created to facilitate the navigation between the different pages of the Web application.
MenuItem	Id, alias, note, status, title, home	This element defines the elements that compose a <i>Menu</i> . Using the menu items as mentioned above you can navigate through the pages of the Web application.
Module	Id, title, type, showtitle, status, orderingstartpublishing, finishpublishing	This element represents the functional elements which can be placed in the positions defined by the template class.
Content	Id, status, alias, title	This element represents the content that is associated with a page of the Web application. Joomla! does not allow users to create their own content types. The number of content types are added installing new extensions.
Media	PixelDimensions, fileSize	This element is a specialization of the element <i>Content</i> . It represents the media content (pictures, videos) that can be displayed within a page of the Web application.
Article	Featured, text, showtitle, Showauthor, showcategorie	This element is another specialization of the element <i>Content</i> . It represents mainly those pages containing text called articles.
Banner	Height, image, alttext, width, clickurl, description	Joomla! allows to create banners to be displayed on the Web application. We consider these banners as a specialization of the element <i>Content</i> .
Client	ContactName, ContactEmail, purchaseType, trackImpressions, trackClicks	This element represents a client who owns an ad.
Feed	Link, ordering	This element is an specialization of the element <i>Content</i> and represents different sources of current news.

ELEMENTS OF THE MMJoomla		
NAME	ATTRIBUTES	DESCRIPTION
Link	Link, ordering, text	This element is a specialization of the element <i>Content</i> and it represents an external url.
Contact	Featured, ordering, text	This element is a specialization of the element <i>Content</i> and it represents the information of a personal contact.
UserNote	Text, reviewtime	This element is a specialization of the element <i>Content</i> and it represents additional information which can be added to a user of the Web application.
Language	Id, name, tag, location, default, author	This element represents the different languages that the content can be displayed on Joomla!. Joomla provides with several language packs which can be installed.
Category	Id, description, title, alias, type	This element represents the categories to classify content in Joomla!.
User	Id, name, password, email, RegDate, blocked, recEmails, loginName	This element represents the users in Joomla! User plays an important role in Joomla! because they interact with the Web application performing different functions.
UserGroup	Id, title	This element represents the roles which can be assigned to a user. According to this role the user may or may not perform certain functions. Joomla! allow the creation of new roles. These roles must be based on existing roles.
ViewingAccessLevel	Id, title	This element represents the viewing permission of a user group. It is considered as another level of security.
Permission	Setting	This element allows to create permissions for each user. These permissions are assigned to user groups.
Component	Id, name	This element represents the packages of functionality within Joomla!. It provides modules to extend the functionality of the Web application.
Funcionalidad	Name	This element represents the functionality associated with the components and plugins of a Web application based on Joomla!.
Plugin	Id, name, enabled, ordering, type	This element represents the extensions of functionality called plugins.
SpecificPermission	Setting	This element represents those permissions which can be assigned to each article defined in the Web application.

Table 4-6 presents the main relationships established among the elements of the MMJoomla.

**Table 4-6. Relationships of MMJoomla.**

RELATIONSHIPS OF THE MMJoomla			
NAME	SOURCE	TARGET	DESCRIPTION
templateStyle	Page	Template	This permission indicates that each page can be related to a unique template.

RELATIONSHIPS OF THE MMJOOMLA			
NAME	SOURCE	TARGET	DESCRIPTION
templatePosition	Template	Position	This composition relationship structures in positions the template. We have defined an opposite relationship called <i>positionTemplate</i> .
modulePosition	Module	Position	This relationship determines that a module is located within a position of a template. In the opposite direction, we have defined another relationship called <i>positionModule</i> .
visibilityPage	Module	Page	This relationship allows to determine which modules are visible within a page. There is an opposite relationship called <i>pageModule</i> .
menuPosition	Menu	Position	This relationship indicates that a menu is located within a unique position of a template. There is an opposite relationship called <i>positionMenu</i> .
menuMenuItem	Menu	Menu Item	This relationship indicates that a menu is composed of menu items. We have defined a relationship in the opposite direction called <i>menuLocation</i> .
mItemPage	MenuItem	Page	This relationship indicates that a menu item is the link which access a page of the Web application.
contentMedia	Content	Media	This relationship determines that a content comprises media content. There is an opposite relationship called <i>mediaContent</i> .
contentPage	Content	Page	This relationship defines that a content is related to a unique page. It means that a content is displayed within a unique page.
contentCategory	Content	Category	Joomla! allows to classify the content of the Web application by using different categories. This relationship allows to determine a category for a content.
parent	Category	Category	This reflexive relationship allows to associate a category with another one to define hierarchies of categories.
categoryLanguage	Category	Language	This relationship determines that a category can be related to one or more languages.
createdBy	Category	User	This relationship associates a category with the user who created it.
languageContent	Language	Content	This relationship associates a language to a content. There is an opposite relationship called <i>contentLanguage</i> .
languagePage	Language	Page	This relationship allows to associate a language to a page. A language can be related to different pages. There is an opposite relationship called <i>pageLanguage</i> .
languageTemplate	Language	Template	This relationship determines that a language is associated with a set of templates. There is an opposite relationship called <i>templateLanguage</i> .
languageModule	Language	Module	This relationship associates a language with modules. There is an opposite relationship called <i>modLanguage</i> .

RELATIONSHIPS OF THE MMJOOMLA			
NAME	SOURCE	TARGET	DESCRIPTION
languageUser	Language	User	This relationship associates a default language with a particular user. There is an opposite relationship called <i>userLanguage</i> .
contentUser	Content	User	This relationship defines a user as the creator of a content.
userUserGroup	User	User Group	This relationship relates a user with a certain user group. As we explained previously, a user group defines the role of the user. There is an opposite relationship called <i>userGroupUser</i> .
Parent	User Group	User Group	This reflexive relationship defines that a user group can be associated with another one creating a hierarchy of user groups.
uGViewing	User Group	Viewing Access Level	This relationship associates a user group with a unique view access level. There is a relationship in the opposite direction called <i>viewingUG</i> .
uGPermission	User Group	Permission	This relationship allows to associate a set of permissions to a user group to enable them to perform certain functionalities. There is an opposite relationship called <i>PermissionUG</i> .
viewingmItem	Viewing Access Level	Menu Item	This relationship allows to relate a viewing access level to a menu item. There is an opposite relationship called <i>mItemViewing</i> .
viewingContent	Viewing Access Level	Content	This relationship relates a viewing access level to a concrete content. There is an opposite relationship called <i>contentViewing</i> .
viewingMod	Viewing Access Level	Module	This relationship relates a viewing access level to a certain module. There is an opposite relationship called <i>modViewing</i> .
funcionalityComponent	Function	Component	This relationship relates a functionality to a certain component. There is an opposite relationship called <i>componentFunctionality</i> .
componentModule	Component	Module	This composition relationship determines that a component can define a set of modules. There is an opposite relationship called <i>moduleComponent</i> .
permisionFunction	Permission	Function	This relationship determines the functions defined for a permission. There is an opposite relationship called <i>functionPermission</i> .
functPlugin	Function	Plugin	This relationship determines the set of functions performed by a plugin. There is an opposite relationship called <i>pluginFunct</i> .
modulePlugin	Module	Plugin	This relationship determines that a module can be performed along with other plugins. There is an opposite relationship called <i>pluginModule</i> .
pluginViewing	Plugin	Viewing Access Level	This relationship determines the functionality of the plugin according to a viewing access level. There is an opposite relationship called <i>viewingPlugin</i> .
Permission	Specific Permission	Article	This relationship allows the configuration of specific permission for each article.

RELATIONSHIPS OF THE MMJOOMLA			
NAME	SOURCE	TARGET	DESCRIPTION
permissionSP	Permission	Specific Permission	This relationship allows to customize a permission for a certain article as a specific permission. There is an opposite relationship called <i>sPPermission</i> .
componentMenuItem	Component	Menu Item	This composition relationship determines that a component can define by code a set of menu items. There is an opposite relationship called <i>menuItemComponent</i> .
componentMenu	Component	Menu	This composition relationship determines that a component can define by code a set of menus. There is an opposite relationship called <i>menuComponent</i> .

#### 4.3.3 Drupal Metamodel

Finally, in this subsection, we present the Drupal metamodel (MMDrupal) which captures the key concepts of the Drupal platform. Figure 4-4 the elements and the relationships that compose this MMDrupal. Some of the main elements represent concepts such as, *User*, *Menu*, *MenuItem* or *Block*. After presenting graphically the MMDrupal, we explain their elements and relationships by using two tables. Table 4-7 presents the main elements gathered in MMDrupal and Table 4-8 shows its relationships.

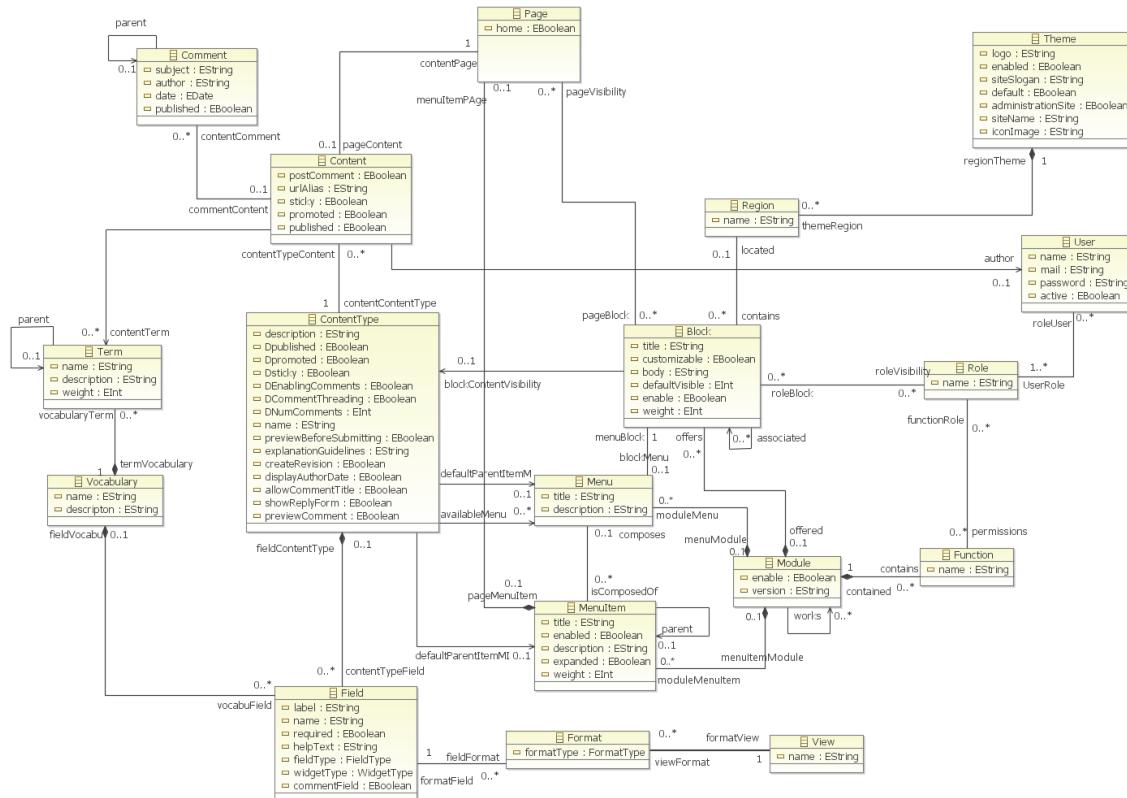


Figure 4-4. MMDrupal

**Table 4-7. Elements of the MMDrupal.**

ELEMENTS OF THE MMDRUPAL		
NAME	ATTRIBUTES	DESCRIPTION
Theme	Logo, enabled, siteSlogan, default, administrationSite, siteName, iconImage	<i>Theme</i> element represents the presentation aspect (structure, look-and-feel and style) of the pages of the Web application.
Region	Name	<i>Region</i> element represents the different areas that defines the structure of a Web page. The regions can locate different elements that are displayed on the Web page. The regions are defined by the theme.
Page	Home, urlAlias	A <i>Page</i> represents the navigational nodes which compose the Web application. <i>Page</i> element has associated the following attributes.
Block	Title, customizable, body, defaultVisible, enable, weight	This element represents component of functionality that can be placed in different regions (Element <i>Region</i> ) defined by the element <i>Theme</i> (Element <i>Theme</i> ).
Content	PostComment, urlAlias, sticky, promoted, published	This element represents the content managed by the Web application. Each page that composes the Web application will have an associated content that belongs to a specific type.
ContentType	Description, Dpublished, Dpromoted, Dsticky, DEnablingComments, DCommentThreading, DNumComments, Name, previewBeforeSubmitting, explanationGuidelines, createRevision, displayAuthorDate, allowCommentTitle, showReplyForm, previewComment	<i>ContentType</i> element represents the type of a content. Drupal allows the definition of specific types of content. These types can be created by the administrator of the Web application. Each content type has a name and can be customized with different fields.
Field	Label, Name, Required, helpText, fieldType, widgetType, commentField	The <i>Field</i> element represents the fields that can be defined in Drupal. These fields can be added to content types and vocabularies.
View	Name	This element defines a presentation to display the contents of a field.
Format	FormatType	This element defines the format that a field will be displayed. This element associates a field with a view.
Term	Name, description, weight	This element represents the words or phrases that can be associated to a content type. These terms allow users to search easily content in the Web application. Also it is worth noting that you can create hierarchical relationships between various terms.
Vocabulary	Name, description	This element represents vocabularies which are the elements that classify the different terms defined by <i>Term</i> class. This element has the following attributes.

ELEMENTS OF THE MMDRUPAL		
NAME	ATTRIBUTES	DESCRIPTION
Menu	Title, description	This element represents the menus of Drupal. Drupal has a high configurable menu system since it allows to define new menus and edit them.
MenuItem	Title, Enable, Description, Expanded, Weight	This element represents the items that compose the menus defined in Drupal. As mentioned above, Drupal has a thorough menu system. Each menu consists of menu items that are ordered hierarchically. A menu item contains a link that provides access to the pages of the Web application.
User	Name, mail, password, active	This element represents the users in Drupal who interact with the Web application managing the different content.
Role	Name	This element represents the role that can be assigned to the users registered to the Web application.
Function	Name	This element represents the different actions to be performed by the users. These functions are related to the roles. Depending on this role the user can interact with the Web applications in a different way.
Module	Enable, version	This element represents the packages of functionality that compose the Web application implemented by Drupal. A <i>Module</i> can add new users, define new menus as well as to extend the functions offered by the Web application.
Comment	Subject, author, content, date, published	This element represents the comments that the users can add to the content of our Web application. It is possible to create threads to keep track of all comments and responses.

Table 4-8 presents the main relationships established among the elements of the MMDrupal.

**Table 4-8. Relationships of the MMDrupal.**

RELATIONSHIPS OF THE MMDRUPAL			
NAME	SOURCE	TARGET	DESCRIPTION
themeRegion	Theme	Region	This composition relationship means that a theme is composed of different regions. There is a relationship called <i>regionTheme</i> in the opposite direction.
located	Block	Region	This relationship represents that a block can be located in one region within a page of the Web application. There is a relationship in the opposite direction called <i>contains</i> .
pageVisibility	Block	Page	This relationship indicates which blocks are visible on a page. There is a relationship in the opposite direction called <i>pageBlock</i> .

RELATIONSHIPS OF THE MMDRUPAL			
NAME	SOURCE	TARGET	DESCRIPTION
contentPage	Content	Page	This relationship represents that a content is displayed by means of a page. There is a relationship in the opposite direction called <i>pageContent</i> .
contentContentType	Content	Content Type	This relationship indicates that a content can be associated with one of the content types. There is a relationship in the other direction called <i>contentTypeContent</i> .
fieldFormat	Field	Format	This relationship means that the fields have just one format associated. In the opposite direction, there is a relationship called <i>formatField</i> .
formatView	Format	View	This relationship determines that a format is related to a unique view. There is an opposite relationship called <i>viewFormat</i> .
vocabularyTerm	Vocabulary	Term	This composition relationship indicates that a vocabulary is composed of a set of terms. In the opposite direction, we have defined the relationship <i>termVocabulary</i> .
parent	Term	Term	This reflexive relationship that relates hierarchically a term of the element <i>Term</i> to another term.
fieldVocabulary	Field	Vocabulary	This relationship allows to define that a field can belong to a certain vocabulary. On the contrary a relationship called <i>vocabularyField</i> has been defined.
composes	Menu Item	Menu	This relationship lets associate a menu item to a menu. There is a relationship in the opposite direction called <i>isComposedOf</i> used to represent that a menu is composed of a set of menu items.
parent	Menu Item	Menu Item	It is a reflexive relationship which allows the definition of hierarchies between the menu items.
defaultParentItemM	Content Type	Menu	This relationship allows a menu to a content type. Also, there is a similar relationship called <i>defaultParentItemMI</i> between the elements <i>ContentType</i> and <i>MenuItem</i> .
availableMenu	Content Type	Menu	This relationship allows menus to be available for one type of content.
menuItemPage	Menu Item	Page	This composition relationship defines that a <i>MenuItem</i> links to a page. There is an opposite relationship called <i>pageMenuItem</i> representing that a page can be accessed by different menu items.
autor	Content	User	This relationship determines the creator of a content.. Also, it is possible to determine anonymous content, in this case any user is related to the content.
roleUser	Role	User	This relationship allows to associate a role with a user. There is an opposite relationship called <i>userRole</i> .
functionRole	Function	Role	This relationship defines the functions that a role can perform. There is a relationship in the opposite direction called <i>permissions</i> .

RELATIONSHIPS OF THE MMDRUPAL			
NAME	SOURCE	TARGET	DESCRIPTION
contains	Module	Function	This relationship allows to define a module as a package of functions. On the other hand, there is an opposite relationship called <b>contained</b> .
works	Module	Module	This reflexive relationship allows a set of modules to be associated to perform a specific function.
roleVisibility	Role	Block	It is a relationship allows to define which blocks are visible for certain roles. There is a relationship in the opposite direction called <b>roleBlock</b> .
parent	Comment	Comment	It is a reflexive relationship which allows the definition of nested comments creating hierarchies of comments.
offers	Module	Block	It is a composition relationship that determines that a module can implement a set of blocks by using code (e.g. PHP code). There is an opposite relationship called <b>offered</b> .
contentComment	Content	Comment	This relationship can relate a content to their comments. There is an opposite relationship called <b>commentContent</b> .
ModuleMenu	Module	Menu	It is a composition relationship that determines that a module can define a set of menu by using code (e.g. PHP code). There is an opposite relationship called <b>menuModule</b> .
ModuleMenuItem	Module	Menu Item	It is a composition relationship that determines that a module can implement a set of menu items by using code (e.g. PHP code). There is an opposite relationship called <b>menuItemModule</b> .

#### 4.4 Metamodel Modularization

In accordance with the complexity of the CMS metamodels, we modularized them by using a set of **concerns**. According to Dijkstra, *concerns* are a technique for effective ordering of one's thoughts as well as for focusing one's attention upon some aspect (Dijkstra, 1976) and by Parnas, *concerns* are a concept necessary for the modularization as a way towards addressing software development complexity (Parnas, 1972). Therefore, we have used concerns to organize the different elements of the CMS metamodels according to their semantic to reduce the complexity of their complexity.

This modularization based on *concerns* can be found in many disciplines, such as Aspect-Oriented Software Development, Software Engineering, Software Architectures or Web Engineering. For instance, in Software architecture *concerns* are represented by architectural views which present different aspects of the software architecture allowing to design it more precisely (Clements et al., 2003),

whereas in Web Engineering, *concerns* are involved in the design and development of Web applications providing different views on them. According to (Kappel, Pröll, et al., 2006) the main *concerns* in Web Engineering are: data, navigation, presentation and behaviour concern.

To define the five concerns to modularize the CMS Common Metamodel, we have based on the *concerns* defined in Web Engineering because of the clear Web nature of the CMS-based Web applications. The five *concerns* used are explained below:

- **Navigation concern** captures the elements that define the navigational structure of the CMS-based Web application.
- **Presentation concern** captures the elements to define the internal structure of each page and its look and feel.
- **Behaviour concern** represents the elements which allow us to define the different functions performed by the CMS-based Web application.
- **Content concern** captures the elements related to the contents and content types managed by the CMS-based web application.
- **User concern** represents the elements which allow us to define the users of the CMS-based Web application, their roles and their permissions.

These concerns are applied on the specification of the elements composing of the CMS Common Metamodel in the following section.

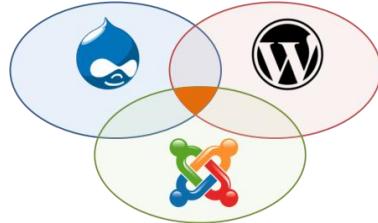
## 4.5 CMS Common Metamodel Specification

In this section, we present the specification of the CMS Common Metamodel (MMCMSCommon) taking account the elements captured in the CMS metamodels defined in the previous sections as well as the set of concerns to organize the selected elements composing the MMCMSCommon.

To select the elements of the MMCMSCommon, we considered three different strategies: 1) the **intersection** of the three CMS metamodels, 2) the **intersection** and the **intersecting pairs** of the three CMS metamodels or finally, 3) the **union** of the CMS metamodels.

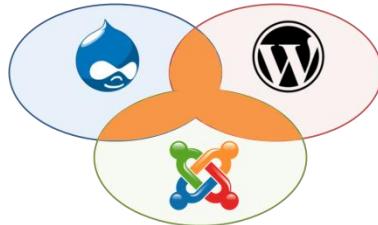
The **intersection** allowed us to obtain most of the most important common elements such as *Page*, *Menu*, *MenuItem*, *Content*, *User* or *Role*, to model CMS-

based Web applications, but not all of them. Some element such as *Field*, *Term*, *Comment* or *Category* were excluded. Figure 4-5 shows the scope of this strategy.



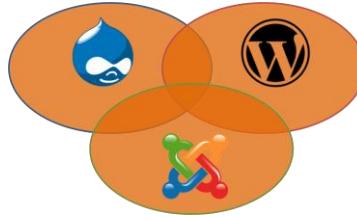
**Figure 4-5. Intersection strategy.**

The **intersection** and the **intersecting pairs** allowed us to increase the set of required elements to model a CMS-based Web application including elements such as *Field*, *Term*, *Comment* or *Category*, but still not all the required elements were included. Figure 4-6 shows the scope of this second strategy.



**Figure 4-6. Intersection and intersecting pairs strategy.**

Finally, the **union** achieved the inclusion of all the necessary elements to model a CMS-based Web application. Figure 4-7 shows its scope.



**Figure 4-7. Union strategy.**

Therefore, we decided to make the union of the three CMS metamodels to define the **MMCMSCommon**. Table 4-9 shows the relation of elements considered within this union and classified considering the five concerns defined in Section 4.4.

**Table 4-9.** Union of the elements conforming the MMCMSCommon.

	<i>MMCMS COMMON</i>	<i>MMWORDPRESS</i>	<i>MMJoomla</i>	<i>MMDRUPAL</i>
<i>NAVIGATION CONCERN</i>	Page	✓	✓	✓
	Menu	✓	✓	✓
	MenuItem	✓	✓	✓
<i>PRESERATION CONCERN</i>	Theme	✓	✓ (Template)	✓
	Region	✓	✓	✓
<i>BEHAVIOUR CONCERN</i>	Function	✓ (Task)	✓	✓
	Block	✓ (Widget)	✓ (Module)	✓
	Module	✓ (Plugin)	✓ (Component)	✓
<i>CONTENT CONCERN</i>	Content	✓	✓	✓
	Term	✓ (Tag)	none	✓
	Vocabulary	none	none	✓
	Comment	✓	none	✓
	Language	none	✓	none
<i>USER CONCERN</i>	Category	✓	✓	none
	Role	✓	✓	✓
	RoleGroup	none	✓	none
	Permission	✓ (Capability)	✓	✓
	User	✓	✓	✓

The same element can take different names in the three CMS metamodels. As it is shown in Table 4-9, the different names are presented in brackets. For example, for the *Theme* element, we can find the name of *Template* in MMJoomla; the *Function* is also called *Task* in MMWordpress; the *Block* is called *Widget* in MMWordpress and *Module* in MMJoomla; the *Term* is called *Tag* in MMWordpress, the *Module* receives the name of *Plugin* in MMWordpress and *Component* in MMJoomla and finally, *Permission* is also called *Capability* in MMWordpress. Our task was to standardize the name selecting the most intuitive one.

It is worth noting that after considering the union, we decided to polish the MMCMSCommon discarding some elements which we considered irrelevant as well as redefining other elements to a better definition. On the one hand, we discarded the element *ViewingAccessLevel* from MMJoomla for being too specific of the Joomla! domain as well as the elements *ContentType*, *Format* or *View* from the MMDrupal were discarded for being also too specific. On the other hand, the *Content* element has been redefined by considering the different types of content defined in the three CMS metamodels. Therefore, we have specialized the *Content* in: *Text*, *Media*, *List* and *MediaGallery*.

Regarding the relationships, we have included just the relevant relationships in MMCMSCCommon, the other ones have been discarded. For instance, the relationship between the *Language* element and the *User* element from MMJoomla has been discarded since we are not interested in knowing the language of a user. Clearly, the MMCMSCCommon evolves and we do not discard the possibility of including new elements and relationships in the future.

In the following subsections we explain in detail the elements and relationships that appear in the MMCMSCCommon. The explanations of these concepts are structured considering the five concerns. Figure 4-8 presents the MMCMSCCommon.

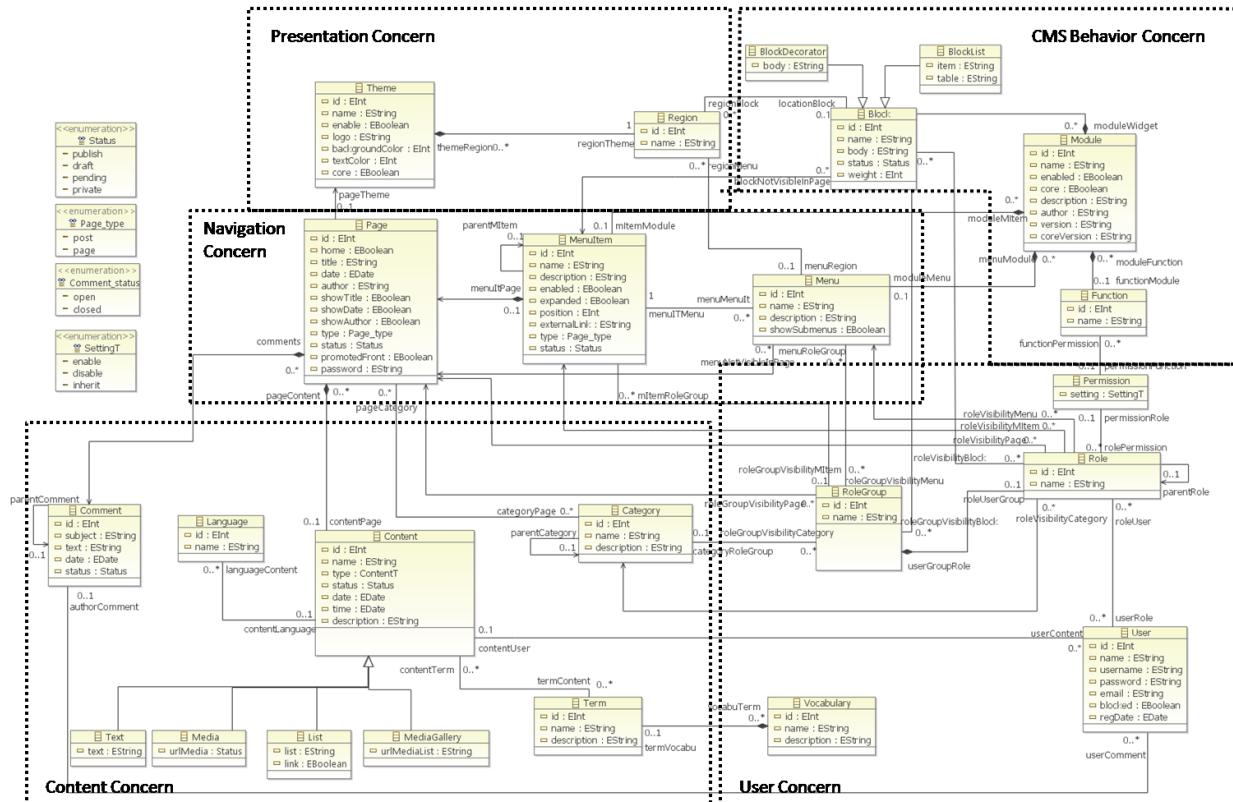


Figure 4-8. MMCMSCCommon.

#### 4.5.1 Navigation Concern

As we explained in Section 4.4, this concern represents the required elements to define the navigational structure of the CMS-based Web application. Figure 4-9 shows the elements and relationships which represent this navigation concern. In the following sections, we explain the meaning of each element, we present their attributes and their relationships between other elements within the metamodel.

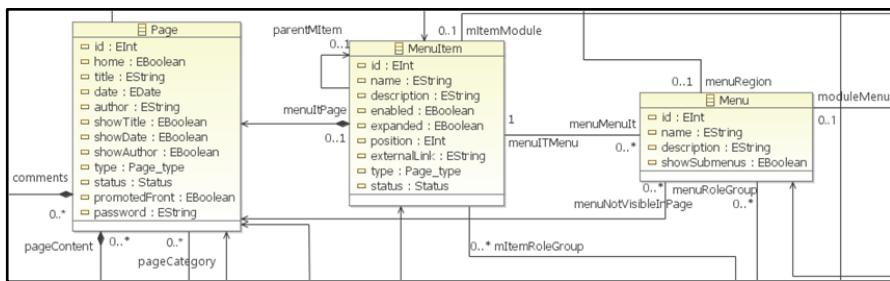


Figure 4-9. Navigation concern in the MMCMSCommon.

#### Page

One of the main elements is *Page* which refers to each navigational node of the CMS-based Web application. This element establishes the navigational structure of the Web application. The *Page* element counts with the following set of attributes.

- **Id:** it is an attribute which determines the unique identifier of a page.
- **Home:** it is an attribute that defines whether the page is considered the home of the CMS-based Web application.
- **Title:** it is an attribute which specifies the name of the page and it is usually displayed on the top of the page.
- **Showtitle:** it is an attribute which specifies whether the title of the page must be displayed or not on the top of the page.
- **Date:** this attribute defines the date in which the pages was created.
- **showDate:** it is an attribute which specifies whether the date of creation of the page must be displayed or not.

- **Author:** this attribute defines the name of the creator of the page. It is worth noting that the author of the page can be different to the author of the content which is displayed within the page.
- **showAuthor:** this attribute specifies whether the author of the page must be displayed or not.
- **Type:** after analyzing the different types of pages in the previous CMS platforms we realized that there are mainly two types of pages. Those pages that display *static* content and those that display *dynamic* content. This attribute defines the type of the page.
- **Status:** after reviewing the states of the pages in the three previous CMS platforms we conclude that a page can be found basically in four different states. The first state is the *publish* state; this state represents that the page can be accessed by all the users with no constraints. Another state is the *draft* state; this state indicates that the page is not found in its last version so that it cannot be still accessed. The third state is the *pending* state; this state indicates that the page is finished but its content is not still approved by the administrator of the Web application. The last state is the *private* state; this state indicates that the page is published but just it can be accessed by using a password.
- **Password:** this string attribute specifies the required password in case that the status of the page is defined as private state.
- **Promoted:** this attribute specifies whether the page is showed in the home page.

The main relationships established from the *Page* element to other elements are explained in the following.

- **PageTheme:** this relationship is defined between the *Page* element and the *Theme* element. This relationship specifies the look and feel of the *Page*. Depending on the CMS platform, it is possible to define a *Theme* for each *Page* of the Web application, e.g. Drupal. Otherwise, other CMS platforms allow to define just one *Theme* which determines the look and feel of all the pages of the Web application. To be more flexible and to cope with all the possibilities, we have established the cardinality of this relationship as a zero-or-one cardinality, allowing to relate one *Page* just to one *Theme*.

- **PageContent:** this relationship is established between the *Page* element and the *Content* element. It indicates the content which is displayed in that page. The cardinality of this relationship is defined as a to-many cardinality since a page can display a set of different content.
- **Comments:** this relationship relates the *Page* element to the *Comment* element. It defines the comments of one page. Its cardinality is to-many since a page can have a set of comments related.
- **PageCategory:** it is the relationship between the *Page* element and the *Category* element. A *Category* can be considered as a container which classify the *Pages* and also ease the management of the Web content. This relationship has a to-many cardinality so that it allows to relate a page to different categories since these categories are not strict divisions.

## Menu

Another important element in this concern is the *Menu* element. This element allows the navigation between the pages composing the navigational structure of the CMS-based Web application. The required attributes to define a *Menu* are specified below:

- **Id:** it is an attribute which determines the unique identifier of the menu.
- **Name:** it is an attribute that defines the name of the menu.
- **ShowMenuItems:** it is an attribute that defines whether a menu can be displayed along with its menu items.

The main relationships established from the *Menu* element to other elements are explained in the following.

- **MenuRegion:** it is a relationship defined between the *Menu* element and the *Region* element. This relationship determines the location of the menu element within the page. According to the cardinality of this relationship a *Menu* just can be located within one region.
- **NotVisibleinPage:** it is a relationship between the *Menu* element and the *Page* element. When menus are located in a region are displayed within all the pages by default. By using this relationship it is possible to constraint the visibility of a menu for one page. The cardinality of this relationship is a to-many cardinality which specifies that a menu can be not visible for a set of pages.

- **MenuMenuItem:** it is a relationship defined between the *Menu* element and the *MenuItem* element. This relationship allows to relate a *Menu* to the *MenuItems* that compose it. The cardinality of this relationship is a to-many cardinality allowing to associate a menu with a group of different menu items.

### Menu Item

The last important element of this concern is the *MenuItem* element which represents the links that compose a menu element and allow the access to the different pages of the CMS-based Web application. This element is defined by the following attributes:

- **Id:** it is an attribute which determines the unique identifier of *MenuItem*.
- **Name:** it is an attribute that defines the name of the *MenuItem* element which will be displayed in the menu.
- **Position:** the different menu items are displayed in a certain order within the menu. This attribute determines the position of each *MenuItem* in the menu.
- **Expanded:** this attribute makes visible the hierarchy of menu items which can be established.
- **ExternalLink:** usually the menu items allow the access to pages of the Web application. Sometimes these menu items can link to external pages. This attribute defines the url of these external pages.
- **Description:** it is an attribute that stores a description about the information to which the *MenuItem* allows us to access.
- **Enabled:** it is an attribute that defines whether the *MenuItem* is visible within the menu.
- **Type:** this attribute defines whether the page to which the *MenuItem* allows the access contains static or dynamic content.
- **Status:** this attribute defines the status (publish, private, draft and pending).

There are a set of relationships defined from the *MenuItem* element. In the following, we present them.

- **MenuItemPage:** it is a relationship established between the *MenuItem* element and the *Page* element. It is a composition relationship which determines the page which the menu item links to. A *Page* is related to just

one *MenuItem*. It does not make sense to create a page non-related to a menu item. Just the external pages can be defined by using the attribute *externalLink* which allows to write its url.

- **ParentMItem:** it is a reflexive relationship called which allows to define hierarchies between the *MenuItem*. It is a relationship with a zero-or-one cardinality so that a *MenuItem* can be related to another *MenuItem* which it is its parent.

#### 4.5.2 Presentation Concern

This concern represents the required elements to define the structure and look and feel of the Web pages of a CMS-based Web application. Figure 4-10 presents the two elements considered within the presentation concern.

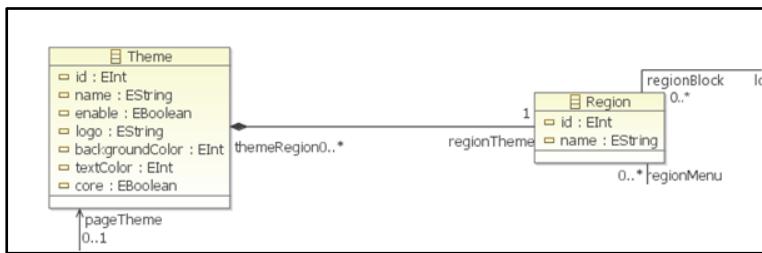


Figure 4-10. Presentation concern in MMCMSCCommon.

##### Theme

The three CMS metamodels define an element that determines the structure of the pages of the Web application and their look and feel. This element is represented by the *Theme* element within the MMCMSCCommon. This element defines a set of attributes that allow the definition of the look and feel of the pages of the Web application. These attributes are explained below.

- **Id:** it is an attribute which determines the unique identifier of the *Theme*.
- **Name:** it is an attribute that defines the name of the *Theme*.
- **Enable:** it is an attribute that allows to activate or deactivate the *Theme*. We can define many *Themes* for a Web application but not all of them determine the look and feel of the pages of the Web application. With this attribute it is possible to specify which of those *Themes* are defining the presentation aspect of the pages.

- **Logo:** it is an attribute which keeps the path where the image that represent the main logo of the Web application is stored.
- **BackgroundColour:** it is an attribute that defines the RGB code to determine the colour of the background of the page.
- **textColour:** it is an attribute that defines the RGB code to determine the colour of the text displayed within the page.

The main relationship defined from the *Theme* element called *themeRegion* is explained below.

- **themeRegion:** this relationship is defined between the *Theme* element and the *Region* element. It specifies the regions to structure the pages of the Web application. The cardinality of this relationship is a to-many relationship so that it allows to define a set of different regions.

### **Region**

As we mentioned, the different regions that determine the internal structure of the page are defined with the *Region* element. These regions allow to organize the information displayed on the page. For example, they can distribute *Menu* elements and *Block* elements. Mainly the two attributes which define this element are two:

- **Id:** it is an attribute which determines the unique identifier of the *Region*.
- **Name:** it is an attribute that defines the name of the *Region*.

Two relationships are defined related to the *Region* element that we explain in the following.

- **RegionBlock:** it is a relationship between the *Region* element and the *Block* element to define the block elements which are located and displayed within a certain region. The cardinality of this relationship is to-many because within a region can be located many *Blocks*.
- **RegionMenu:** it is a relationship between the *Region* element and the *Menu* element. As the previous relationship, this relationship allows to define which menus are located within a concrete region and also it has a to-many cardinality.

### 4.5.3 Behaviour Concern

This concern considers the elements that define the functionality of the CMS-based Web application. It is worth noting, that one of the main features of this kind of Web applications is the modularity of its functionality which makes these Web applications more extensible, scalable, and they can be adapted to the necessities of users. Figure 4-11 shows the elements gathered within this concern.

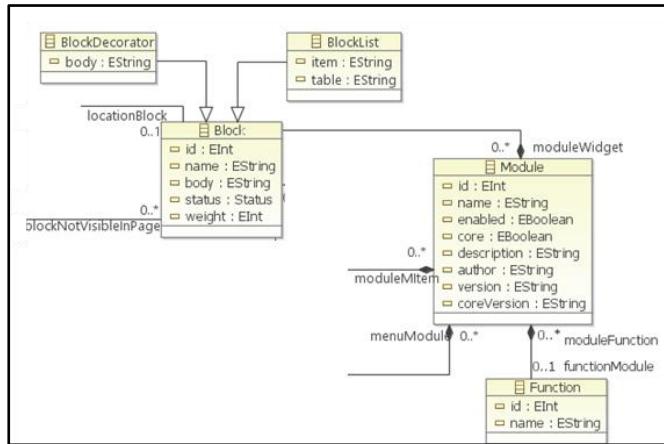


Figure 4-11. Behaviour concern in MMCMSCommon.

### Module

The main concept of this concern is represented by the *Module* element. This element represents the packages implemented by source code which define the different functionalities to be added to the CMS-based Web application. Besides functionalities, a *Module* can define *Blocks*, *Menu* and *MenuItems*. The attributes defined for this element are explained below:

- **Id**: it is an attribute which determines the unique identifier of the *Module*.
- **Name**: it is an attribute that defines the name of the *Module*.
- **Enabled**: it is an attribute that allows to activate or deactivate the module. We can define many modules for a Web application but not all of them are activated providing its functionalities. With this attribute it is possible to specify which of those modules are taking part in the functionality of the CMS-based Web application.
- **Core**: *Modules* can be classified into two groups, those ones which provide basic functionality and are installed by default by the CMS platform and

those modules which provide extended functionality which are added later to increase the capacities of the CMS-based Web application. The former are called the core of the CMS-based Web application, e.g. add users, add comments; and the latter are considered the extensions. This attribute allows us to know whether the module is considered part of the core or, otherwise is considered one of the extensions.

- **Description:** this attribute keeps an excerpt about the main functionalities provided by the the *Module*.
- **Author:** it represents the name of the person which has created the source code that implements the *Module*.
- **Version:** a *Module* can evolve by restructuring the code or adding new functionalities. This string attribute allows to control the different versions of a module.
- **CoreVersion:** a CMS platform is constantly evolving generating different versions. The main differences are reflected in the core of the CMS platform. This attribute allows to determine the version of the core which we are working with.

The four main relationships specified with the *Module* element are defined between the *Block*, the *Menu*, the *MenuItem* and the *Function* elements. These relationships are explained below.

- **Moduleblock, moduleMenuItem, moduleMenu and moduleFunction:** they are relationships with similar meaning. They represent that the code written within a *Module* can define a *Block*, a *Menu*, a *MenuItem* or another *Function*. The cardinality of these relationships are defined as to-many relationships.

## Block

Another main element is the *Block* element. This element represents those components offering a functionality such as a *clock* or a *calendar*, which can be placed and displayed in some part of the Web page. We can find a large number of different *Blocks*, each one with concrete features. We think it could be possible to categorize these *Blocks* in different types. Within the MMCMSCCommon we have tried to establish a first classification by defining two elements which are specializations of the main element *Block*. The first specializations is defined by the element *BlockDecorator* and it represents those blocks that offer

functionalities such as a *clock*, *twitter access*, a *calendar* or a *thermometer*. The other specialization is specified by the element *BlockList* which represents those ones which display a certain list, such as a list of connected users or a list of last posts.

The attributes specified for the *Block* element are explained below.

- **Id:** it is an integer attribute which determines the unique identifier of the *Block*.
- **Name:** it is an attribute that defines the name of the *Block*.
- **Status:** this attribute defines whether a *Block* is visible and displayed within the page. It is possible to locate a *Block* within a region of the page but it can remain invisible since it is not enabled.
- **Weight:** the set of blocks in a region follows an order. This order is specified by this attribute.

The two specializations, *BlockDecorator* and *BlockList* elements, inherits the attributes defined by the *Block* element. Apart from that, each element defines its own attribute, such as the *BlockDecorator* element which specifies the following attribute.

- **Body:** this attribute keeps the source code that implement the *Block*. It is easy to import the code of these *Blocks* from Web pages in Internet and install the blocks in the CMS-based Web application.

On the other hand, the *ListDecorator* element defines the attributes presented below:

- **Table:** the elements listed are usually stored in the tables of the data base of the Web application. This attribute determines the table from where the listed information is extracted.
- **Item:** this attribute defines the element or value which is displayed in the list.

According to the relationships defined for the *Block* element, we can say that mainly we have defined three relationships explained below.

- **LocationBlock:** The first one is established between *Block* element and the *Region* element which specifies the region in which the *Block* is located. Its cardinality is zero-or-one since if a block is located just can be in one region.

- **BlockModule:** this relationship is defined between the *Module* element and the *Block* element. As we said previously, the code implementing a *Block* is written and defined within a concrete module.
- **BlockNotVisibleInPage:** this relationship is specified between the *Block* element and the *Page* element. It determines in which pages the *Block* will be not visible. By default, a *Block* placed in a region is visible within all the pages. With this relationship it is possible to constrain the visibility of a *Block*.

### Function

The other element of the behaviour concern is the *Function* element. This element represents all the actions that a user can perform by means of the CMS-based Web application, such as, to comment a content, to edit a page or add change their data from his user's profile. These *Functions* are implemented by code which is defined within the different *Modules*. The main attributes of this element are explained below:

- **Id:** it is an attribute which determines the unique identifier of the *Function*.
- **Name:** it is a string attribute that defines the name of the *Function*.

In the following, we present the relationships defined from the element *Function*.

- **FunctionModule:** a relationship related to the *Module* element since the code that implements a function needs to be defined within a *Module*.
- **functionPermission:** it is a relationship defined between the *Function* element and the *Permission* element. The availability of each *Function* is set for each *User* or *Role*.

#### 4.5.4 Content Concern

*Content* is one of the main concepts in a CMS-based Web application. This concern gathers all the concepts related to the data which is managed by the CMS-based Web application. Apart from the data, elements used for the classification of the data by means of categories and categorization by using tags are represented in this concern. Figure 4-12 shows the elements considered within this concern.

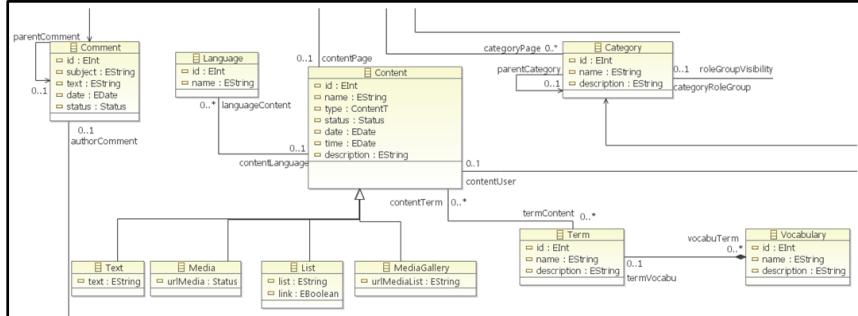


Figure 4-12. Content concern in MMCMSCommon.

## Content

The content managed and displayed by the CMS-based Web applications is represented by the element *Content*. We have made a specialization of this element depending on the type of this *Content*, such as *Text*, *Media*, *List* and *MediaGallery*. The attributes defined for the *Content* element are explained below.

- **Id:** it is an attribute which determines the unique identifier of the *Content*.
- **Name:** it is an attribute that defines the title that identify the *Content*.
- **Date:** this attribute indicates the date when the content was created.
- **Time:** this attribute indicates the time when the content was created.

In the following, we explain the attributes defined for each specialization of the *Content* element.

The *Text* element represents the text that can be added and displayed within a Web page. Its main attribute is a string attribute called *text* which keeps the text data.

The *Media* element represents all the multimedia content which can be attached within a Web page such as pictures, audio or videos. This element has an attribute called *urlMedia* which defines the path where the multimedia content is stored into the CMS-based Web application project.

The *List* element represents a list of items. There are two attributes defined for this element; the attribute *list* which represents the items of the list to be displayed and the attribute *link* which indicates that the items of the list are not just text but links that allow to navigate to other Web pages.

Finally, the *MediaGallery* element represents a set of multimedia elements displayed as a gallery. Its main attribute is an attribute called *urlMediaList* which keeps the paths where the multimedia elements are stored.

As for the relationships defined for the *Content* element, it is possible to distinguish those ones which relate the *Content* element to the *Language* element and the *Term* element.

- **ContentLanguage:** the relationship defined with the *Language* element specifies the language in which the content has been written. Although the relationship is defined at *Content* element, it mainly refers to the *Text* element and *List* element. It is a relationship with a zero-or-one cardinality since a content just can be related to one language.
- **ContentTerm:** it is the relationship defined between the *Content* element and the *Term* element. It is used to associate the content with terms. According to its to-many cardinality it is possible to associate a set of terms to the same content. This association allows to tag content so that it is possible to categorize it and facilitate the management of the information within the CMS-based Web application.

## Language

The element *Language* of this concern represents the languages in which the text content is written. It is worth noting, that no relationships have been defined from the *Language* element. The attributes of this element are:

- **Id:** it is an attribute which determines the unique identifier of the *Language*.
- **Name:** it is an attribute that defines the title that indentify the *Language*.

## Comment

Another relevant element is the *Comment* element. This element represents all the comments and reviews created by users. These *Comments* are associated to the Web page which displays the Web content. The attributes that define this element are explained below:

- **Id:** it is defined an attribute which determines the unique identifier of the *Comment*.
- **Subject:** it is an attribute that defines the title that identifies the *Comment*.
- **Text:** this attribute keeps the text that represents the body of the *Comment*.

- **Date:** this attribute specifies the date when the *Comment* was written.
- **Status:** this attribute defines the status of the *Comment*. The status of a *Comment* basically can present four different states. The first state is the *publish* state; this state represents that the comment is published and visible for all the users. Another state is the *draft* state; this state indicates that the comment is not found in its last version so that it cannot be published and visible. The third state is the *pending* state; this state indicates that the comment is finished but it is not approved by the administrator of the CMS-based Web application. The last state is the *private* state; this state indicates that the comment is published but just visible for its author.

In the following, we explain the two main relationships defined from the *Comment* element.

- **ParentComment:** it is a reflexive relationship which allows to relate a *Comment* to its parent so that it is possible to define a hierarchy of comments. The cardinality of this relationship is zero-or-one cardinality since a comment just can be related to one parent comment.
- **AuthorComment:** it is a relationship between the *Comment* element and the *User* element to specify the author of the *Comment*. Likewise the *parentComment* relationship, its cardinality is zero-or-one since a comment just can belong to one author.

### Term

The *Term* element represents the tags which can be attached to a content and works as the required metadata to categorize it. The attributes that specify this element are:

- **Id:** it is an attribute which determines the unique identifier of the *Term*.
- **Name:** it is an attribute that defines the name that identifies the *Term*.
- **Description:** it is an excerpt that helps to describe the meaning of the *Term*.

The *Term* element is related basically to two relationships which we explain in the following.

- **TermContent:** this relationship associates the *Term* element with the *Content* element. This relationship has a to-many cardinality since one *Term* can be attached to different content.

- **TermVocabu:** it is a relationship which relates the *Term* element with the *Vocabulary* element. It allows to classify the terms in different groups depending on their semantics. It has a zero-or-one cardinality since a *Term* just can belong to one *Vocabulary*.

### **Vocabulary**

A *Vocabulary* element represents the different groups in which terms can be classified according their semantics. The attributes of this element are:

- **Id:** it is an integer attribute which determines the unique identifier of the *Vocabulary*.
- **Name:** it is an attribute that defines the name that indentifies the *Vocabulary*.
- **Description:** it is an excerpt that helps to describe the meaning of the *Vocabulary*.

The relationships defined from the *Vocabulary* element are presented in the following.

- **VocabuTerm:** this relationship associates the *Vocabulary* element with the *Term* element. This composition relationship has a cardinality of to-many indicating that a *Vocabulary* is composed of a set of terms.

### **Category**

The last element of this concern is the *Category* element. This element represents the categories which allow to classify de pages of a CMS-based Web application. It is worth noting, that no relationships have been defined from the *Category* element. This element is composed of the following attributes:

- **Id:** it is an attribute which determines the unique identifier of the *Category*.
- **Name:** it is an attribute that defines the name that indentify the *Category*.
- **Description:** it is an excerpt that helps to describe the meaning of the *Category*.

#### **4.5.5 User Concern**

The user concern refers to all those concepts related to the users of the CMS-based Web application such as the roles and their permissions. Figure 4-13 shows the elements which represent the user concern.

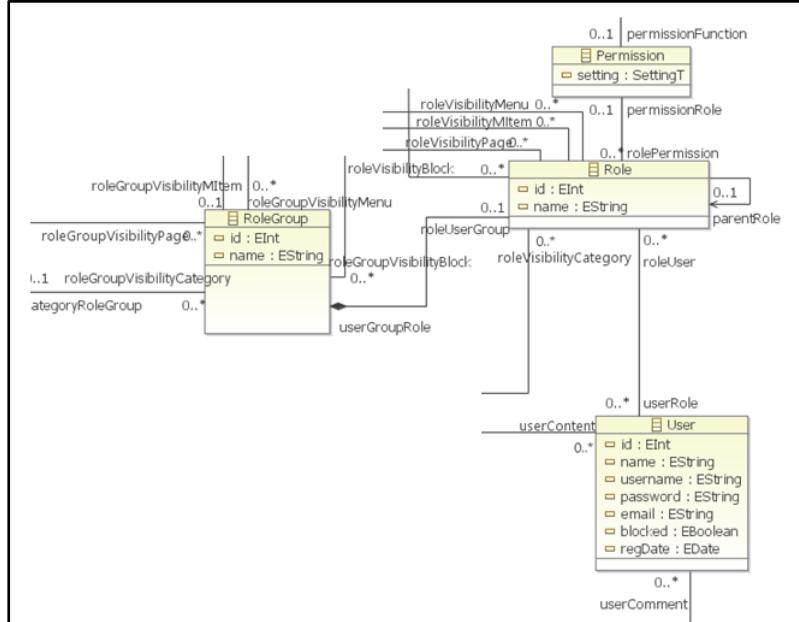


Figure 4-13. User concern in MMCMSCommon.

## User

The *User* element represents the profile data of each user within the CMS-based Web application. The attributes defining this element are explained below:

- **Id**: it is an attribute which determines the unique identifier of the *User*.
- **Name**: it is an attribute that defines the full name of the *User*.
- **Username**: it is an attribute that defines the name with which the user can access the CMS-based Web application.
- **Password**: it is an attribute that defines the password with which the user can access the CMS-based Web application.
- **Email**: it is an attribute that defines the user's email to contact him.
- **Blocked**: it is an attribute indicating whether a user has been blocked to access the CMS-based Web application.
- **RegDate**: this attribute indicates the date and time when the user was registered into the CMS-based Web application.

We have specified a set of relationships from the *User* element which we explain below.

- **UserRole:** this relationship relates the *User* element with a *Role* element. It indicates the *Role* of the *User* within the CMS-based Web application. This relationship has a cardinality of to-many since it is possible to assign different roles to one user.
- **UserContent:** this relationship is defined between the *User* element and the *Content* element to indicate which content has been created by a certain user. Its cardinality is to-many cardinality since a user can create an undetermined number of content.
- **UserComment:** this relationship relates the *User* element with the *Comment* element. It indicates the comments which have been written by one user. The cardinality of this relationship is a to-many cardinality.

## Role

The *Role* element defines the permissions of a user within the CMS-based Web application. A *Role* determines the functionality that a user can perform within the Web application, such as add a comment or edit a Web page. The attributes that compose this element are defined below:

- **Id:** it is an attribute which determines the unique identifier of the *Role*.
- **Name:** it is an attribute that defines the full name of the *Role*.

From the element *Role*, we have defined three relationships that we present below.

- **RoleUser:** it is a relationship which associates a *Role* with a certain *User*. It is a relationship with a to-many cardinality since it is possible to assign a *Role* to different users.
- **ParentRole:** it is a relationship which permits to establish a hierarchy between *Roles*.
- **RoleVisibilityBlock, roleVisibilityMenu, roleVisibilityMItem, roleVisibilityPage, roleVisibilityCategory:** these relationships define the visibility of these elements according to a certain role. Thereby, the relationship between the *Role* element and the *Block* element called *roleVisibilityBlock* specifies that a block is visible to users belonging to a

certain role. The cardinality of this relationship is to-many since it is possible to set the visibility of a block for a set of *Roles*.

### **Role Group**

The *RoleGroup* element represents the element that allows to group different *Roles* and allows to define for those groups of roles the visibility settings on the elements explained previously. The attributes that compose this element are explained below:

- **Id:** it is an attribute which determines the unique identifier of the *RoleGroup*.
- **Name:** it is an attribute that defines the full name of the *RoleGroup*.

Likewise the *Role* element, the *RoleGroup* element is related to the *Block* element, the *Menu* element, *MenuItem* element, *Page* element and the *Category* element to set the visibility of these elements according to the role group.

### **Permission**

The last element within this concern is the *Permission* element which represents the availability of a certain function in accordance with a *Role*. By means of this element, it is possible to specify the functions which a *Role* can perform within the CMS-based Web application. It is defined with the following attribute:

- **Setting:** this attribute sets the availability of a certain function for a concrete *Role*. The value of this attribute is defined as, the first value is the *enable* value which determines that a certain function is available for a certain role; the second value, is the *disable* value which determines that a certain function is not available for a certain role, finally, the third value is the value *inherit* which is used within hierarchies of roles to determine that a role inherits the permission of its parent role.

We have defined two relationships from the *Permission* element called *functionPermission* which we explain below.

- **Permissionfunction and permissionRole:** the former is a relationship with the *Function* element and the latter with the *Role* element. With these two relationships we can determine whether a *Function* is available or unavailable for a certain *Role*.

## 4.6 Concluding Remarks

In this chapter, we have presented the **CMS Common Metamodel**, the metamodel that specifies the abstract syntax of the CMS DSL proposed in the *ADMigraCMS* method. It captures the key elements of the CMS domain required to define the CMS model.

The process to define this metamodel is composed of four steps: 1) *CMS platform analysis*, where we study the CMS market and we choose the three most used CMS platforms; 2) *CMS metamodels specification*, specification of a metamodel for each of those three CMS platforms capturing their key aspects; 3) *Metamodel modularization*, the definition of a set of concerns in order to classify the elements of those CMS metamodels and finally 4) *CMS Common Metamodel specification*, specification of the CMS Common Metamodel making the union of the elements of the previous three metamodels.

During the study of the **CMS market**, we concluded that from 2010 to 2014 there has been an increase in the use of the open-source CMS platforms (from 70.1% to 73.7%). Concretely, the highest percentages of market share from 2010 up to 2014 are assigned to Wordpress, Joomla! and Drupal, with over a 70% market share. Then, we selected these three CMS platforms: Wordpress, Joomla! and Drupal to base our CMS Common Metamodel (MMCMSCommon).

We analyzed the main features of these three CMS platforms with a metamodel that we called **CMS metamodels**: a Wordpress metamodel (MMWordpress), a Joomla! metamodel (MMJoomla) and a Drupal metamodel (MMDrupal).

According to the complexity of the CMS domain, we experienced the necessity of modularizing the CMS metamodels. Then, we specified a set of **concerns** which allowed us to classify their elements. We defined five concerns based on the Web Engineering views: content, navigation, presentation, behaviour and user.

To build the CMS Common Metamodel we decided to create the **union** of the three previous CMS metamodels to assure that our metamodel allows the modelling and the implementation of a CMS-based Web application based in any of these CMS platforms. After considering this union, we decided to polish the MMCMSCommon discarding some elements which we considered irrelevant as well as redefining other elements to a better definition.

*Chapter 5: The ADMigraCMS  
Method*

---

---



In this chapter, we explain in detail the definition of the *ADMigraCMS* method. This method is based on ADM so that it considers models as a key artefact capturing the knowledge involved in the migration process and automated transformations that systemize this process. The features of this method are introduced in Section 5.1.

The *ADMigraCMS* method specifies a set of modelling languages defined as a Domain Specific Language (DSL). Each modelling language is related to one modelling level. In Section 5.2, we present the specification of these DSLs (PHP DSL, ASTM\_PHP DSL, KDM\_CODE DSL and CMS DSL).

Moreover, it defines a set of transformations that automate the transition between the different modelling levels. The specification of these transformations is done in Section 5.3.

## 5.1 Method Definition

The *ADMigraCMS* method is an ADM-based method that defines standard guidelines to migrate CMS-based Web applications to other CMS platforms. As already mentioned, it considers models at different abstraction levels as key artefacts for the migration process as well as a set of automated transformations systemizing this process.

Moreover, it follows the three reengineering stages proposed by Chikofsky (Chikofsky & Cross, 1990) explained in Chapter 1. In the following, we explain these reengineering stages in the contexts of the *ADMigraCMS* method.

- The **reverse engineering stage** is focused on the extraction of models representing the knowledge obtained from the PHP code that implements the legacy CMS-based Web application and the subsequent abstraction of these models at different abstraction levels. The abstraction levels raise to a model where it is possible to identify the system's components and their relationships within the CMS domain, allowing a better understanding of the CMS-based Web application.
- The **restructuring stage** is centred in restructuring the model representing the system's components in the CMS domain, to meet the features of the

target CMS platform to which the legacy CMS-based Web application is being migrated.

- The **forward engineering stage** uses the restructured model to obtain a set of models defined at different abstraction levels to represent the PHP code that will implement the target CMS-based Web application. At the end of this stage, the generation of PHP code is executed from these models.

The *ADMigracMS* method defines four modelling levels each related to a modelling language and an abstraction level. We define the **Level 0 (L0)** which represents the legacy code that implements the legacy CMS-based Web application; the **Level 1 (L1)** which represents the syntax of this legacy code at PSM level; the **Level 2 (L2)** representing the semantics of the legacy code at PIM level and the **Level 3 (L3)** which represents the system's components in the CMS domain at PIM level. The complete explanations of these modelling levels are presented later.

On the other hand, the *ADMigracMS* method includes a set of automated transformations which allow systemizing the transition from one modelling level to another. Therefore, we define six automated transformations: **L0-to-L1** that defines the T2M transformations between the L0 and L1 modelling levels; **L1-to-L2** defines the M2M transformations between the L1 and L2 modelling levels; **L2-to-L3** defines the M2M transformations between the L2 and L3 modelling levels; **L3-to-L2** defines the M2M transformations between the L3 and L2 modelling levels; **L2-to-L1** defines the M2M transformations between the L2 and L1 modelling levels and, finally, **L1-to-L0** defines the M2T transformations between the L1 and L0 modelling levels. These automated transformations are comprehensively explained in depth in Section 5.1.2.

### 5.1.1 Modelling Levels

In this section, we explain the modelling levels defined in the *ADMigracMS* method. For each modelling level, we specify a modelling language defined as a DSL to represent models capturing the knowledge involved in the migration process at different abstraction levels. Figure 5-1 shows the complete migration process proposed by the *ADMigracMS* method considering the four modelling levels.

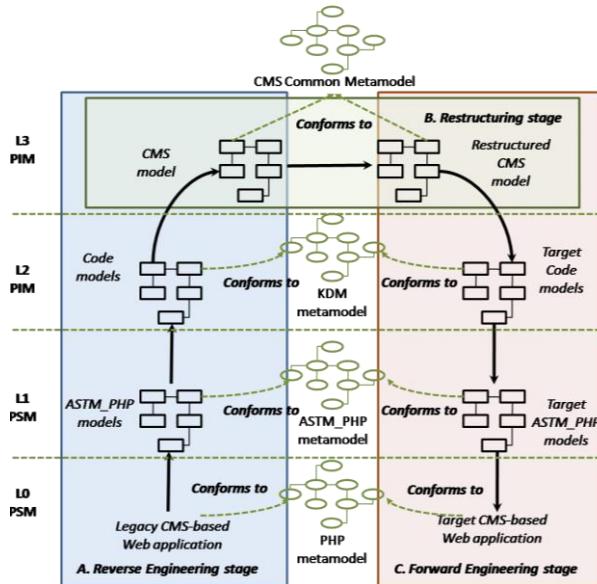


Figure 5-1. ADMigraCMS method and their modelling levels.

- **Level 0 (L0).** This level represents the PHP code that implements the CMS-based Web application. This PHP code conforms to the **PHP DSL**. The abstract syntax of this DSL is defined with the PHP metamodel specified in Section 5.2.1.1. L0 is affected by the reverse engineering stage during the extraction of the ASTM\_PHP models and forward engineering stage in the generation of PHP code from the ASTM\_PHP models, as we can see in Figure 5-1.
- **Level 1 (L1).** In this modelling level we obtain ASTM\_PHP models which represent the syntax of the PHP code implementing the CMS-based Web application. This model is defined at PSM level and it conforms to the **ASTM\_PHP DSL** presented in Section 5.2.2. The abstract syntax of this ASTM\_PHP DSL is defined with the ASTM\_PHP metamodel explained in Section 5.2.2.1. L1 is affected by the reverse engineering stage and forward engineering stage. As it is shown in Figure 5-1, in the reverse engineering stage, the ASTM\_PHP model represents the PHP code implementing the legacy CMS-based Web applications and in the forward engineering stage, it represents the PHP code implementing the target CMS-based Web application.
- **Level 2 (L2).** This level is related to the Code model which represents the semantics of the PHP code implementing the CMS-based Web application at

PIM level. It conforms to the **KDM\_CODE DSL** explained in Section 5.2.3. The abstract syntax of the KDM\_CODE DSL is based on the packages code and action of the KDM metamodel explained in Section 5.2.3.1. As we can see in Figure 5-1, L2 is affected by the reverse engineering and forward engineering stages. In the reverse engineering stage, the Code model represents at PIM level the PHP code implementing the legacy CMS-based Web applications and in the forward engineering stage, it represents the PHP code implementing the target CMS-based Web application.

- **Level 3 (L3).** In this level we define a model called CMS model that represents the components of the CMS-based Web application within the CMS domain at PIM level. This CMS model conforms to the **CMS DSL** which is explained in Section 5.2.4. The abstract syntax of this DSL is specified with the CMS Common Metamodel that we have presented in Chapter 4. L3 is affected mainly by the restructuring stage as it is shown in Figure 5-1. In this stage, the CMS model is restructured considering the features of the target CMS platform.

### **5.1.2 Automated Transformations**

In this section, we present the automated transformations defined between the modelling levels of the *ADMigraCMS* method. These transformations are required to systemize the migration process. We can find different types of transformations: T2M, M2M and M2T transformations.

For each transformation, we present the two modelling levels between which it allows the transition as well as the role played within the migration process. Figure 5-2 presents the reengineering stages along with the modelling levels and the automated transformations.

- **L0-to-L1 T2M Transformation.** This transformation is defined between the modelling levels L0 and L1. It is focused on the extraction of knowledge from the PHP code implementing a legacy CMS-based Web application. Concretely, it generates an ASTM\_PHP model that represents the syntax of this code. Among the different existing reverse engineering techniques (static analysis, dynamic analysis, program slicing and dicing, formal concept analysis, subsystem decomposition) (Canfora & Penta, 2007) the *ADMigraCMS* method opts for using that of static analysis (Zou, Lau, Kontogiannis, Tong, & McKegney, 2004), (Perez-Castillo & García-

Rodríguez de Guzmán, 2009) which is focused on the extraction of knowledge from the code.

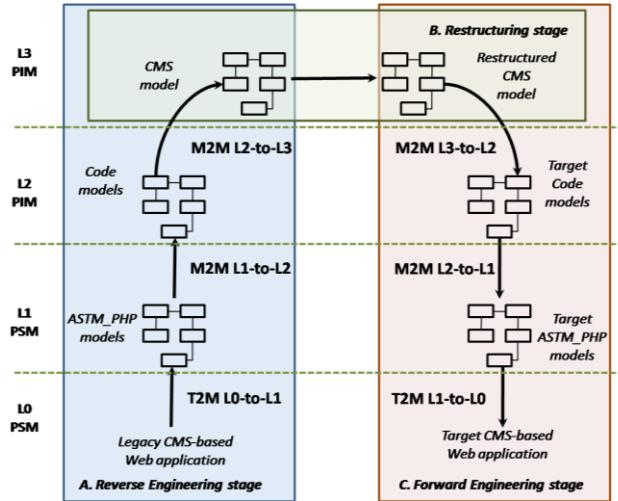


Figure 5-2. Automated transformations between modelling levels.

- **L1-to-L2 M2M Transformation.** This transformation is specified between the modelling levels L1 and L2. This transformation is defined within the reverse engineering stage to raise the abstraction level of the knowledge extracted from the previous PHP code. Specifically, these transformations generate a Code model defined at PIM level from the ASTM\_PHP model specified at PSM level.
- **L2-to-L3 M2M Transformation.** This is the transformation defined between the modelling levels L2 and L3. This transformation is addressed by the reverse engineering stage to generate the CMS model from the Code model. The CMS model models at PIM level the knowledge extracted from the legacy CMS-based Web application within the CMS domain using elements such as menu items, modules or blocks among others elements captured in the CMS Common Metamodel presented in Chapter 4.
- **L3-to-L2 M2M Transformation.** This transformation is between the modelling levels L3 and L2. This transformation is defined within the forward engineering stage to obtain the target Code model from the restructured CMS model. This target Code model represents at PIM level the PHP code that implements the target CMS-based Web application.

- **L2-to-L1 M2M Transformation.** This transformation is defined between the modelling levels L2 and L1. This transformation takes part of the forward engineering stage to decrease the abstraction level of the model that represents the PHP code of the target CMS-based Web application. A target ASTM\_PHP model at PSM level is generated from the previous target Code model defined at PIM level.
- **L1-to-L0 M2T Transformation.** This transformation is defined between the modelling levels L1 and L0. This transformation takes part of the forward engineering stage and it is focused on the generation of the PHP code that implements the target CMS-based Web application. This code is generated from the previous target ASTM\_PHP model.

In next sections, we present the specification of the DSLs that define the modelling languages used to create the models at each modelling level. Furthermore, we present the mappings defined for the automated transformations of the *ADMigraCMS* method.

## 5.2 Specification of the Modelling Languages

For each modelling level, we have defined a modelling language implemented in the form of DSLs that represent the knowledge involved in the migration process defined with the *ADMigraCMS* method. In the following sections, we explain the definition of these DSLs.

### 5.2.1 The PHP DSL

This DSL, called PHP DSL, captures all the syntax elements of the PHP programming language. Its specification is required to implement the model extractor that generates an ASTM\_PHP model from a PHP code. The implementation of this model extractor is presented in Section 6.4.

This DSL is based on the definition of a PHP metamodel which represents its abstract syntax. The concrete syntax of this DSL is defined in a textual manner by the PHP code itself. In this section, we explain the definition of this PHP metamodel. Figure 5-3 presents the L0 modelling level within the specification of the PHP DSL.

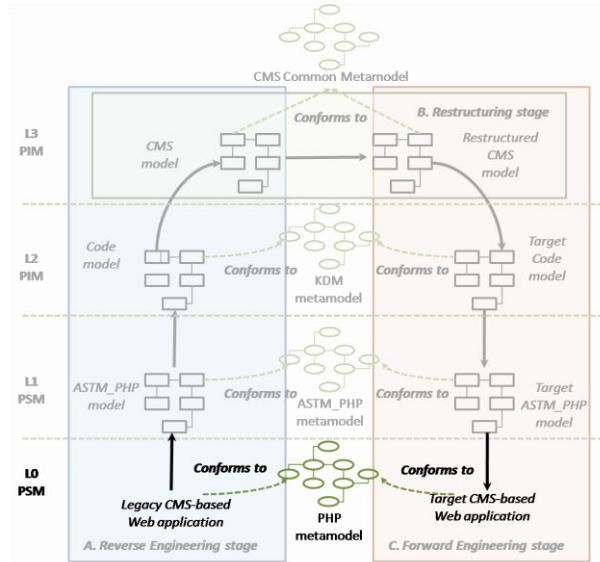


Figure 5-3. L0 modelling level.

We decided to centre the attention on PHP programming language because the proposal is focused on open-source CMS platforms and the 56% of these platforms are supported by PHP (Plain Black Corporation, 2010). Other programming languages are also used to program these platforms but they represent a lower percentage: ASP.NET (4.5%), PERL (8.2%), Java (16.4%) and Python (3.6%).

To the best of our knowledge, there is not any PHP metamodel described in the literature. Given that, we needed to define our PHP metamodel. We consider this definition an interesting contribution for the research community which can be reused by other model-driven approaches. In the following, we present the syntax elements that conform the PHP code and that have been captured in this PHP metamodel.

Any PHP code is defined among these two symbols `<?php` and `?>`. The rest of the syntax elements of the PHP code can be classified in two groups: **PHP Expressions** and **PHP Statements**. Table 5-1 shows the categorization of any PHP element considering these two groups.

**Table 5-1. Categorization of PHP elements.**

<i>GROUP</i>	<i>CATEGORIZATION</i>
PHP Expressions	Simple expressions
	Function call expressions
	Object/class access expressions
	Array access expressions
	Unary expressions
	Binary expressions
PHP Statements	Definitions
	Subroutine calls
	Object Access Statement
	Increment/Decrement Statement
	Control statements

### 5.2.1.1 The PHP Metamodel

In this section, we explain the specification of the PHP metamodel which represents the abstract syntax of the PHP DSL. It is worth noting that the concrete syntax of this DSL is the PHP code itself.

The specification of this metamodel is divided into two parts: the definition of the elements that represent the *PHP Expressions* and the definition of the elements that represent the *PHP Statements*. To explain the specification of these elements, we have used a set of tables whose first column shows the name of the element in the PHP code, whereas the second column presents the name of the element within the PHP metamodel. The third column defines the attributes of the PHP element and the fourth one the conditions that the element from the PHP code has to meet to be represented in the PHP metamodel.

#### PHP Expressions

**PHP Expressions** are code structures which evaluate to a certain value i.e. arithmetic expressions such as `$var+5+3` which evaluates to a number or logical expressions such as `$var or $tar` which evaluates to a logical value. They are considered the core syntax elements since they allow the definition of other elements such as *Statements*. As shown in Table 5-1, *PHP Expressions* can be categorized as follows:

- **Simple expressions:** these expressions refer to simple structures evaluating to a certain value. These simple expressions can be composed to build more complex expressions. Among the simple expressions, we can find *variables*, *literals* and *array definition*. *Literals* can be classified in different types depending on their semantics: octal literal, hexadecimal literal, string literal, Boolean literal, float literal and number literal.

- **Function call expressions:** these expressions execute the call of a function that evaluates to a certain value.
- **Object/class access expressions:** these expressions are used to access variables (also called *members*) or functions defined by a class or an object.
- **Array access expressions:** this kind of expression is used to access the positions of an array to consult or modify the value stored in that position.
- **Unary expressions:** these expressions consider just one operator and one operand and evaluates to a certain value. Unary operators considered within the PHP code are explained later in this section.
- **Binary expressions:** these expressions consider one binary operator and two operands, *left operand* and *right operand*, and also they evaluate to a certain value. The binary operators defined in PHP code are presented later in this section.

### *Simple Expressions*

After presenting the different types of PHP expressions, we show the representation of the *simple expressions* within the PHP metamodel. To do it, we have established a set of mappings between the elements of the PHP code and the elements of the PHP metamodel (PHP elements). These mappings are shown in Table 5-2.

**Table 5-2. Mappings from simple expressions to PHP elements.**

<b>PHP CODE</b>	<b>PHP METAMODEL</b>		<b>CONDITION</b>
	<i>Element</i>	<i>Properties</i>	
<b>Variable</b>	<b>Name</b>	nameIdentifier	If <i>Variable</i> is defined the first time.
	<b>Variable</b>	value	If the <i>Variable</i> has been already defined
<b>Literal</b>	<b>NumberLiteral</b> <b>StringLiteral</b> <b>FloatLiteral...</b>	value	Depending on the type of value represented by the literal.
<b>Array Definition</b>	<b>ArrayType</b>	params	No conditions.
	<b>ArrayEntry</b>	Key, value	No conditions.

- **Mapping from the element Variable to the elements Name and Variable:** The element *Variable* is mapped to two elements in the PHP metamodel (*Name* and *Variable*). On the one hand, if it represents the first definition of a variable, this variable is mapped to the *Name* element. This element defines a string attribute called *nameIdentifier* storing the name of the variable. On the other hand, if it represents a variable that has already been defined

previously, it is mapped to the *Variable* element. It is composed of one string attribute called *value* that stores the name of the variable. Even if these two elements seem to be identical, we have decided to differentiate them because of their different semantics and because of it eases us the definition of the following mappings between the PHP elements and the ASTM\_PHP elements.

- ***Mapping from the element Literal to the element Literal (StringLiteral, NumberLiteral...):*** The element *Literal* is mapped to six different PHP elements (*NumberLiteral*, *StringLiteral*, *FloatLiteral*, *HexadecimalLiteral*, *OctalLiteral* and *BooleanLiteral*) depending on the type of the value represented by the *Literal*. They have the attribute *value* where is stored the literal value.
- ***Mapping from the element Array Definition to the elements ArrayType and ArrayEntry:*** the element *Array Definition* is mapped to two different PHP elements, *ArrayType* and *ArrayEntry*. *ArrayType* element represents the new array. It has an attribute called *params* which stores the array entries that can be specified during the array definition. These array entries define a position in the array and the value to be stored. These elements are mapped to the *ArrayEntry* element in the PHP metamodel, composed of two attributes *key* that stores the position and *value* that stores the value to be stored.

#### *Function Call Expressions and Array Access Expressions*

In this section, we present the representation of the *function call expressions* and the *array access expressions* into the PHP metamodel. The mappings established for these elements are shown in Table 5-3. In Appendix C the mapping for the *object/class access expressions* is presented.

**Table 5-3. Mappings from PHP expressions to PHP elements.**

<b>PHP CODE</b>		<b>PHP METAMODEL</b>		<b>CONDITION</b>
<b>Element</b>	<b>Element</b>	<b>Properties</b>		
<b>Function call expression</b>	<b>FunctionCallExpression</b>	functionIdentifier, params		No conditions
<b>Array access expression</b>	<b>ArrayAccess</b>	arrayIdentifier, sus		No conditions
	<b>Subscript</b>	expression		No conditions

- ***Mapping from the element Function Call Expression to the element FunctionCallExpression:*** The element *Function Call Expression* is mapped to the PHP element *FunctionCallExpression*. This element is composed of

two attributes, *functionIdentifier* and *params*. The former stores the name of the function and the latter stores the parameters passed to this function call.

- **Mapping from the element Array Access Expression to the elements ArrayAccess and Subscript:** The *Array Access Expression* is mapped to two elements in the PHP metamodel, *ArrayAccess* and *Subscript*. The *ArrayAccess* element represents the access to a class. It is composed of two attributes, *arrayIdentifier* and *sus*. The former stores the name of the array and the latter stores the positions in the array being accessed. These positions are mapped to the *Subscript* element which is defined with one attribute called *expressions* storing the expression representing the position.

### *Unary Expressions*

In this section, we present the representation of the *unary expressions* into the PHP metamodel. We have established a set of mappings to relate this kind of expressions to the PHP elements. These mappings depend on the *unary operators* of the expressions. Table 5-4 presents the existing *unary operators* in PHP which have been classified into seven groups.

**Table 5-4. Unary operators in PHP.**

<i>OPERATOR TYPE</i>	<i>UNARY OPERATOR</i>
Casting	(int) (float) (string) (array) (object) (bool)
Arithmetic	- (negation)
Bitwise	~ (not)
Logical	! (not)
Reference	&
Class	New, clone
Increment/Decrement	++, --

- **Casting operator:** there are six unary casting operators (int, float, string, array, object, bool) which allow casting a variable as a specific type.
- **Arithmetic operator:** the only unary arithmetic operator in PHP is the negation that allows changing the sign of an arithmetic value.
- **Bitwise operator:** it is a unary operator (not) which allows the evaluation and manipulation of specific bits within an integer.
- **Logical operator:** it is a unary operator (not) which allows changing the sign of a Boolean value.
- **Reference operator:** it is a unary operator (&) which allows accessing the same variable content by different names.

- **Class operator:** they are two unary operators (`new`, `clone`) allowing the access to an object or class.
- **Increment/decrement operator:** they are two unary operators (`++`,`--`) allowing the pre- and post- increment and decrement operation.

It is worth mentioning that for the definition of these mappings we need to consider the order of precedence between operators (unary and binary operators) in PHP because this precedence specifies how tightly two expressions are bind together (PHP Group, 2013). For instance, in the expression `1 + 5 * 3`, the result is `16` and not `18` because the multiplication ("\*") operator has a higher precedence than the addition ("+" ) operator. Table 5-5 shows the mappings between some of the *unary expressions* and the PHP elements. The mappings for the rest of *unary expressions* can be consulted in Appendix C.

**Table 5-5. Mappings from unary expressions to PHP elements.**

<i>PHP CODE</i>		<i>PHP METAMODEL</i>		<i>CONDITION</i>
<i>Element</i>	<i>Element</i>	<i>Properties</i>		
<b>Unary expression</b>	<b>NegateOrCast</b>	Operator, right, cast		If the unary operator is casting (int) (float) (string) (array) (object) (bool), arithmetic - (negation) or bitwise ~ (not)
	<b>LogicalNot</b>	exp		If the unary operator is logical ! (not)
	<b>Reference</b>	value_ref		If the unary operator is reference &

- **Mapping from the element Unary Expression to the element NegateOrCast:** If the *Unary Expression* is defined with a unary operator of the type *casting*, *arithmetic* or *bitwise* it is mapped to the PHP element *NegateOrClass*. This element is composed of three attributes, *operator*, *right* and *cast*. If the operator of the *unary expression* is an *arithmetic* or *bitwise* operator, it is stored in the attribute *operator*. Otherwise, if the operator is a *casting* operator, it is stored in the attribute *cast*. The attribute *right* stores the expression affected by the unary operator.
- **Mapping from the element Unary Expression to the element LogicalNot:** If the *Unary Expression* is defined with a unary operator of the type *logical*, it is mapped to the PHP element *LogicalNot*. This element is defined with one attribute called *expr* which stores the expression evaluated by the logical operator.
- **Mapping from the element Unary Expression to the element Reference:** If the *Unary Expression* is defined with a unary operator of the type *reference* it is mapped to the PHP element *Reference*. This element is composed of one

attribute called *value\_ref*. This attribute stores the expression assessed by the *reference* operator.

### *Binary Expressions*

In this section, we present the representation of the *binary expressions* into the PHP metamodel. As occurred with the *unary expressions*, the binary expressions are mapped to different PHP elements depending on its binary operator. Table 5-6 presents the *binary operators* in PHP classified into six groups.

**Table 5-6. Binary operators in PHP.**

OPERATOR TYPE	BINARY OPERATOR
Assignment	= += -= *= /= .= %= &=  = ^= <<= >>= ==>
Arithmetic	+, -, *, /, %
Logical	And, or, xor, &&,
Bitwise	&,  , ^, <<, >>
Class	instanceof
Comparison	==, !=, ===, !==, <>, <, <=, >, >=

- **assignment operator:** there are thirteen assignment binary operators in PHP which allow to assign a value to a variable. These assignment operators can be classified into two groups: simple and composed. The simple assignment operator is that allowing a simple assignment (=), otherwise a composed assignment operator is defined along with another operator like an arithmetic operator, e.g. +=, \*=.
- **Arithmetic operator:** there are five arithmetic binary operators in PHP (+, -, \*, /, %) to define arithmetic functions.
- **Logical operator:** there is five binary operators (and, or, xor, &&, ||) which allows defining expressions evaluating to a Boolean value.
- **Bitwise operator:** there are five operators (&, |, ^, <<, >>) which allow the evaluation and manipulation of specific bits within an integer.
- **Class operator:** there is one binary operator (instanceof) which evaluates the nature of an object. It returns a Boolean value.
- **Comparison operator:** there are nine binary operators (==, !=, ===, !==, <>, <, <=, >, >=) which allow to compare the value of two expressions.

In Table 5-7, we present some of the mappings between the *binary expressions* and the PHP elements. They are so similar, so that we have decided to

explain just one of them (*Assignment*) The rest of mappings can be found in Appendix C.

**Table 5-7. Mappings from binary expressions to PHP elements.**

<i>PHP CODE</i>	<i>PHP METAMODEL</i>		<i>CONDITION</i>
	<i>Element</i>	<i>Properties</i>	
<b>Binary expression</b>	<b>Assignment</b>	Left, operator, right	If the binary operator is an assignment operator
	<b>Addition</b>	Left, operator, right	If the binary operator is one of these two arithmetic operators (+, -)
	<b>EqualityCheck</b>	Left, operator, right	If the binary operator is one of these comparison operators ==, !=, ===, !==, <>
	<b>BitWiseShift</b>	Left, operator, right	If the binary operator is one of these bitwise operators <<, >>
	<b>LogicalAnd</b>	Left, operator, right	If the binary operator is one of these two logical operators (&&)
	<b>Instanceof</b>	Left, operator, right	If the binary operator is the instanceof operator

- **Mapping from the element Binary Expression to the element Assignment:** As we can see in Table 5-7, the *Binary Expression* defined with a binary operator of type *assignment* is mapped to the PHP element *Assignment*. This element is composed of three attributes, *operator*, *right* and *left*. The attribute *operator* stores the operator of the binary expression, the attribute *left* stores the left operand and the attribute *right* stores the right operand of the binary expression.

### PHP Statements

After the explanation of the representation of the *PHP Expressions* in the PHP metamodel, we present in the following sections the representations of the **PHP Statements** in this metamodel. *PHP Statements* are code structures that represent the sentences written in PHP. The different *PHP Statements* are separated by semicolons and they are composed of a set of *PHP Expressions*. As we can see in Table 5-8, the *PHP Statements* can be categorized in three categories: *Definitions*, *Expression Statements* and *Control Statements*.

**Table 5-8. Categorization of PHP Statements.**

<i>TYPE OF STATEMENT</i>	<i>STATEMENTS</i>
Definitions	Variable definitions
	Function definitions
	Class definitions
Expression Statements	Subroutines
	Object access statements
	Increment/decrement statements

Control Statements	If statements
	Switch statements
	For statements
	Foreach statements
	While statements
	Return statements

- **Definitions:** these statements allow the definition of different elements in PHP: *variable definitions*, *function definitions* and *class definitions*.
- **Expression statements:** these statements are similar to expressions but are defined with a semicolon at the end of the statement defining an independent entity. We can find different types of expression statements such as, *subroutines*, *object access statements* and *increment/decrement statements*.
- **Control statements:** the control statements define structures that manage the flow of execution of a program written in PHP code. We can find different kinds of control statements, such as, *if statements*, *switch statements*, *for statements*, *foreach statements*, *while statements* and *return statements*

### Definition Statements

In the following, we present the representation of the *definition statements* in the PHP metamodel. We have defined a set of mappings between this kind of statements and the elements of the PHP metamodel. Table 5-9 shows these mappings. The rest of mappings for these statements can be found in Appendix C.

**Table 5-9. Mappings from definition statements to PHP elements.**

PHP ELEMENT	PHP METAMODEL		CONDITION
	Element	Properties	
Variable definition	VariableDefStatement	Visibility, left, right	No conditions
Function definition	FunctionDefStatement	Visibility, functionIdentifier, params, body	No conditions
	FuncParameters	Var, value	No conditions

- **Mapping from the element Variable Definition to the element VariableDefStatement:** The *Variable Definition* is mapped to the element *VariableDefStatement* in the PHP metamodel. It is composed of three attributes, *visibility*, *left* and *right*. The attribute *visibility* stores the visibility level of the variable. To define the visibility there are three states (*public*, *protected* and *private*). 1) *public* state considers the variable accessible from anywhere; 2) *protected* state makes the variable accessible to the element itself and to the inherited and parent elements, finally 3) *private* state converts the variable just available to the parent element. The attribute *left*

stores the name of the variable, whereas the attribute *right* stores the expression which defines the initial value for this variable definition.

- **Mapping from the element Function Definition to the elements FunctionDefStatement and FuncParameters:** The *Function Definition* is mapped to two elements the *FunctionDefStatement* and *FuncParameters* in the PHP metamodel. The *FunctionDefStatement* element is composed of four attributes, *visibility*, *functionIdentifier*, *params* and *body*. As occurred with the variable definition, the attribute *visibility* defines the visibility level of the function, whereas the attribute *functionIdentifier* defines the name of the function; otherwise, the attribute *params* stores the parameters defined for the function definition. Finally, the attribute *body* stores the statements composing the body of the function definition. Each parameter stored in the attribute *params* is mapped to the element *FuncParameters* defined with two attributes (*var* and *value*). The *var* attribute is used to store the name of the parameter and the attribute *value* stores the optional initializing value.

### *Expression Statements*

In this section, we show the representation of the expression statements within the PHP metamodel. Table 5-10 presents the mappings that make this representation possible. The rest of the mappings for this kind of statements are presented in Appendix C.

**Table 5-10. Mappings from expression statements to PHP elements.**

<b>PHP ELEMENT</b>	<b>PHP METAMODEL</b>		<b>CONDITION</b>
	<i>Element</i>	<i>Properties</i>	
<b>Subroutine</b>	<b>FunctionCallStatementExp</b>	exp	No conditions
<b>Increment/decrement statements</b>	<b>IncrementStatement</b>	Operator, right, left	No conditions

- **Mapping from the element Subroutine to the element FunctionCallStatementExp:** The *Subroutine* is mapped to the element *FunctionCallStatementExp* in the PHP metamodel. This element is composed of the attributes *exp* where the subroutine is stored.
- **Mapping from the element Increment/Decrement Statement to the element IncrementStatement:** The *Increment/Decrement Statements* are mapped to the element *IncrementStatement* in the PHP metamodel. This element is defined with three attributes *operator*, *right* and *left* as occurred with the *Increment* element. The attribute *operator* stores one of the operators *increment* or *decrement*, whereas the attributes *left* and *right* store the

expression assessed by these operators. The attribute *left* stores the expression whether the expression defined as a post-increment/decrement expression. Otherwise, whether it is defined as a pre-increment/decrement, the expression is stored in the attribute *right*.

### **Control Statements**

Finally, in this section we present the representation of the *control statements* in the PHP metamodel. Table 5-11 shows some of the mappings of this third group of *PHP Statements* to PHP elements. In Appendix C, it is possible to consult the rest of mappings for this kind of statements.

**Table 5-11. Mappings from control statements to PHP elements.**

<b>PHP ELEMENT</b>	<b>PHP METAMODEL</b>		<b>CONDITION</b>
	<b>Element</b>	<b>Properties</b>	
<b>If statements</b>	<b>IfStatement</b>	condIf, ifStatements, listElsif, elseStatements	No conditions
	<b>Elsif</b>	condElsif, elsifStatements	No conditions
<b>For statements</b>	<b>ForStatement</b>	exp1, exp2, exp3, statements	No conditions
<b>Return statements</b>	<b>ReturnStatement</b>	return	No conditions

- **Mapping from the element If Statement to the elements IfStatement and Elsif:** The *If Statements* are mapped to two elements in the PHP metamodel, the element *IfStatement* and the element *Elsif*. The *IfStatement* element is composed of four attributes, *condIf*, *ifStatements*, *listElsif* and *elseStatements*. The *condIf* attribute stores the condition of the *if* statement, whereas the attribute *ifStatements* stores the statements defining the body of the *if* statement. Otherwise, the attributes *listElsif* and *elseStatements* define the alternatives of the *if* statement. These alternatives are represented by the element *Elsif*. This element is composed of two attributes, *condElsif* and *elsifStatements*. The former represents its condition and the latter the statements composing the body of the alternative.
- **Mapping from the element For Statement to the element ForStatement:** The element *For Statement* is mapped to the element *ForStatement* in the PHP metamodel. The *ForStatement* element is composed of four attributes *exp1*, *exp2*, *exp3* and *statements*. The three first attributes represent three required expressions separated by semicolons. *Exp1* stores an expression which is executed at the beginning of the iteration. Usually, it is a variable definition. *Exp 2* is the condition which continues or stops the iteration. *Exp 3* increments or decrements the value of variable defined in *exp1*. The statements composing its body are stored in the attribute *statements*.

- **Mapping from the element Return Statement to the element ReturnStatement:** The *Return Statements* are mapped to the element *ReturnStatement* in the PHP metamodel. It is composed by the attribute *return* which stores the value to be returned by the statement.

### 5.2.2 The ASTM\_PHP DSL

In this section, we explain the specification of the ASTM\_PHP DSL. This DSL defines a modelling language to represent a model that contains the knowledge extracted from the PHP code. The abstract syntax of this DSL is defined on the ASTM\_PHP metamodel, explained in Section 5.2.2.1. This metamodel is based on the ASTM standard metamodel proposed by ADM and it captures the syntax elements of the PHP programming language.

Instead of using the PHP metamodel defined in the previous section to define the abstract syntax of this DSL, we decided to use the ASTM\_PHP metamodel because it is based on a standard that provides the DSL with a recognized level of quality and facilitates the interoperability.

In Section 5.2.2.2 , we specify the concrete syntax of this DSL defining the graphical notation of the elements captured in the ASTM\_PHP metamodel. Figure 5-4 relates the L1 modelling level with the specification of the ASTM\_PHP DSL.

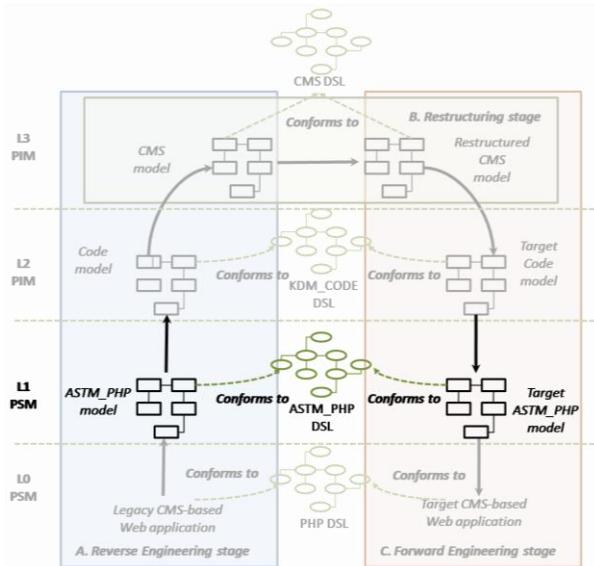


Figure 5-4. L1 modelling level.

### 5.2.2.1 The ASTM\_PHP Metamodel

As we mentioned, the ASTM\_PHP metamodel is based on the ASTM standard metamodel which has been extended with the specific elements of the PHP code. The ASTM metamodel allows representing in a model the syntax and the basic semantics of a source code. As already said, this metamodel is composed of two domains. GASTM domain, that captures the common elements of the different programming languages; and SASTM domain, that gathers the specific elements of a concrete programming language.

In the following, we present the core elements conforming the GASTM domain to understand better the definition of the ASTM\_PHP metamodel. Figure 5-5 presents a hierarchy with the main elements of GASTM.

#### GASTM domain

As we can see in Figure 5-5, the root element of the hierarchy is the *GastmObject* element that is specialized in three elements: *GastmSourceObject*, *GastmSemanticObject* and *GastmSyntaxObject*. In next sections, we explain them.

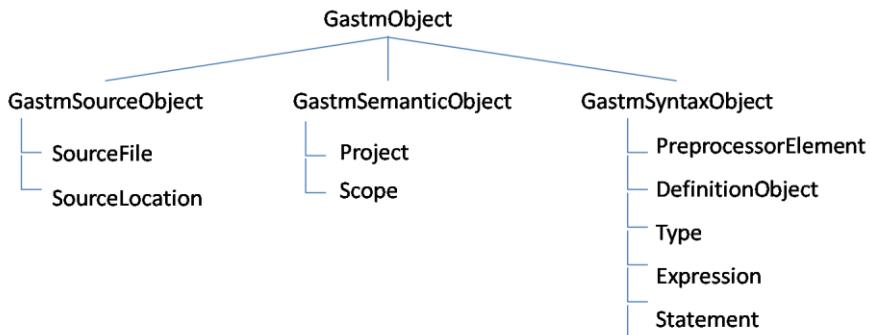


Figure 5-5. Hierarchy of GASTM elements.

- **GastmSourceObject Element:** This element represents the source properties of the programming language elements. It is specialized into two subelements *SourceFile* and *SourceLocation*. In turn, the *SourceFile* element is specialized into two elements, the *CompilationUnit* and the *SourceFileReference*. The *CompilationUnit* represents the files that contain the code. It is composed of the following attributes: *language*, *fragments* and *openScope*. The attribute *language* defines the name of the programming language used to implement the code, the attribute *fragments* stores the different definitions included in the code (*class definitions*, *function*

*definitions or variable definitions*) and the attribute *openScope* defines the semantic of the file.

- **GastmSemanticObject Element:** This element is used for modelling certain kinds of basic semantic properties between ASTM elements. It is specialized into two elements: *Project* and *Scope*. The element *Project* represents the project that is being analyzed. A *Project* is defined as a set of *CompilationUnits*, presented in the previous section. The element *Scope* describes the largest declarative region (as part of a program). *Scope* is specialized into several kinds of scopes: *FunctionScope*, *AggregateScope*, *BlockScope*, *ProgramScope* and *GlobalScope*.
- **GastmSyntaxObject Element:** This element represents the syntactic elements of the code. It is subclassed in six different elements: *PreprocessorElement*, which refers to source code that is typically processed by a preprocessor and converted into the code that is processed by a parser or a compiler; *DefinitionObject*, refers to the definition structures such as variable definitions or function definitions; *Type*, represents the types that can be defined within the code; *Expression*, represents the expressions which can be found in the code; *Statement*, represents the different statements which compose the code and *MinorSyntaxObject* which represents that syntax elements that allow defining other syntax elements such as expressions or statements.

Some of the elements defined within the ASTM\_PHP metamodel extend the following *GastmSyntaxObject* elements: *Expression*, *Statement* and *DefinitionObject* that are marked in Figure 5-6 and explained below:

- **Statement Element:** The *Statement* element represents the statements of a programming language. As we can see in Figure 5-6, the *Statement* element is specialized in two elements: the *BlockStatement* and the *ExpressionStatement*. The *BlockStatement* element represents blocks of statements, whereas the *ExpressionStatement* element contains an expression such as *assignments*, *references*, *literals* or *function calls*. Even not appearing in Figure 5-6, the *Statement* element is specialized in more elements such as, *IfStatement*, *SwitchStatement*, *ForStatement*, *WhileStatement*, *ReturnStatement* and *DefinitionOrDeclarationStatement*.

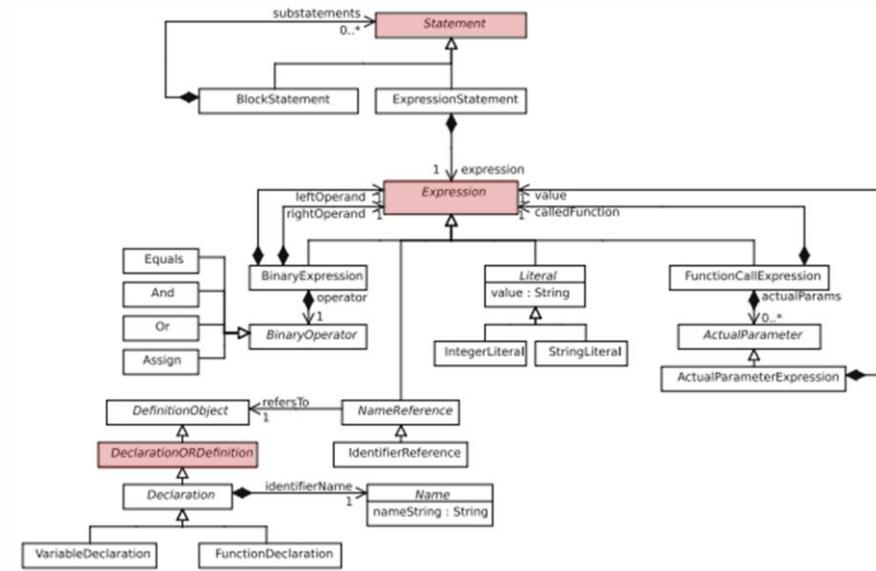


Figure 5-6. **Expression**, **Statement** and **DefinitionObject** elements of the GASTM domain.

- **Expression Element**: The expressions of a programming language are represented by the *Expression* element. As it is shown in Figure 5-6, the *Expression* element is subclassed in a set of elements: *BinaryExpression*, *NameReference*, *Literal* and *FunctionCallExpression*, but we can find other ones, such as *UnaryExpression*, *CastExpression* and *CollectionExpression*. The *BinaryExpression* represents both assignment and *logical expressions*; the *NameReference* element allows elements to refer to other abstract syntax elements, this element is specialized in the element *IdentifierReference*; the *Literal* element represents literal values, the *FunctionCallExpression* element represents function calls; the *UnaryExpression* represents expressions defined with unary operators; the *CastExpression* defines expressions used to explicitly convert an expression to a given type and the *CollectionExpression* represents structures to handle collections of elements.
- **DefinitionObject Element**: This element represents the definition of other syntax elements such as variables or functions among others. As we can see in Figure 5-6, this element is specialized in the element *DeclarationOrDefinition*, in turn, it is specialized into two elements, *Declaration* and *Definition*. As subelements of the element *Declaration*, there are the elements *VariableDeclaration* and *FunctionDeclaration* otherwise the *Definition* element is subclassed in the *VariableDefinition*,

*FunctionDefinition*, *FormalParameterDefinition* and *AggregateTypeDefintion* elements.

### SASTM domain

After explaining the main elements of the GASTM domain, we explain the process followed to specify the ASTM\_PHP metamodel. The main objective of this metamodel is to allow the mappings between the PHP elements and the ASTM elements. We extended the ASTM metamodel with a set of elements representing specific PHP elements. These elements are classified in those which were not considered in GASTM domain and those even existing in GASTM did not fit well with the specification of the PHP element and needed to be redefined.

All these new specific elements are defined within the SASTM (SASTM elements) domain as specializations of elements of the GASTM (GASTM elements). Thus, in this section we explain how we have defined these elements in the SASTM.

Table 5-12 presents all the elements included within SASTM. The first column refers to the GASTM element that has been specialized; the second column denotes the name of the SASTM element; the third column indicates whether the SASTM element is a new or a redefined element and finally, the fourth column is a description of the SASTM element. Most of these SASTM elements represent new operators and expressions. Otherwise, three of them represent redefined elements (two statements and one expression).

**Table 5-12. Elements defined within the SASTM.**

GASTM ELEMENT	SASTM ELEMENT	NEW/REDEFINED	DESCRIPTION
<b>EXPRESSION</b>	ObjectAccess	New	Access to an object
	ClassAccess	New	Access to class
	ArrayAccessPHP	New	Access to an array
<b>BINARY OPERATOR</b>	DuplaArray	New	Array entry
	Xor	New	Boolean operator
	NotIdentical	New	Boolean operator
	Identical	New	Boolean operator
<b>UNARY OPERATOR</b>	InstanceOf	New	Boolean operator
	New	New	Creates an object
	Clone	New	Creates a copy
<b>STATEMENT</b>	ForStatement PHP	Redefined	For loop
	Switch StatementPHP	Redefined	Switch condition
	ForEach Statement	New	For each loop
<b>COMPILEATIONUNIT</b>	Compilation UnitPHP	Redefined	File containing PHP code

Figure 5-7 shows the extension the GASTM element *Expression* to represent the specific PHP expressions. Below, we explain these new SASTM elements.

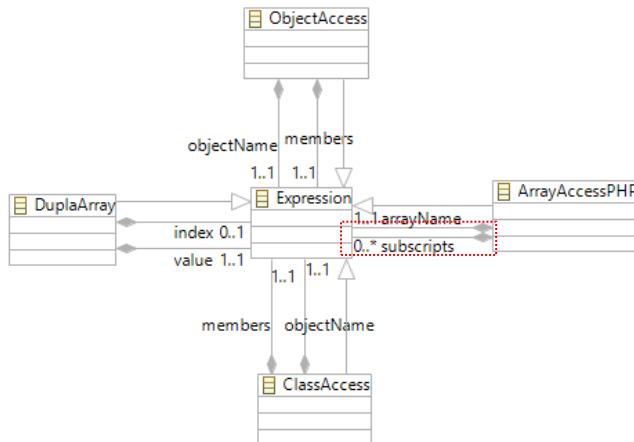
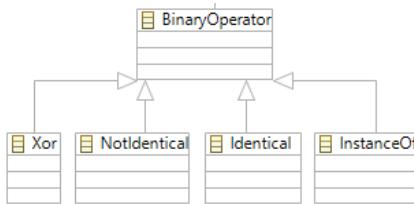


Figure 5-7. Specific PHP expressions defined in SASTM.

- **ObjectAccess and ClassAccess elements**, these two elements represent two expressions that do not exist within GASTM. The former represents the access to a member or a function of an object, and the latter represents the access to the same items, but these ones belonging to a class. As we can see in Figure 5-7, the definitions of these two elements are similar, but from the semantic point of view are different, so that we decided to define two different elements.
- **ArrayAccessPHP element**, this element represents the access to a position of an array. This element redefines the *ArrayAccess* element of GASTM. Concretely, we redefined the cardinality of the relationship *subscripts*. This relationship specifies the positions of the array (e.g. array [1] [2]). In the GASTM domain, the definition of these positions is required since cardinality of subscripts is 1..\*. Otherwise, an array access in PHP can be specified without the definition of any position so that we needed to redefine the cardinality of *subscripts* to 0..\*. This change is marked in a red dotted line in Figure 5-7.
- **DuplaArray element**, it is a new element which has been included within the SASTM to represent the key-value pairs. It is composed of two attributes called *index* and *value* defined as two aggregation relationships, as it is shown in Figure 5-7. On the one hand, the *index* is a non-required attribute

that refers to an expression representing the index of the pair, on the other hand, the *value* is a required attribute linked to an expression representing the value.

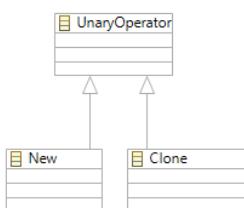
Figure 5-8 presents the subelements defined in SASTM from the GASTM element *BinaryOperator*. In the following, we explain each element.



**Figure 5-8. Specific PHP binary operators defined in SASTM.**

- **Xor element**, it represents the *xor* logical operator. It did not exist within the standard definition of GASTM, thus it was inserted as a new element within the SASTM domain. This element inherits all the attributes from its parent *BinaryOperator* element.
- **NotIdentical and Identical elements**, they represent the comparison operators `!==` and `==`, respectively. These two elements did not exist within the standard definition of GASTM either. They were defined without any specific attribute, thus its attributes are inherited from the GASTM element *BinaryOperator*.
- **InstanceOf element**, it represents a logical operator which allows checking whether an object is an instance of a certain class. It did not exist within the standard definition of GASTM and thus, it was inserted as a new element within the SASTM. It just provides the attributes inherited from the element *BinaryOperator*.

In Figure 5-9 we present the two subelements which extend the GASTM element *UnaryOperator*. In the following, we explain them.



**Figure 5-9. Specific PHP unary operators defined in SASTM.**

- **New and Clone elements**, they represent the unary operators which allow creating an object from a certain class as well as to generate a copy of a certain object. It neither exists within the standard definition of GASTM and thus they were inserted as a new element within the SASTM domain.

In Figure 5-10, we show the three elements that extend the GASTM element *Statement*. These new elements represent specific statements of PHP. As we can see, these three elements are related to the element *Expression* to define some of their attributes.

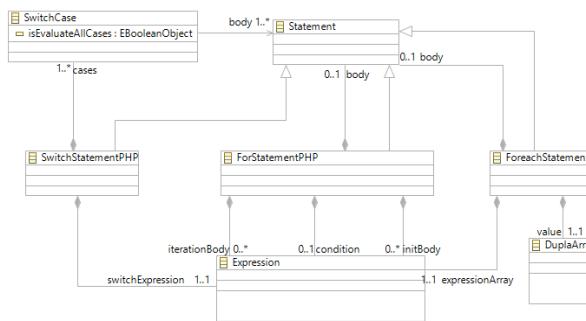
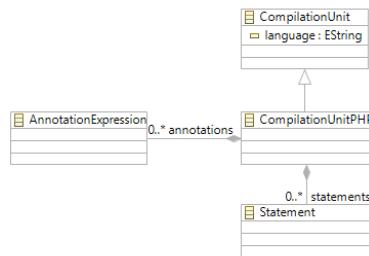


Figure 5-10. Specific PHP statements defined in SASTM.

- **ForStatementPHP element**, it is an element which represents the *for* iterations in PHP. In GASTM there is an element called *ForStatement*, but its specification does not fit well with the *for* statements of PHP, so that we needed to redefine this element. The reason is because PHP allows defining *for* statements without any condition and the attribute *condition* of the GASTM element is a required attribute that must store one value at least. Accordingly, we had to redefine the *ForStatement* element by converting the attribute *condition* as an optional attribute.
- **SwitchStatementPHP element**, it is an element defined from the GASTM element *Statement*. In GASTM, there is an element called *SwitchStatement* but its specification does not fit well with the specification of this PHP statement. The main reason is because its attribute *cases* is not multivalued so that it is not possible to assign a set of *Case* statements. Hence, we redefined this *SwitchStatement* element with the *SwitchStatementPHP* element by defining the attribute *cases* as a multivalued attribute.
- **ForeachStatement element**, it is a new element defined within the SASTM domain. This element has three attributes: *expressionArray*, *value* and *body*.

These three attributes are defined as aggregation relationships. The attribute *expressionArray* is a required attribute related to the element *Expression* that stores the name of the array being iterated. The attribute *value* is related to the element *DuplaArray* and stores the element located in the position accessed. Finally, the *body* attribute is a non-required attribute which stores the set of statements that conform the body of the iteration.

Finally, in Figure 5-11 we can see the redefinition of the GASTM element *CompilationUnit* into the element *CompilationUnitPHP*. We explain the reasons why we have specialized this element.



**Figure 5-11. *PHPCompilationUnit* element defined in SASTM.**

- **CompilationUnitPHP element**, it represents the physical file where the PHP code is defined. This element is the specialization of the element *CompilationUnit* from GASTM. The main reason to redefine this element is that this element constraints the specification of statements within a *definition* such as a class definitions or a function definitions. It is a feature clearly focused on the object-oriented programming languages. It does not fit well for programming languages which are not object-oriented, such as PHP. Therefore, we redefined this element by the creation of the element *CompilationUnitPHP* to allow the insertion of statements out of a *definition*. So that, we defined the multivalued and non-required attribute *statements* that is a reference to the element *Statement*. Moreover, we defined the multivalued attribute *annotations* which allows the insertion of comments within the code. The attribute *annotations* is a reference to the GASTM element *AnnotationExpression*.

### 5.2.2.2 Customized Graphical Notation

After defining the abstract syntax of the ASTM\_PHP DSL with the ASTM\_PHP metamodel, we present the concrete syntax of this DSL. Thus, in this

section, we define the graphical notation of the elements gathered in the ASTM\_PHP metamodel. Thus, we make these elements more intuitive visually.

Table 5-13 shows the graphical notation of some of the ASTM\_PHP elements which appear during the validation of this proposal in Chapter 7. The first column is the name of the ASTM\_PHP element, the second column contains its corresponding graphical notation and finally, the third column explains textually the description of the graphical notation.

**Table 5-13. Customized graphical notation of the ASTM\_PHP DSL.**

ASTM_PHP ELEMENT	GRAPHICAL NOTATION	DESCRIPTION
Project		The <i>Project</i> element is graphically represented with a folder. It represents the CMS-based Web application.
CompilationUnit PHP		The <i>CompilationUnit</i> element is graphically represented with a document which has the initials <i>PHP</i> printed.
SourceLocation		This element is represented as a document as a software asset.
Function Definition		The <i>FunctionDefinition</i> is represented with a box containing a PHP code that defines the function.
FunctionCall Expression		This element is represented with an arrow pointing to the function called.
ByValueActualParameterExpression		This element is represented by a pair of curly brackets to represent the parameter of a function call.
VariableDefinition		The <i>VariableDefinition</i> is represented with the characters <i>x</i> and <i>=</i> emulating the assignment to a variable.
Collection Expression		The <i>CollectionExpression</i> is represented as a set of elements ordered sequentially.
DuplaArray		The <i>DuplaArray</i> is represented with the pair of brackets used in PHP to define these entries.
ReturnStatement		The <i>ReturnStatement</i> is represented with a circular arrow indicating the return of a value.
BinaryExpression		This element is represented with a character <i>x</i> indicating the left operand, a <i>+</i> for the operator and the <i>y</i> for the right operand.
ArrayAccessPHP		This element is represented with the character <i>x</i> representing a variable and a pair of square brackets for the array access.
Assign		The <i>Assign</i> is represented with the mathematical symbol <i>=</i> representing the assignment.

ASTM_PHP ELEMENT	GRAPHICAL NOTATION	DESCRIPTION
StringLiteral	ABC	The <i>StringLiteral</i> is represented with the letters ABC simulating a string of characters.

These graphical notations are considered during the implementation of the tree-like graphical editors in Chapter 6. These graphical editors will allow us to define models conforming to the ASTM\_PHP DSL.

### 5.2.3 The KDM\_CODE DSL

In this section, we present the KDM\_CODE DSL that defines the modelling language to represent the Code models for the *ADMigraCMS* method. The abstract syntax of this DSL is based on the packages *code* and *action* of the KDM metamodel explained in Section 5.2.3.1.

In Section 5.2.3.2, we specify the concrete syntax of this DSL defining the graphical notation of the elements of these packages. Figure 5-12 relates the L2 modelling level with the specification of the KDM\_CODE DSL.

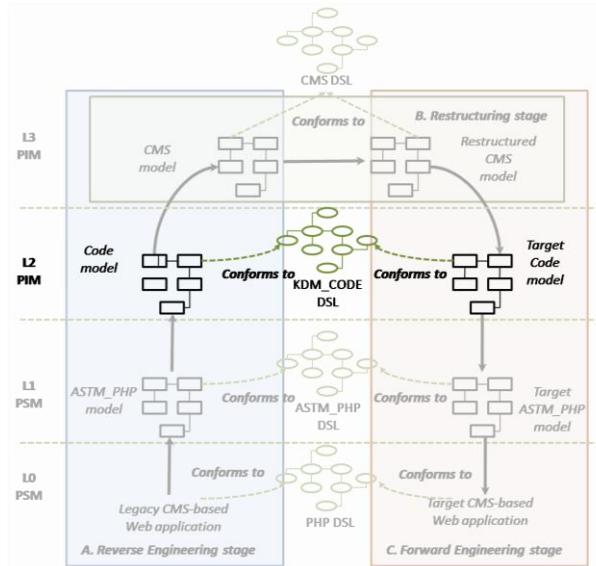
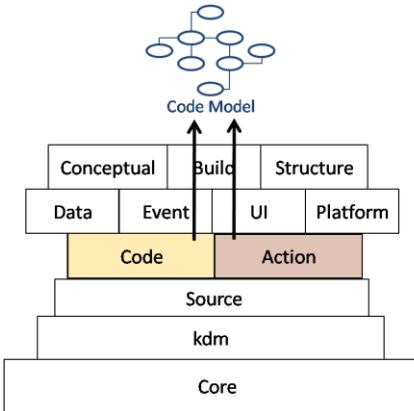


Figure 5-12. L2 modelling level.

#### 5.2.3.1 The KDM Metamodel

We present the elements gathered in the *code* and *action* packages of the KDM metamodel. These two packages are used to define the abstract syntax of the

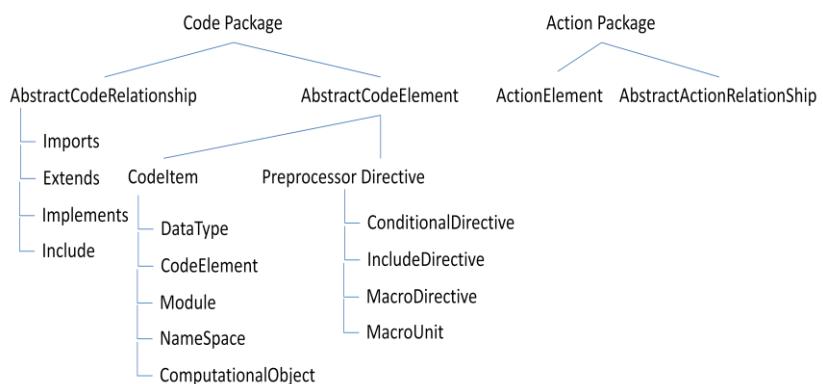
KDM\_CODE DSL that allows defining the Code models at L2 modelling level of the *ADMigraCMS* method. Figure 5-13 shows the structure in packages of the KDM metamodel.



**Figure 5-13. Code and action packages of the KDM metamodel.**

KDM is a standard metamodel proposed by ADM to define PIM models which represent the syntax and semantics of a system. Its *code* package is used to represent the structure of the source code by using code elements at a logical level, whereas the *action* package is composed of a set of elements that allow the representation of behaviour descriptions and control-and-data-flow relationships between the previous code elements.

Figure 5-14 shows the hierarchy of the main elements of the *code* package and *action* package. Below, we explain these main elements to understand better the mappings we have defined later.



**Figure 5-14. Hierarchy of the elements of the code and action packages of the KDM.**

## Code Package

Two of the core elements of the *code* package are *AbstractCodeElement* and *AbstractCodeRelationship* marked in Figure 5-15. In the following, we explain these two elements and the *CodeItem* element that is derived from *AbstractCodeElement*.

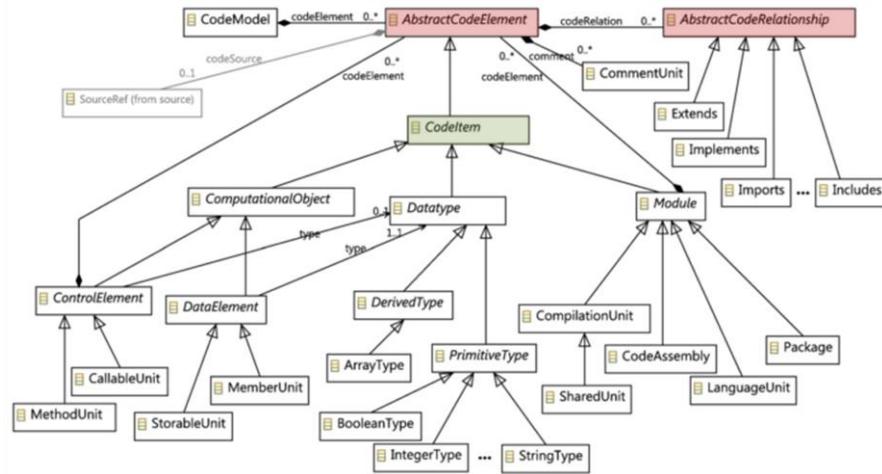


Figure 5-15. Elements of the code package of the KDM.

- **AbstractCodeElement Element:** This element represents all those code elements from the code of the legacy system. It is an element specialized in three elements: *CodeItem*, *Pre-processorDirective* and *ActionElement*.
  - **CodeItem Element:** It is an element representing the named elements determined by the programming language. It is specialized into five elements: the *DataType*, *CodeElement*, *Module*, *NameSpace* and *ComputationalObject*. 1) *DataType* element, defines the data type of the items of legacy systems such as variables or formal parameters in callable units. It is specialized into several elements representing different types such as, *PrimitiveType* or *TemplatType*. 2) *CodeElement* element, it is considered one of the KDM extension points and it is used to define new virtual elements within the KDM metamodel. 3) *Module* element, represents a discrete and identifiable program unit that contains other program elements and may be used as a logical component of the software system. 4) *NameSpace* element represents a group of code

elements. A *NameSpace* can be owned by a *Module* element representing a unit of visibility and 5) *ComputationalObject* element, represents the callable computational objects and the main data elements of the code. This element is specialized into two elements: the *ControlElement* element and the *DataElement* element. The former represents the callable objects such as *CallableUnits* and *MethodUnits* and the later represents the data elements such as *StorableUnits*, *ItemUnits*, *IndexUnits*, *MemberUnit*, *ParameterUnit* and *ValueElement*.

- **AbstractCodeRelationship Element:** It is an element that allows representing the structural relationships between the code elements. This element is specialized in other elements such as *Imports*, *Extends*, *Implements*, *Include* among others.

### Action Package

As we can see in Figure 5-16, two of the core elements of the *action* package are *ActionElement* and *AbstractActionRelationship* elements. The elements of this package represent behaviour descriptions in the Code model. Below, we describe the *ActionElement* and *AbstractActionRelationship* elements.

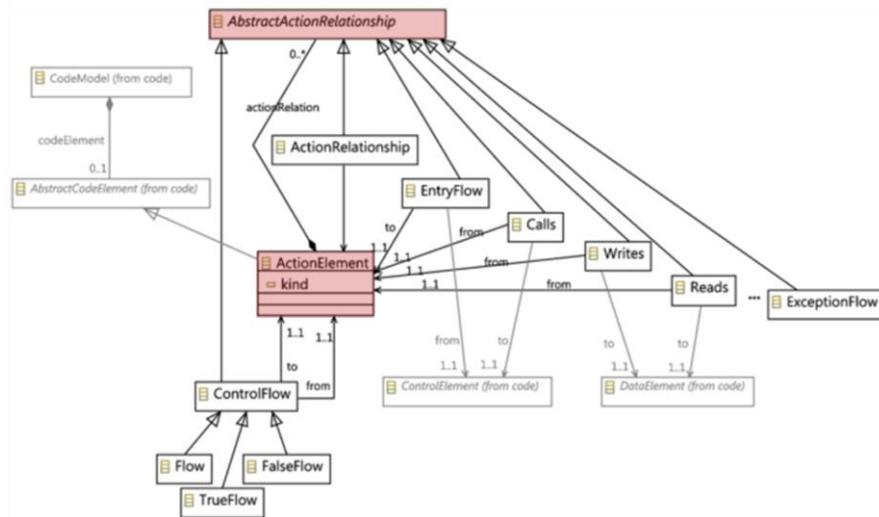


Figure 5-16. Elements of the action package of the KDM.

- **ActionElement Element:** This element depicts a basic unit of behaviour and represents some programming language constructs, such as statements, operators and conditions. Moreover, it is defined with the attribute *kind* that

establishes the semantics of the action. These semantics are defined according to a finite set of action types in the Micro KDM package which is explained later.

- **AbstractActionRelationship Element:** This element of the action package specifies various relationships from an *ActionElement*. It is specialized into other elements such as *Reads*, *Calls* or *Writes*.

### **Micro KDM Package**

During the explanation of the *ActionElement* element, we mentioned the *Micro KDM* package. This package defines a set of compliance rules, called micro actions, which are used as semantics constraints since they define the granularity and meaning of the *ActionElements*. These semantics are assigned by using their attribute *kind*.

Each micro action is depicted by four parts (ISO/IEC, 2012): 1) **the action kind**, specifying the nature of the operation performed; 2) **outputs**, representing the outgoing *Writes* relationships, which represent the result of the action; 3) **inputs**, defining the outgoing *Reads* relationships, which represent the arguments of the micro action; and finally 4) **the control part**, defining the outgoing *Control Flow* relationships.

In the following, we present some of these micro actions. For the presentation of each micro action, we use a table. The first column called *action kind* defines the name of the micro action which, in turn, it is the value assigned to the attribute *kind* of the *ActionElement*, the second column defines the *inputs*, the third column the *outputs* and, finally, the fourth one the *control* part. The column called *KDM representation* shows graphically the specification of the micro action. The complete catalogue of micro actions is presented in the standard specification (ISO/IEC, 2012).

#### **Micro Action Assign**

This micro action represents the copy of a value to a variable. Table 5-14, shows the features of this micro action.

**Table 5-14. Definition of the micro action Assign.**

<b>ACTION KIND</b>	<b>INPUT</b>	<b>OUTPUT</b>	<b>CONTROL</b>
Assign	A <i>Reads</i> relationship to a <i>DataElement</i> representing the <i>Value</i>	A <i>Writes</i> relationship to a <i>StorableUnit</i> to which the value of the input is assigned	A <i>Flow</i> relationship to the next micro action (optional).

**KDM REPRESENTATION**

```

classDiagram
    class Value
    class StorableUnit
    class ActionElement {
        <<kind : assign>>
    }
    class ActionElement {
        <<kind : flow>>
    }

    Value "1" --> "1" ActionElement : actionRelation
    StorableUnit "1" --> "1" ActionElement : actionRelation
    ActionElement "1" --> "1" ActionElement : actionRelation
  
```

**Micro Action Condition**

This micro action represents the definition of a simple condition where a value is evaluated to a Boolean result. Depending on this result, the program execution takes one flow or another. Table 5-15 shows the definition of this micro action.

**Table 5-15. Definition of the Micro action Condition.**

<b>ACTION KIND</b>	<b>INPUT</b>	<b>OUTPUT</b>	<b>CONTROL</b>
Condition	A <i>Reads</i> relationship to a <i>DataElement</i> representing the <i>Value</i>	none	A <i>TrueFlow</i> and a <i>FalseFlow</i> relationships, depending on the value evaluation

**KDM REPRESENTATION**

```

classDiagram
    class StorableUnit
    class ActionElement {
        <<kind : condition>>
    }
    class TrueFlow
    class FalseFlow

    StorableUnit "1" --> "1" ActionElement : Reads
    ActionElement "1" --> "1" TrueFlow : actionRelation
    ActionElement "1" --> "1" FalseFlow : actionRelation
  
```

### **Micro Action Call**

This micro action represents a simple function call which de value returned is assigned to a variable. Table 5-16 shows the definition of this micro action.

**Table 5-16. Definition of the micro action Call.**

<i>ACTION KIND</i>	<i>INPUT</i>	<i>OUTPUT</i>	<i>CONTROL</i>
Call	Zero or more <i>Reads</i> relationships to a <i>DataElement</i> representing the parameters	A <i>Writes</i> relationship to a <i>StorableUnit</i> to which the value of the input is assigned (optional)	A <i>Calls</i> relationship to the function represented as <i>MethodUnit</i>

**KDM REPRESENTATION**

```

classDiagram
    class DataElement
    class StorableUnit
    class MethodUnit
    class ActionElement {
        kind : call
    }
    class Reads
    class Writes
    class Calls

    ActionElement "0..*" --> "1" Reads : actionRelation
    ActionElement "0..*" --> "1" Writes : actionRelation
    ActionElement "0..*" --> "1" Calls : actionRelation
    Reads "1" --> "1" DataElement : to
    Writes "1" --> "1" StorableUnit : to
    Calls "1" --> "1" MethodUnit : to
  
```

### **Micro Action MethodCall**

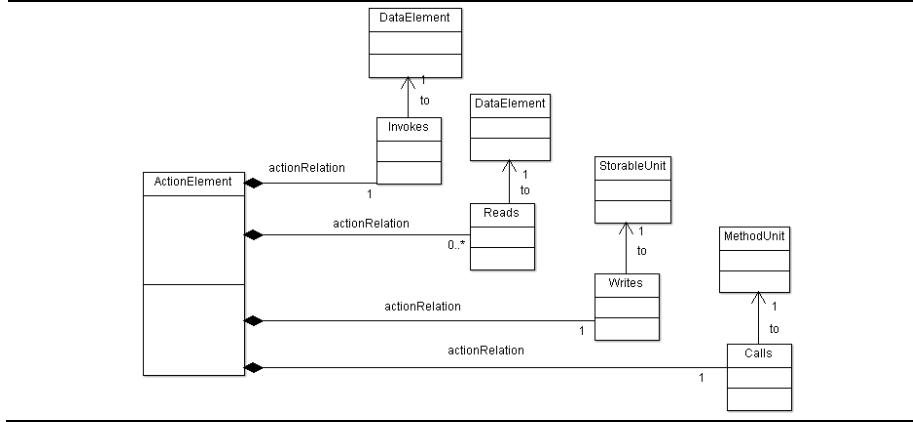
This micro action is similar to the previous *Calls* micro action, but this function call can be executed from an instance (a class or an object). Table 5-17 shows the definition of this micro action.

**Table 5-17. Definition of the micro action MethodCall.**

<i>ACTION KIND</i>	<i>INPUT</i>	<i>OUTPUT</i>	<i>CONTROL</i>
MethodCall	An <i>Invokes</i> relationship to <i>DataElement</i> which represents an instance. Zero or more <i>Reads</i> relationships to a <i>DataElement</i> which represent parameters.	A <i>Writes</i> relationship to a <i>StorableUnit</i> to which the value of the input is assigned (optional)	A <i>Calls</i> relationship to the function represented as <i>MethodUnit</i>

**KDM REPRESENTATION**

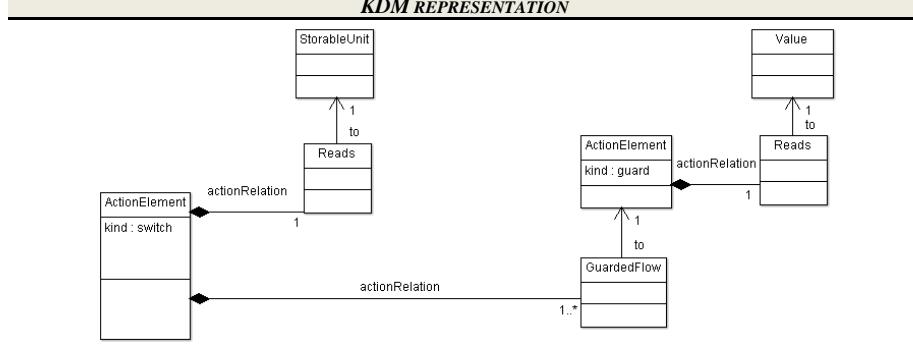


### Micro Action Switch

This micro action represents a branching based on the value of a *StorableUnit*. Depending on this value, the program execution takes one branch or another. Table 5-18 shows the definition of this micro action.

**Table 5-18. Definition of the micro action Switch.**

ACTION KIND	INPUT	OUTPUT	CONTROL
Switch	A <i>Reads</i> relationship to a <i>DataElement</i> representing the <i>Value</i>	none	One or more <i>GuardedFlow</i> relationships that represent the guard. The guard value is represented by the micro action Guard



### Micro Action Compound

This micro action represents a set of micro actions which are executed as a block. The specification of this micro action seems no very interesting, but we

have decided to explain it because it has been pretty used in the generation of the Code Model. Table 5-27 shows the definition of this micro action.

**Table 5-19. Definition of the micro action Compound.**

ACTION KIND	INPUT	OUTPUT	CONTROL
Compound	none	none	A Flow relationship to the first internal action element
<b>KDM REPRESENTATION</b>			
ActionElement kind : compound		ActionElement Flow	actionRelation 1 to 1

```

classDiagram
    class ActionElement {
        kind
    }
    class Flow {
        1 to 1
    }
    ActionElement "0..1" --> "1..1" ActionElement : actionRelation
    ActionElement "kind : compound"
    ActionElement "Flow"
  
```

### 5.2.3.2 Customized Graphical Notation

As occurred with the ASTM\_PHP metamodel, after presenting the abstract syntax of the KDM\_CODE DSL, we present the concrete syntax of this DSL. We define the graphical notations of the elements of the code and action packages.

Table 5-20 shows the graphical notation of some of the elements appearing during the validation in Chapter 7.

**Table 5-20. Customized graphical notation for the KDM\_CODE DSL.**

KDM ELEMENT	GRAPHICAL NOTATION	DESCRIPTION
CodeModel	Folder icon	The <i>CodeModel</i> is represented by a box simulating a container with all the elements conforming the Code model.
CompilationUnit	Document icon	The <i>CompilationUnit</i> is represented by a document containing the code.
MethodUnit	Black box icon	The <i>MethodUnit</i> is represented is represted by a black box containing all the elements of the method unit
SourceRef	Upward arrow icon	The <i>SourceRef</i> is represented by an arrow that specifies a reference to a concrete fragment of the code.
StorableUnit	Blue box icon	The <i>StorableUnit</i> is represented by a box containg an object which refers to a value.

KDM ELEMENT	GRAPHICAL NOTATION	DESCRIPTION
ActionElement		The <i>ActionElement</i> is represented by moving blades simulating an action.
Calls		The <i>Calls</i> is represented by an arrow pointing to a method unit.
Reads		The <i>Reads</i> is represented by a hand lens over a document simulating the reading action.
Writes		The <i>Writes</i> is represented by a pencil over a document simulating the writing action.
Creates		The <i>Creates</i> is represented by a toolkit simulating the creating action.
Value		The <i>Value</i> is represented by a cube representing a value.
Addresses		The <i>Addresses</i> is represented by a hand pointing to an object.
Flow		The <i>Flow</i> is represented by a pair of flows simulating the control is flowing.
Signature		The <i>Signature</i> is represented by a pen writing as it was signing.

As occurred previously, these graphical notations are considered during the implementation of the tree-like graphical editor, in Chapter 6, which allows the representation of Code models conforming to the KDM\_CODE DSL.

#### 5.2.4 The CMS DSL

Now, we focus on the specification of the CMS DSL. In this section, we present the concrete syntax since the abstract syntax has been presented in Chapter 4 with the definition of the CMS Common Metamodel. Figure 5-17 relates the CMS DSL with the L3 modelling level of the *ADMigraCMS* method.

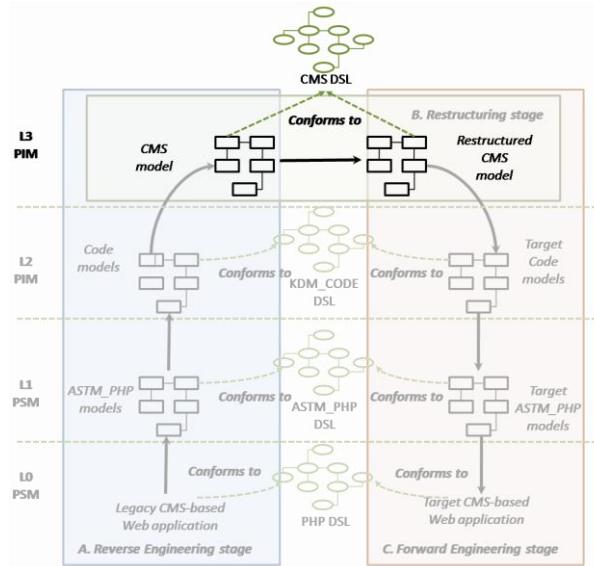


Figure 5-17. L3 modelling level.

#### 5.2.4.1 Customized Graphical Notation

Table 5-21 shows the graphical notations of some of the CMS elements appearing during the validation in Chapter 7. In Appendix D, we present the complete definition of the concrete syntax of the CMS DSL.

Table 5-21. Customized graphical notation for the CMS DSL.

CMS ELEMENT	GRAPHICAL NOTATION	DESCRIPTION
Menu		The <i>Menu</i> is presented by a grey rectangle with the stereotype <Menu>
MenuItem		The <i>MenuItem</i> is presented by blue rectangle with the stereotype <MenuItem>
Page		The <i>Page</i> is presented by yellow rectangle with the stereotype <Page>
MediaGalleryContent		The <i>MediaGalleryContent</i> is presented by set of media elements.

CMS ELEMENT	GRAPHICAL NOTATION	DESCRIPTION
ListContent		The <i>ListContent</i> is presented by document presenting a list.
MediaContent		The <i>MediaContent</i> is presented by a film reel.
TextContent		The <i>TextContent</i> is presented by document containing a text.

As occurred with the previous graphical notations, this one is considered during the implementation of the UML-like graphical editor.

After explaining the DSLs considered within the *ADMigraCMS* method, we present in the following sections the mappings defined between the different modelling levels. For the explanations of these mappings, we have defined a set of tables. These tables show the source element of the mapping, the target element and the conditions that the source element has to meet to be transformed.

### 5.3 Specification of the Automated Transformations

In this section, we explain the specification of the automated transformations between the different modelling levels defined in the *ADMigraCMS* method.

This specification has been defined at metamodel level. We have determined a set of mappings that relate the elements of a source metamodel (at a modelling level) to the elements of a target metamodel (at a different modelling level). The same mappings can be considered for the implementation of the transformations between two modelling levels in the reverse engineering stage as well as in the forward engineering stage of the *ADMigraCMS* method.

To explain the specification of these mappings, we have used a set of tables whose first and second columns represent the name and the attributes of the elements of a source metamodel, whereas the third and fourth columns represent the name and the attributes of the elements of a target metamodel. Finally, the fifth column represents the conditions that the source elements must meet to be transformed into target elements.

### 5.3.1 Between PHP Metamodel and ASTM\_PHP Metamodel

We present the mappings between the elements of the PHP metamodel and the elements of the ASTM\_PHP metamodel. As we explained previously, the definition of these mappings cover the L0-to-L1 T2M transformations and the L1-to-L0 M2T transformations.

Firstly, we explain the mappings of the PHP elements that represent *PHP Expressions* and then we present those ones representing *PHP Statements*.

#### PHP Expressions

Before presenting the mappings between the *PHP Expressions* and the elements of the ASTM\_PHP metamodel, we present the mappings from the software assets of the legacy CMS-based Web application that create the required elements *Project* and *CompilationUnitPHP* which are the root elements of the ASTM\_PHP model. To present these mappings, we present Table 5-22 whose first column shows the software asset and the second column presents the ASTM\_PHP element which is related to.

**Table 5-22. Mappings from software assets to root elements in the ASTM\_PHP metamodel.**

SOFTWARE ASSET <i>Element</i>	ASTM_PHP METAMODEL <i>Element</i>		CONDITION
	<i>Element</i>	<i>Properties &amp; value</i>	
Folder	Project	files	No condition
PHP file	Compilation UnitPHP	statements	No condition

The folder that contains the files that implement the CMS-based Web application is mapped to the element *Project*. All the files contained in this folder are stored in its attribute *files*. The PHP files that contains the PHP code intended to be migrated is mapped to the element *CompilationUnitPHP*. All the PHP Expressions and Sentences are stored in its attribute *statements*.

In the following, we explain the mappings between the *simple expressions* in PHP and the elements of the ASTM\_PHP metamodel.

#### Simple Expressions

Table 5-23 presents these mappings between the PHP elements that represent *simple expressions* and the ASTM\_PHP elements.

**Table 5-23.** Mappings from simple expression to ASTM\_PHP elements.

PHP METAMODEL		ASTM_PHP METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Variable	value	Identifier Reference	name = value refersTo = Variable definition (element)	If Variable represents a storable unit
		Identifier Reference	name = value refersTo = Function definition (element)	If Variable represents a method unit
		ByValueActualParameterExpression	value = Identifier Reference (element)	If Variable represents a parameter in a function call
Name	Name Identifier	Name	value = nameIdentifier	No condition
NumberLiteral StringLiteral FloatLiteral HexadecimalLiteral OctalLiteral BooleanLiteral	value	Literal	value = value	No condition
ArrayType	params	Collection Expression	expressionList = params	No condition
ArrayEntry	key value	DuplaArray	index = key value = value	No condition

- **Mapping from the element Variable to the element IdentifierReference element:** If the element *Variable* represents a storable unit, it is mapped to the element *IdentifierReference* whose attribute *refersTo* points to a *VariableDefinition*. Otherwise, if it represents a method unit the attribute *refersTo* points to a *FunctionDefinition*. In both cases, the attribute name of the element *IdentifierReference* takes the value from the attribute *value*.
- **Mapping from the element Variable to the element ByValueActualParameterExpression:** if the element *Variable* represents a parameter in a function call, it is mapped to the element *ByValueActualParameterExpression* whose attribute *value* points to an *IdentifierReference*.
- **Mapping from the element Name to the element Name:** the element *Name* is mapped to the element *Name*. Its attribute *value* takes the value from the attribute *nameIdentifier*.
- **Mapping from the element Literal elements to the element Literal:** The elements representing *Literals* e.g. *NumberLiteral* or *StringLiteral* are mapped to the element *Literal* of the ASTM\_PHP metamodel. The attribute *value* of the elements *Literal* takes the value from the attribute *value*.

- **Mapping from the element *ArrayType* to the element *CollectionExpression*:** The *ArrayType* element is mapped to the *CollectionExpression* element. This element defines the attribute *expressionList* which stores the expressions used to define the positions in the array and their values at the time of creating the array. The *expressionList* attribute takes the value from the attribute *params* of the *ArrayType* element.
- **Mapping from the element *ArrayEntry* to the element *DuplaArray*:** The *ArrayEntry* element is mapped to the *DuplaArray* element. This element is composed of two attributes, *index* and *value*. The values of these two attributes are assigned from the attributes *key* and *value* of the *ArrayEntry* element.

### Function Call Expressions and Array Access Expressions

Table 5-24 shows the mappings between the elements *functionCallExpressions* and *arrayAccesses* of the PHP metamodel to ASTM\_PHP elements. In Appendix E, we show the mapping for the *ObjectAccess* element.

**Table 5-24. Mappings from PHP expressions to ASTM\_PHP elements.**

<i>PHP METAMODEL</i>		<i>ASTM_PHP METAMODEL</i>		<i>CONDITION</i>
<i>Element</i>	<i>Properties</i>	<i>Element</i>	<i>Properties &amp; value</i>	
<b>FunctionCall Expression</b>	functionIdentifier params	<b>Function Call Expression</b>	calledFunction = functionIdentifier actualParams = ByValueActualParameter Expression	No condition
<b>ArrayAccess</b>	arrayIdentifier sus	<b>ArrayAccess PHP</b>	arrayName = arrayIdentifier subscripts = sus	No condition
<b>Subscript</b>	expression	<b>Expression (element)</b>	--	The <i>Expression</i> mapped depends on the value in the <i>expression</i> attribute.

- **Mapping from the element *FunctionCallExpression* to the element *FunctionCallExpression*:** The *FunctionCallExpression* element is mapped to the element *FunctionCallExpression*. The attributes of this element are *calledFunction* and *actualParams*. The attribute *calledFunction* takes its value from the attribute *functionIdentifier*, whereas the attribute *actualParams* takes the expressions from the attribute *params*. The expressions defined within the attribute *params* are mapped to the element *ByValueActualParameterExpression* which is defined with an attribute called *value* which stores the expression.

- **Mapping from the element ArrayAccess to the element ArrayAccessPHP:** The *ArrayAccess* element is mapped to the element *ArrayAccessPHP*. This element is composed of two attributes *arrayName* and *subscripts*. The attribute *arrayName* takes the name from the attribute *arrayIdentifier*, whereas the attribute *subscripts* takes the value from the attribute *sus*.
- **Mapping from the element Subscript to the element Expression:** The element *Subscripts* has an attribute called *expression*. Depending on the value of this attribute, it is mapped to one of the elements in the PHP metamodel representing expressions.

### Unary Expressions

Table 5-25 presents some of the mappings defined between the PHP elements that represent *unary expressions* and the ASTM\_PHP elements. In Appendix E, the rest of mappings for these expressions are presented.

**Table 5-25. Mappings from unary expressions to ASTM\_PHP elements.**

PHP METAMODEL		ASTM_PHP METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
<b>NegateOrCast</b>	operator right cast	<b>Cast Expression</b>	expression = right castType = cast	If the attribute <i>cast</i> has a value
		<b>Unary Expression</b>	operand = right operator = operator	If <i>operator</i> is – it is mapped to <i>UnaryMinus</i> element. If operator is ~ it is mapped to <i>BitNot</i> element.
<b>LogicalNot</b>	expr	<b>Unary Expression</b>	operand = expr operator = <i>Not</i> ( <i>element</i> )	No condition
<b>Reference</b>	value_ref	<b>Unary Expression</b>	operand = value_ref operator = <i>Deref</i> ( <i>element</i> )	No condition

- **Mapping from the element NegateOrCast to the element CastExpression:** If the attribute *cast* of the element *NegateOrCast* contains a value, then it is mapped to an element *CastExpression*. The element *CastExpression* is composed of two attributes *expression* and *castType*. The former stores the expression of the unary expression and its value is taken from the attribute *right* of the *NegateOrCast* element. The later represents the casting operator applied to the expression and its value got from the attribute *cast*.
- **Mapping from the element NegateOrCast to the element UnaryExpression:** If the attribute *cast* does not contain any value, it is mapped to the element

*UnaryExpression*. The *UnaryExpression* element is composed of two attributes *operand* and *operator*. The former takes its value from the attribute *right* and the later from the attribute *operator*. The value of this last attribute can be mapped to two different elements, the *UnaryMinus* element or to the *BitNot* element. If the attribute *operator* contains a ‘-’ it is mapped to the *UnaryMinus* element, otherwise if it contains a ‘~’ it is mapped to the *BitNot* element.

- **Mapping from the element *LogicalNot* to the element *UnaryExpression*:** The element *LogicalNot* is mapped to the *UnaryExpression* element. The attribute *operand* of this element takes the value from the attribute *expr* of the *LogicalNot* element and the attribute *operator* contains an instance of the *Not* element of the ASTM\_PHP metamodel.
- **Mapping from the element *Reference* to the element *UnaryExpression*:** The element *Reference* is mapped to the *UnaryExpression* element. The attribute *operand* of this element takes the value from the attribute *value\_ref* of the element *Reference* and the attribute *operator* contains an instance of the element *Deref* element of the ASTM\_PHP metamodel.

### Binary Expressions

In the following, we explain the mappings between the PHP elements representing *binary expressions* and the ASTM\_PHP elements. In Table 5-26, we present a set of these mappings, the rest of them can be consulted in Appendix E.

**Table 5-26. Mappings from binary expressions to ASTM\_PHP elements.**

PHP METAMODEL		ASTM_PHP METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
<b>Assignment</b>	left operator right	<b>Binary Expression</b>	left = left operator = operator right = right	If <i>operator</i> is '=' or '+=' or '*=' is mapped to an <i>Assign</i> element. If <i>operator</i> is combined (+=, *=) the <i>right</i> attribute stores a <i>BinaryExpression</i> .
<b>Addition</b>	left operator right	<b>Binary Expression</b>	left = left operator = operator right = right	If <i>operator</i> is '+' is mapped to an <i>Add</i> element. If <i>operator</i> is '-' is mapped to <i>Subtract</i> element.
<b>Equality Check</b>	left operator right	<b>Binary Expression</b>	left = left operator = operator right = right	If the value of <i>operator</i> is '==' is mapped to <i>Equal</i> element. If the value of <i>operator</i> is '!= is mapped to <i>NotEqual</i> . If the value of <i>operator</i> is '===' is mapped to <i>Identical</i> . If the value of <i>operator</i> is '!==' is mapped to <i>NotIdentical</i>

<i>PHP METAMODEL</i>		<i>ASTM_PHP METAMODEL</i>		<i>CONDITION</i>
<i>Element</i>	<i>Properties</i>	<i>Element</i>	<i>Properties &amp; value</i>	
				element.
<b>BiteWise Shift</b>	left operator right	<b>Binary Expression</b>	left = left operator = operator right = right	If <i>operator</i> is ‘>>’ is mapped to <i>BitRightShift</i> element If <i>operator</i> is ‘<<’ is mapped to <i>BitLeftShift</i>
<b>Logical And</b>	left right	<b>Binary Expression</b>	left = left operator = <i>And</i> (element) right = right	No condition
<b>InstanceOf</b>	left right	<b>Binary Expression</b>	left = left operator = <i>InstanceOf</i> element right = right	No condition

- **Mapping from the element Assignment to the element BinaryExpression:** The element *Assignment* is mapped to the *BinaryExpression* element. It is compound of three attributes: *left*, *right* and *operator*. The attribute *left* of this element takes the value from the attribute *left* of the *Assignment* element and the attribute *right* takes the value of the attribute *right*. The attribute *operator* can be mapped to two different elements depending on the type of the operator. If it is a simple assign represented by ‘=’ it is mapped to the *Assign* element, otherwise if it is a combined assignment, such as ‘+=’ or ‘\*=-’, it is mapped to the *OperatorAssign* element. The operator which goes along the assignment is stored in the attribute *operator* of this last element.
- **Mapping from the element Addition to the element BinaryExpression:** The element *Addition* mapped to the *BinaryExpression* element. As occurred with the *Assignmet* element, the attribute *left* of the *BinaryExpression* takes the value from the attribute *left* of the *Addition* element and the attribute *right* takes the value of the attribute *right*. The attribute *operator* can be mapped to two different elements depending on the type of the operator. If it is represented by ‘+’ it is mapped to the *Add* element, otherwise if it is a ‘-’ it is mapped to the element *Subtract*.
- **Mapping from the element EqualityCheck to the element BinaryExpression:** The element *EqualityCheck* is mapped to the element *BinaryExpression*. Its definition is very similar to the previous elements. The attribute *operator* of the *BinaryExpression* can be mapped to three different elements depending on the type of the operator. If it is represented by a ‘==’ it is mapped to the *Equal* element and if it is a ‘!=’ it is mapped to the *Notequal* element, both elements of the GASTM. Otherwise, if it is a ‘==='

it is mapped to the element *Identica*, and finally, if it is a ‘!==’ it is mapped to the *NotIdentical* element.

- **Mapping from the element BitWiseShift to the element BinaryExpression:** The element *BitWiseShift* element is mapped to the *BinaryExpression* element. Its attribute *operator* can be mapped to two different elements depending on the type of the operator. If it is represented by a ‘>>’ it is mapped to the *BitRightShift* element, if it is a ‘<<’ it is mapped to the *BitLeftShift* element.
- **Mapping from the elements LogicalAnd to the element BinaryExpression:** This element is mapped to the *BinaryExpression* element. The attribute *operator* in both cases is mapped to the *And* element of the ASTM\_PHP metamodel.
- **Mapping from the element InstanceOf to the element BinaryExpression:** The element *InstanceOf* is mapped to the *BinaryExpression* element. The attribute *operator* is mapped to the *InstanceOf* element of the ASTM\_PHP metamodel.

## PHP Statements

In this section, we present the mappings of those elements that represent PHP Statements in the PHP metamodel.

### Definition Statements

In the following, we present the mapping established between the PHP elements representing *definition statements* and the elements of ASTM\_PHP metamodel. Table 5-27 shows the mappings from the PHP elements which represent definitions such as *variable definition*, *function definition* and *class definition*. The rest of these mappings can be consulted in Appendix E.

Table 5-27. Mappings from definition statements to ASTM\_PHP elements.

PHP METAMODEL		ASTM_PHP METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Variable Def Statement	visibility left right	DefinitionOr Declaration Statement	declOrDefn = Variable Definition	If the attribute <i>visibility</i> is defined it is mapped to a <i>AccessKind</i> element
		Variable Definition	accessKind = visibility identifierName = left initialValue = right	
		Expression Statement	expression = <i>BinaryExpression</i> (element)	If the <i>left</i> attribute is defined as an <i>arrayAccess</i>

PHP METAMODEL		ASTM PHP METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Function Def Statement	visibility functionIdentifier params body	DefinitionOrDeclarationStatement	declOrDefn = Function Definition	If the attribute <i>params</i> is defined it is mapped to a <i>Func Parameters</i> element
		Function Definition	accessKind = visibility identifierName = functionIdentifier formalParameters = params body = body	
Func Parameters	var value	Formal Parameter Definition	identifierName = var initialValue = value	No condition

- *Mapping from the element VariableDefStatement to the elements DefinitionOrDeclarationStatement and VariableDefinition:* The *VariableDefStatement* element is mapped to two elements, a *DefinitionOrDeclarationStatement* element whose attribute *declOrDefn* stores a *VariableDefinition* element. The attribute *accesskind* of this element takes the value from the attribute *visibility* (public, protected and private). The attribute *identifierName* of the *VariableDefinition* element takes the value from the attribute *left* of the *VariableDefStatement* element; and finally the attribute *initialValue* takes the value from the attribute *right*.
- *Mapping from the element VariableDefStatement to the element ExpressionStatement:* If the attribute *left* of the element *VariableDefStatement* is defined as an *ArrayAccess*, it is mapped to an *ExpressionStatement* element whose attribute *expression* stores a *BinaryExpression* element.
- *Mapping from the element FunctionDefStatement to the elements DefinitionOrDeclarationStatement and FunctionDefinition:* The *FunctionDefStatement* element is mapped to two elements, a *DefinitionOrDeclarationStatement* element whose attribute *declOrDefn* stores a *FunctionDefinition* element. The attribute *accesskind* of the *FunctionDefinition* takes the value from the attribute *visibility*. Otherwise, the attribute *identifierName* of the *FunctionDefinition* element takes the value from the attribute *functionIdentifier* and the attribute *formalParameters* takes the value from the attribute *params*. The attribute *params* stores a set of instances of the *FuncParameters* element that also needs to be mapped. The mapping of the *FuncParameters* element is explained below. Finally, the

attribute *body* takes the value from the attribute *value* of the *FunctionDefStatement* element.

- **Mapping from the element *FuncParameters* to the element *FormalParameterDefinition*:** The *FuncParameters* element is mapped to the *FormalParameter* element. The attribute *identifierName* of this element takes the value from the attribute *var* and the attribute *initialValue* takes the value from the attribute *value*.

### Expression Statements

Below, the Table 5-28 shows the mappings from the PHP elements that represent *expression statements* such as *subroutines* and *increment statements* to elements of the ASTM\_PHP metamodel. The rest of these mappings can be consulted in Appendix E.

**Table 5-28. Mappings from expression statements to ASTM\_PHP elements.**

<i>PHP METAMODEL</i>		<i>ASTM_PHP METAMODEL</i>		<i>CONDITION</i>
<i>Element</i>	<i>Properties</i>	<i>Element</i>	<i>Properties &amp; value</i>	
<b>FunctionCall StatementEx p</b>	fun op exp	<b>Expression Statement</b>	expression = <i>FunctionCall Expression</i> (element)	No condition
<b>Object Access Statement</b>	obj right left	<b>Expression Statement</b>	expression = <i>ObjectAccess or ClassAccess</i> (element)	The expression mapped depends on the value in the <i>obj</i>
<b>Member</b>	Left Obj right	<b>Expression</b>	--	The <i>Expression</i> mapped depends on the value of the attribute <i>right</i>
<b>Increment Statement</b>	operator right left	<b>Expression Statement</b>	expression = <i>UnaryExpression</i> (element)	The <i>Unary Expression</i> element depends on the <i>operator</i> , <i>right</i> and <i>left</i> attributes.

- **Mapping from the element *FunctionCallStatementExp* to the element *ExpressionStatement*:** The *FunctionCallStatement* element is mapped to the element *ExpressionStatement* whose attribute *expression* stores the element *FunctionCallExpression*.
- **Mapping from the element *ObjectAccessStatement* to the element *ExpressionStatement*:** The element *ObjectAccessStatement* is also mapped to the *ExpressionStatement*. In this case, within the attribute *expression* we store the element *ObjectAccess* or *ClassAccess*.
- **Mapping from the element *Member* to the element *Expression*:** Depending on the attribute *right* of the element *Member*, this element is mapped to one

or another element representing an *Expression* within the ASTM\_PHP metamodel.

- **Mapping from the element *IncrementStatement* to the element *ExpressionStatement*:** The *IncrementStatement* element is also mapped to the *ExpressionStatement*. Its attribute *expression* stores an instance of the *UnaryExpression* element. To define the instance of this element we use the values stored within the attributes *left*, *right* and *operator* of the *IncrementStatement* element.

### Control Statements

Finally, Table 5-29 presents the mappings established between the PHP elements representing the *control statements* such as *ifStatements*, *forStatements* or *returnStatements* among others and the ASTM\_PHP elements. The rest of these mappings can be consulted in Appendix E.

**Table 5-29. Mappings from control statements to ASTM\_PHP elements.**

<i>PHP METAMODEL</i>		<i>ASTM_PHP METAMODEL</i>		<i>CONDITION</i>
<i>Element</i>	<i>Properties</i>	<i>Element</i>	<i>Properties &amp; value</i>	
<b>IfStatement</b>	condIf ifStatements listElsif elseStatements	<b>IfStatement</b>	condition = condIf / condElsif thenBody = ifStatements / elsifStatements elseBody = elseStatements	No condition
<b>Elsif</b>	condElsif elsifStatements			
<b>For Statement</b>	exp1 exp2 exp3 statements	<b>For Statement PHP</b>	initBody = exp1 condition = exp2 iterationBody = exp3 body = statements	No condition
<b>Return Statement</b>	return	<b>Return Statement</b>	returnValue = return	No condition

- **Mapping from the element *IfStatement* element and *Elsif* to the element *IfStatement*:** These two elements are mapped to the *IfStatement* element. This element has three attributes: *condition*, *thenBody* and *elseBody*. The attribute *condition* takes the value from the *condIf* attribute from the *IfStatement* PHP element or from the *condElsif* in the case of the *Elsif* element. On the other hand, the attribute *thenBody* takes the values from the attribute *ifStatements* from the *IfStatement* PHP element or from the attribute *elsifStatements* from the *Elsif* element. Finally, the attribute *elsebody* store the statements from the attribute *elseStatements*.
- **Mapping from the element *ForStatement* to the element *ForStatementPHP*:** The *ForStatement* element is mapped to the element

*ForStatementPHP*. This element has four attributes: *initBody*, *condition*, *iterationBody* and *body*. The attribute *initBody* represents the expression performed at the beginning of the *loop* and its value is mapped from the attribute *exp1* of the *ForStatementPHP*. Otherwise, the attribute *condition* takes the value from the attribute *exp2*, whereas the attribute *iterationBody* takes the value from the attribute *exp3*. The attribute *body* stores the statements that conforms the body of the *loop*.

- **Mapping from the element *ReturnStatement* to the element *ReturnStatement*:** The element *ReturnStatement* is mapped to the *ReturnStatement* element of the ASTM\_PHP metamodel. This element is composed of one attribute called *returnValue* which represents the value returned which is an instance of the *Expression* element. The value of this attribute is taken from the attribute *return*.

### 5.3.2 Between ASTM\_PHP Metamodel and KDM Metamodel

In this section, we present the mappings between the elements of the ASTM\_PHP metamodel and the elements of the code and action packages of the KDM metamodel (KDM elements). As we explained previously, the definition of these mappings cover the L1-to-L2 M2M transformations and the L2-to-L1 M2M transformations.

#### PHP Expressions

In this section, we present the mappings from the elements of the ASTM\_PHP metamodel that represent *PHP Expressions* to the elements of the KDM metamodel.

Before presenting the mappings between the *PHP Expressions* and the elements of the KDM metamodel, we present the mappings from the root elements of the ASTM\_PHP metamodel to the root elements of the KDM metamodel. These mappings are presented in Table 5-32.

**Table 5-30. Mappings from the root elements of the ASTM\_PHP to the KDM metamodel.**

ASTM_PHP METAMODEL		KDM METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
<b>Project</b>	files	<b>CodeModel</b>	codeElement = files	No condition

ASTM_PHP METAMODEL		KDM METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Compilation UnitPHP	statements	Compilation Unit	codeElement = statements	No condition

The element *Project* of the ASTM\_PHP metamodel is mapped to the element *CodeModel* in KDM metamodel. The value stored in the attribute *files* is mapped to the attribute *codeElement*. Otherwise, the element *CompilationUnitPHP* is mapped to the element *CompilationUnit* in the KDM metamodel. Its attribute *statements* is mapped to the attribute *codeElement*.

### Simple Expressions

In the following, we start with the mappings of the ASTM\_PHP elements that represent *simple expressions*. Table 5-31 gathers theses mappings.

Table 5-31. Mappings from simple expressions to KDM elements.

ASTM_PHP METAMODEL		KDM METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Identifier Reference	name refersTo locationInfo	Storable Unit	name = name kind = <i>StorableKind</i> sourceCode = locationInfo	No condition
Name	value	--	name = value	No condition
Literal	value	Value Element	name = value	No condition
Collection Expression	expressionList	Action Element	kind = <i>newArray</i> sourceCode = locationInfo actionRelation = <i>Creates, Writes and Flow</i>	If <i>expressionList</i> (DuplaArray) is null
		Action Element	kind = <i>compound</i> sourceCode = locationInfo codeElement = ActionElement (kind= <i>newArray</i> ) + expressionList	If <i>expressionList</i> (DuplaArray) is not null
Dupla Array	index value	Action Element	kind = <i>arrayReplace</i> sourceCode = locationInfo codeElement = index / value actionRelation = <i>Addresses, Reads, Writes, Flow</i>	No condition

- **Mapping from the element *IdentifierReference* to the element *StorableUnit*:** The element *IdentifierReference* is mapped to the element *StorableUnit*. This element represents variables. Its main attributes are *name*, *sourceCode*, *type* and *kind*. The attribute *name* takes the value from the value of the attribute *name* of the *IdentifierReference* element. The attribute *sourceCode* links the element *StorableUnit* with the PHP source code. The

value of this attributed is extracted from the *locationInfo* attribute of the element *IdentifierReference*. The attribute *type* is a required attribute that defines the type of the variable. Its value is extracted from the attribute *expressionType*. Finally, the attribute *kind* takes the value from the enumerated type *StorableKind* (*global*, *local*, *static*, *external*, *register* and *unknown*).

- **Mapping from the element *Name* to the attribute *Name*:** The element *Name* of the ASTM\_PHP metamodel is mapped to the attribute *name* of the *StorableUnit* and *ValueElement* elements. The value of this attribute is taken from the attribute *value*.
- **Mapping from the element *Literal* to the element *ValueElement*:** The element *Literal* is mapped to the element *ValueElement*. The element *Value* is associated with a single value of a primitive datatype such a *Boolean*, *string* or *integer*. The main attribute of this element is the attribute *name* which takes the value of attribute *value*.
- **Mapping from the element *CollectionExpression* to the element *ActionElement*:** If the attribute *expressionList* of the element *CollectionExpression* is empty, it is mapped to the element *ActionElement* whose attribute *kind* takes the value of *newArray*, according to the Micro KDM package. Otherwise, if the attribute *expressionList* contains a value, the *ActionElement* defined as *newArray* is contained within another *ActionElement* whose attribute *kind* is *compound*. In both cases, the *ActionElement* defined as *newArray* follows the Micro KDM specification. Thus, the *inputs* of this micro action are a *Creates* relationship to the *Datatype* being created (in this case an *ArrayType*) and a *Reads* relationship to the *DataElement* that represents the length of the new array. In PHP the definition of an array does not need the specification of its length so that the *Reads* relationship is not defined. The *output* of this micro action is a *Writes* relationship which represents the *DataElement* such as a *StorableUnit* where the new array is stored. As we explained previously, in PHP it is possible to define the values of the positions at the time of creating the new array. These values are represented by the *DuplaArray* element and are stored within the attribute *expressionList* of the *CollectionExpression* element. Below, we explain the mapping between the *DuplaArray* to an *ActionElement*.
- **Mapping from the element *DuplaArray* to the element *ActionElement*:** The element *DuplaArray* is mapped to the element *ActionElement*. The attribute

*kind* of this element takes the value of *arrayReplace*. Regarding the Micro KDM specification the *inputs* of this micro action are an *Addresses* relationship to a *DataElement* (an *ArrayType*), a *Reads* relationship that represents the index (the key in the *DuplaArray* element) and a *Reads* relationship to a *DataElement* representing the new value. As *outputs*, it considers a *Writes* relationship to an *ItemUnit* object which represents a position of the array. The values stored within the attributes *index* and *value* of the *DuplaArray* element can be mapped to *StorableUnits* or *ValueElements*. These elements are stored within the attribute *codeElement* of the *ActionElement* element.

### Function Call Expressions and Array Access Expressions

Table 5-32 presents the mappings from the ASTM\_PHP elements defining *function call* and *array accesses* to KDM elements. The rest of these mappings can be consulted in Appendix E.

**Table 5-32. Mappings from PHP expressions to KDM elements.**

ASTM_PHP METAMODEL		KDM METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
<b>FunctionCall Expression</b>	calledFunction actualParams	<b>Action Element</b>	name = calledFunction kind = <i>call</i> codeElement = actualParams actionRelation = <i>Calls, Reads</i>	If the attribute <i>name</i> is defined, a <i>Calls</i> element is created.
<b>ByValue Actual Parameter Expression</b>	value	<b>Storable Unit</b>	name = value	Depenging on the context it is mapped to a <i>StorableUnit</i> or to a <i>ValueElement</i>
		<b>Value Element</b>	name = value	
<b>ArrayAccess PHP</b>	arrayName subscripts	<b>Action Element</b>	name = arrayName kind= <i>arraySelect / arrayReplace</i> codeElement = subscripts actionRelation = <i>Addresses, Reads, Writes</i>	If the array access is on the right side of a Variable Definition

- **Mapping from the element FunctionCallExpression to the element ActionElement:** The element *FunctionCallExpression* is mapped to the element *ActionElement*. Its attribute *kind* is defined as a *call*. According to the specification of the Micro KDM package, the *inputs* of this micro action is defined with a set of *Reads* relationships to *DataElements* such as *StorableUnit*, *ParameterUnit* or *ValueElement* which represent the input parameters. These parameters are stored in its attribute *codeElement*. The *output* is an optional *Writes* relationship to the *DataElement* that stores the returned value. Moreover, for this micro action there is a control part

represented by a *Calls* relationship to the *ControlElement*, such as *CallableUnit* or *MethodUnit*, which represents the function. Moreover, the *ActionElement* element has an attribute called *name* which takes the value from the attribute *calledFunction* of the *FunctionCallExpression* element.

- ***Mapping from the element ByValueActualParameterExpression to the elements StorableUnit or ValueElement:*** The element *ByValueActualParameterExpression* can be mapped to a *StorableUnit* element or to a *ValueElement* element depending on the context. Their attribute *name* takes the value from the attribute *value*.
- ***Mapping from the element ArrayAccessPHP to the element ActionElement:*** The element *ArrayAccessPHP* is mapped to the *ActionElement* element. The attribute *kind* can be defined as an *arraySelect* or *arrayReplace*, both representing the access to a position of the array. According to the specification of the Micro KDM package, the *inputs* of this micro actions are an *Addressees* relationship to a *DataElement* (of an *ArrayType*) and two *Reads* relationships; one to the *ItemUnit* representing the element being accessed and the other one to the *IndexUnit* representing the position accessed of the array. The *output* is composed of an optional *Writes* relationship representing the *DataElement* to which the value of the *ItemUnit* is assigned. It is worth noting that if it is a multidimensional array access (`$var[3][2]`) we have to define each dimension with an *ActionElement* whose attribute *kind* is *ArrayAccess*. This set of *ActionElements* is grouped within a super *ActionElement* defined as *compound*. This *ActionElement* requires an *Entry* flow to the first internal action element. Its attribute *name* takes the value from the attribute *arrayName* of the *ArrayAccessPHP* element. Moreover, its attribute *codeElement* can store the objects from the attribute *subscripts* of the *ArrayAccessPHP* element. For instance, in this case `$var[3+2]` the attribute *codeElement* stores the addition.

### Unary Expressions

In the next section, we explain how we define the mappings between the ASTM\_PHP elements which represent the *unary expressions* and the KDM elements. Table 5-33 shows these mappings.

**Table 5-33.** Mappings from unary expressions to KDM elements.

ASTM_PHP METAMODEL		KDM METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
<b>Cast Expression</b>	expression castType	<b>Action Element</b>	kind = <i>TypeCast</i> <i>codeElement</i> = <i>expression, castType</i> <i>actionRelation</i> = <i>Reads, UsesType and Writes</i>	No condition
<b>Unary Expression</b>	operand operator	<b>Action Element</b>	<i>codeElement</i> = <i>operand</i> <i>kind</i> = <i>operator</i> <i>actionRelation</i> = <i>Reads, Writes</i>	The <i>kind</i> attribute depends on the operator: UnaryMinus → minus (incl) BitNot → bitNot Not → not Deref → deref (incl) New → new Clone → clone (incl) Increment → preIncr (incl) Decrement → preDecr (incl) PostIncrement → incr PostDecrement → decr

- **Mapping from the element CastExpression to the element ActionElement:** The *CastExpression* element is mapped to the *ActionElement* element. The micro action assigned to its attribute *kind* is the *typeCast*. The *input* of this micro action is a *Reads* relationship which represents the *DataElement* affected by the conversion and a *UsesType* relationship which represents the data type. The *output* is represented by an optional *Writes* relationship to a *DataElement*. The attribute *codeElement* of the *ActionElement* element stores the objects resulting from the mapping of the attributes *expression* and *castType* of the *CastExpression* element.
- **Mapping from the element UnaryExpression to the element ActionElement:** The element *UnaryExpression* is mapped to an element *ActionElement*. The value of its attribute *kind* depends on the attribute *operator* of the *UnaryExpression* (see Table 5-33). Moreover, the attribute *codeElement* of the *ActionElement* element takes the value stored within the attribute *operand*.

### Binary Expressions

In the next section, we explain how we defined the mappings between the elements in ASTM\_PHP metamodel which represent the *binary expressions* and the KDM elements. Table 5-34 shows these mappings.

**Table 5-34.** Mappings from binary expressions to KDM elements.

ASTM_PHP METAMODEL		KDM METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Binary Expression	left operator right	Action Element	kind = operator codeElement = left, right actionRelation = Reads, Reads, Writes	The <i>kind</i> attribute depends on the operator: Assign → assign OperatorAssign → add, subtract, ... Add → add Subtract → subtract Multiply → multiply Divide → divide Modulus → remainder Equals → equals NotEqual → notEqual (incl) Identical → identical (incl) NotIdentical → notIdentical Less → lessThan NotGreater → ...

- *Mapping from the element BinaryExpression to the element ActionElement:* The *BinaryExpression* element is mapped to an *ActionElement*. As we can see in Table 5-34, the micro action assigned to its attribute *kind* depends on the attribute *operator* of the *BinaryExpression* element. Moreover, the attribute *codeElement* of the *ActionElement* element takes the value stored within the attributes *left* and *right* of the *BinaryExpression* element.

## PHP Statements

In this section, we present the mappings from the elements of the ASTM\_PHP metamodel that represent PHP Statements.

### Definition Statements

Table 5-35 shows the mappings defined between the ASTM\_PHP elements which represent the *definition statements* and the KDM elements. The rest of these mappings can be found in Appendix E.

**Table 5-35.** Mappings from definition statements to KDM elements.

ASTM_PHP METAMODEL		KDM METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
DefinitionOr DeclarationS statement	declOrDefn	--	name = identifierName kind = initialValue	If declOrDefn = Variable Definition The value of the

ASTM PHP METAMODEL		KDM METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Variable Definition	accessKind identifierName initialValue	Action Element		attribute <i>kind</i> depends on <i>initialValue</i> of the <i>VariableDefinition</i>
DefinitionOrDeclarationStatement	declOrDefn	--	export = accessKind name = identifierName codeElement = body	If declOrDefn = <i>Function Definition</i>
Function Definition	accessKind identifierName formalParameters body	Method Unit		
Formal Parameter Definition	locationInfo identifierName definitionType accessKind initialValue	Parameter Unit	parameterUnit = formalParameters Source = locationInfo name = identifierName kind, type = definitionType pos	No condition

- *Mapping from the element *DefinitionOrDeclarationStatement* with *VariableDefinition* to the element *ActionElement*:* The element *DefinitionOrDeclarationStatement* containing a *VariableDefinition* element is mapped to an *ActionElement*. The value of its attribute *kind* depends on the expression defined within the attribute *initialValue* of the *VariableDefinition* element. For instance, if the *initialValue* is an array access the attribute *kind* of the *ActionElement* refers to the *memberSelect* micro action, otherwise if the *initialValue* is a function call it refers to the *call* micro action. On the other hand, the attribute *name* of the *ActionElement* takes the value from the attribute *identifierName* of the *VariableDefinition* element.
- *Mapping from the element *DefinitionOrDeclarationStatement* with *FunctionDefinition* to the elements *MethodUnit* and *Signature*:* The element *DefinitionOrDeclarationStatement* containing a *FunctionDefinition* element is mapped to a *MethodUnit* element containing a *Signature* element. The main attributes of this element are: *export*, *name* and *codeElement*. Its attribute *export* represents the visibility of the method which is of the type *ExportKind*. This type is an enumerated type defined by a set of values: *public*, *private* and *protected*. The value of this attribute is taken from the attribute *accesskind* of the *FunctionDefinition* element. The attribute *codeElement* allows storing the statements of the body and the parameters defined for the *MethodUnit*. The body is mapped from the attribute *body* of the *FunctionDefinition* element and the parameters from the attribute *formalParameters*. Finally, the parameters mapped are grouped within the object *Signature*. This object is an element composed of a multivalued attribute called *parameterUnit* where the parameters are stored.

- **Mapping from the element *FormalParameterDefinition* to the element *FormalParameterDefinition*:** The parameters defined by the function definition and represented by the *FormalParameterDefinition* element are mapped to the element *ParameterUnit*. This element has the following attributes: *source*, *name*, *kind*, *type* and *pos*. The attribute *source* relates the *ParameterUnit* object of the model with the source code. The value of this attribute is taken from the attribute *locationInfo* of the *FormalParameterDefinition* element. The attribute *name* defines the identifier of the parameter. The value of this attribute is mapped from the attribute *identifierName* of the *FormalParameterDefinition* element. Moreover, the attribute *kind* is an optional attribute defining the parameter passing convention e.g. by value or by reference. The value of this attribute is mapped from the attribute *definitionType*. Moreover, the attribute *type* defines the data type of the parameter. This data type is also taken from the attribute *definitionType* which also defines the primitive type of the parameter. Finally, the attribute *pos* is an optional attribute that indicates the position of the attribute in the signature. There are two attributes of the *FormalParameterDefinition* element, which have not been possible to map to the *ParameterUnit* element. These attributes are *initialValue* and *accessKind*.

### Expression Statements

In the next section, we explain how we defined the mappings between the elements in ASTM\_PHP metamodel which represent the *expression statements* and the KDM elements. Table 5-36 shows these mappings.

**Table 5-36. Mappings from expression statements to KDM elements.**

ASTM_PHP METAMODEL		KDM METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Expression Statement	expression	Action Element	Kind	The value of the attribute <i>kind</i> depends on the expression stored in the attribute <i>expression</i>

- **Mapping from the element *ExpressionStatement* to the element *ActionElement*:** The *ExpressionStatement* element is mapped to an *ActionElement*. The value of its attribute *kind* depends on the expression stored in the attribute *expression* of the *ExpressionStatement* element. For instance, if the *Expression* attribute stores an *Add* expression the attribute *kind* refers to the *add* micro action, otherwise if the *Expression* attribute is an object access it can refers to the *memberSelect* micro action.

## Control Statements

In the following, we explain the mappings which refer to the ASTM\_PHP elements representing *control statements*. Table 5-37 shows these mapping between these ASTM\_PHP elements and KDM elements. The rest of these mappings can be found in Appendix E.

**Table 5-37. Mappings from control statements to KDM elements.**

ASTM_PHP METAMODEL		KDM METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
IfStatement	condition thenBody elseBody	Action Element	kind = Condition codeElement = condition, thenBody, elseBody actionRelation = <i>Reads, TrueFlow, FalseFlow</i>	No condition
For Statement PHP	initBody condition iterationBody body	Action Element	kind = Loop codeElement = initBody, condition, iterationBody, body actionRelation = <i>Reads, TrueFlow, FalseFlow, Flow</i>	No condition
Return Statement	returnValue	Action Element	Kind = Return codeElement = returnValue, actionRelation = <i>Reads, Flow</i>	No condition

- **Mapping from the element IfStatement to the element ActionElement:** The *Ifstatement* element is mapped to an *ActionElement* element whose attribute *kind* is defined as *condition*. The *input* of this micro action is a *Reads* relationship to a *DataElement* representing the Boolean value. Otherwise, this micro action does not define any *output*. On the other hand, it defines a control part composed of two control flows, a *TrueFlow* and a *FalseFlow*. These two control flows are stored within the attribute *actionRelation* of the *ActionElement* element. On the one hand, the *TrueFlow* points to the first *ActionElement* of the body of the *if* statement. This body is mapped from the attribute *thenBody* of the *IfStatement* element and stored within the attribute *codeElement* of the *ActionElement*. On the other hand, the *FalseFlow* points to the first *ActionElement* of the body of the *else* part. The body of this part is mapped from the attribute *elseBody* of the *IfStatement* element.
- **Mapping from the element ForStatementPHP to the element ActionElement:** The *ForStatement* element is mapped to an *ActionElement*. The micro action assigned to its attribute *kind* is *loop*. The specification we have defined for this micro action is as follows, the *input* is defined with a single *Reads* relationship to a *DataElement* representing the Boolean value obtained from the attribute *condition* of the *ForStatement* element. Similarly,

the expressions defined by the attributes *initBody* and *iterationBody* of the *ForStatement* element are also stored in the attribute *codeElement*. We did not need to define outputs for the specification of this micro action. Otherwise, its *control* part is defined by three control flows, a *TrueFlow*, a *FalseFlow* and a *Flow*. These control flows are stored within the attribute *actionRelation* of the *ActionElement* element. The *TrueFlow* points to the first *ActionElement* of the body of the *for* statement. This body is mapped from the attribute *body* of the *ForStatement* element and stored within the attribute *codeElement* of the *ActionElement*. Otherwise, the *FalseFlow* points to the next *ActionElement* of the *for* statement. Finally, the *Flow* relates the last *ActionElement* of the body of the *for* statement to the *ActionElement* to indicate the start of the *loop*.

- ***Mapping from the element ReturnStatement to the element ActionElement:*** The *ReturnStatement* element is mapped to an *ActionElement* element whose attribute *kind* is *return*. The standard specification of this micro action is defined with a *Reads* relationship to a *DataElement* representing the returned value. This *DataElement* is mapped from the attribute *returnValue* of the *ReturnStatement* element. As for the control part, it specifies a control flow to a *ControlElement*. This control flow reflects that the control is returned back to the *ControlElement* which has performed the call.

### 5.3.3 Between KDM Metamodel and CMS Common Metamodel

In this section, we explain the definition of the mappings between the elements of the KDM metamodel and the elements of the CMS Common Metamodel (CMS elements). As we explained previously, the definition of these mappings cover the L2-to-L3 M2M transformations and the L3-to-L2 M2M transformations.

For the definition of these mappings, it is necessary to specify a set of conditions which allows interpreting the KDM elements to carry out the generation the CMS elements. These conditions are related to a specific CMS platform. Namely, depending on the CMS platform we will specify a set of conditions or another to perform the mappings between the KDM elements and the CMS elements.

For the *ADMigraCMS* method, we define the mappings considering two CMS platforms (Wordpress and Drupal). In the following section, we explain the mappings considering the conditions of the Drupal platform. In Appendix E, the

mappings considering the conditions of the Wordpress platform are explained in detail.

### Considering the Drupal Platform

We present in this section the mappings between the KDM elements and the CMS elements considering the Drupal platform. The important point of these mappings is the definition of the conditions to interpret the KDM elements or to generate them from the CMS model.

For the explanation of these mappings we grouped them considering the concerns defined to group the elements of the CMS Common Metamodel (see Chapter 4).

Firstly, in Table 5-41 we present the mappings of the root elements of the KDM metamodel to the root elements of the CMS Common Metamodel.

**Table 5-38. Mappings from the root elements of KDM to the CMS Common Metamodel.**

KDM METAMODEL		CMS COMMON METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
<b>Code Element</b>	codeElement	<b>CMS Model</b>	modules = codeElement Themes = codeElement User = codeElement userRole = codeElement	No conditions
<b>CompilationUnit</b>	codelElement	<b>Module</b>	Blocks Menus menuItems	No conditions

The *CodeElement* of the KDM metamodel is mapped to the *CMSModel* element in the CMS Common Metamodel that represents the root element of the CMS model. The values defined within the attribute *codeElement* are mapped to the different attributes composing the element CMSModel (*modules*, *themes*, *user*, *userRole* among others). Otherwise, the element *CompilationUnit* of the KDM metamodel is mapped to the element *Module* in the CMS Common Metamodel.

### Navigation Concern

We start explaining the mappings to CMS elements belonging to the navigation concern. In the Table 5-39, we present the tables where we analyze the PHP code which implement the CMS elements on the Drupal platform. Thereby, they allowed us to determine the conditions required to define the mappings between the KDM elements and CMS elements in this section.

**Table 5-39. Mappings from the KDM elements to the CMS elements of navigation concern.**

KDM METAMODEL		CMS COMMON METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Action Element	source codeElement	Menu	path = source name = code Element	<b>Source</b> = sites\all\modules\ menu.module <b>Kind</b> = call <b>Calls flow</b> → menu_save
Action Element	source	Menu Item	path = source name, description, position, type, status = code Element	<b>Source</b> = sites\all\modules\ menu.module <b>Kind</b> = arrayReplace Defined in a MethodUnit called name_module_menu
Action Element	codeElement	Menu Item	path = source name, description, position, type, status = code Element	<b>Source</b> = sites\all\modules\ menu.module <b>Kind</b> = arrayReplace <b>Reads flows</b> → indexes and values Defined in a MethodUnit called name_module_menu and in an ActionElement with kind=compound
Action Element	Source codeElement	Page	path = source id, type, title, status, date, comment, author, content, description, name, password = code Element	<b>Source</b> = sites\all\modules\ menu.module <b>Kind</b> = arrayReplace <b>Reads flows</b> → 'page_callback' and name_function_content Defined in a MethodUnit called name_module_menu and in an ActionElement with kind=compound

- **Mapping from the element ActionElement to the element Menu:** This is a 1 to 1 transformation. If any *ActionElement* matches with the following conditions, it is mapped to the element *Menu*. The *first condition* is that this *ActionElement* have to represent a piece of code located in a PHP file stored in *sites\all\modules\menu.module*. The *second condition* is that its attribute *kind* must be defined as a *call* micro action since it is represented by a function call. Finally, the *third condition* is that its *Calls flow* has to point to a *MethodUnit* called *menu\_save*. After generating the *Menu* element it is necessary to assign values to its attributes. Its attribute *name* takes the value from the attribute *codeElement* and its attribute *path* from the attribute *source*.
- **Mapping from the element ActionElement to the element MenuItem:** This is a N to 1 transformation. If any *ActionElement* matches with the following conditions, it is mapped to the element *MenuItem*. The *first condition* is that they have to represent a piece of code in a PHP file located in *sites\all\modules\menu.module*. The *second condition* is that its attribute *kind* is defined as a *arrayReplace*. The third condition is that this *arrayReplace*

must be defined within a *MethodUnit* whose name must finish with the token *\_menu* (called *name\_module\_menu*). The values for the attributes of the element *MenuItem* are taken from a set of *ActionElements*. The conditions of this *ActionElement* are: 1) the attribute *kind* must be defined as an *arrayReplace* micro action since it is the assignment of a value to a position of an array and 2) this *arrayReplace* is defined in a *MethodUnit* called *name\_module\_menu* as well as in an *ActionElement* with kind defined as compound.

- **Mapping from the element ActionElement to the element Page:** This is a 1 to 1 transformation. From one *ActionElement* of the KDM model we generate the *Page* element within the CMS model. This *ActionElement* must match with the following conditions: 1) the source code must be located in *sites\all\modules\menu.module* 2) its attribute *kind* is defined as an *arrayReplace* micro action since, 3) its *Reads* flow points to a value defined as *page\_callback* and 4) this *arrayReplace* is defined in a *MethodUnit* called *name\_module\_menu* as well as in an *ActionElement* with kind defined as compound.

### Presentation Concern

In the following, we explain the mappings between the elements of the KDM model and the CMS elements considered within the presentation concern. Table 5-40 sums up the information explained below.

**Table 5-40. Mappings from the KDM elements to CMS elements of presentation concern.**

<i>KDM METAMODEL</i>		<i>CMS COMMON METAMODEL</i>		<i>CONDITION</i>
<i>Element</i>	<i>Properties</i>	<i>Element</i>	<i>Properties &amp; value</i>	
<b>Compilation Unit</b>	source	<b>Theme</b>	path = source name = text	<b>Source</b> = \sites\all\themes\theme_directory\theme_name.info
<b>Action Element</b>	text			<b>Source</b> = \sites\all\themes\theme_directory\theme_name.info <b>Kind</b> = assign <b>Reads flows</b> → the value <i>theme_name</i> or <i>theme_description</i> or <i>theme_core</i> among others. <b>Writes flows</b> → the <i>StorableUnit name</i> or <i>description</i> or <i>core</i> among others.
<b>Action Element</b>	source actionRelatio n	<b>Region</b>	path = source name = action Relation	<b>Source</b> = \sites\all\themes\theme_directory\theme_name.info <b>Kind</b> = assign <b>Reads flows</b> → the value

KDM METAMODEL		CMS COMMON METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
				<i>name_region</i> Writes flows → the StorableUnit named regions.

- **Mapping from the elements *CompilationUnit* and *ActionElement* to the element *Theme*:** This is a N to 1 transformation. From a *CompilationUnit* and *ActionElement* of the KDM model we generate the element *Theme* within the CMS model. The *CompilationUnit* must meet the following condition: 1) the source code must be located in `\sites\all\themes\theme_directory\theme_name.info`. On the other hand, the *ActionElement* element must meet the following conditions: 1) the source code must be located in `\sites\all\themes\theme_directory\theme_name.info`, 2) its attribute kind has to be defined as an assign and 3) its Reads flows must point to the following values: *theme\_name* or *theme\_description* or *theme\_core* among others.
- **Mapping from the element *ActionElement* to the element *Region*:** This is a 1 to 1 transformation. We generate the *Region* element from an *ActionElement* which is located in `\sites\all\themes\theme_directory\theme_name.info`. This *ActionElement* must be defined as an *assign* and its *Reads* flow must point to the value *name\_region*.

### Behaviour Concern

Below, we explain the mappings between the elements of the KDM model and the CMS elements considered within the behaviour concern. Table 5-41 reflects data of the mappings defined to generate these CMS elements.

**Table 5-41. Mappings from the KDM elements to the CMS elements of behaviour concern.**

KDM METAMODEL		CMS COMMON METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Action Element	Source codeElement	Block	path = source name = codeElement <sup>1</sup> content = code Element <sup>2</sup>	Source = sites\all\modules\block.module Kind = methodCall Calls flow → execute
Compilation Unit	source	Module	path = source	Source = \modules\module_directory
Method Unit	source name	Function	Path = source name = name	Source = \includes\file.inc

- **Mapping from the element ActionElement to the element Block:** It is an 1 to 1 transformation. The *Block* element is obtained from an *ActionElement* defined as a *methodCall* located in *sites\all\modules\block.module*. This *ActionElement* must point to the *methodUnit* execute by means of its *Calls* flow.
- **Mapping from the element CompilationUnit to the element Module:** It is a 1 to 1 transformation. The *Module* element is obtained from a *CompilationUnit* located in \modules\module\_directory.
- **Mapping from the element MethodUnit to the element Function:** It is a 1 to 1 transformation. The *Function* element is obtained from a *MethodUnit* located in \includes\file.in.

### Content Concern

After presenting the mappings related to the CMS elements of the behaviour concern, we explain the mappings between the elements of the KDM model and the CMS elements considered within the content concern. It is worth noting that the elements Language and Category of the CMS Common Metamodel have not been included in the mappings because their implementation in PHP is not considered by Drupal. Table 5-42 presents the complete list of the mappings.

**Table 5-42. Mappings from the KDM elements to the CMS elements of content concern.**

KDM METAMODEL		CMS COMMON METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
<b>MethodU nit</b>	codeElement	<b>Content</b>	text = code Element	<b>Source</b> = sites\all\modules\menu.module <b>name</b> = name_function <i>It has to be called from the page_callback</i>
<b>Action Element</b>	Source codeElement	<b>Comment</b>	path = source id, text, date, status = codeElement	<b>Source</b> = sites\all\modules\comment.module <b>Kind</b> = call <b>Calls flow</b> → comment_submit
<b>Action Element</b>	source codeElement	<b>Term</b>	path = source name = code Element <sup>i</sup> id, description = code Element <sup>ii</sup>	<b>Source</b> = sites\all\modules\term.module <b>Kind</b> = call <b>Calls flow</b> → taxonomy_term_save
<b>Action Element</b>	Source codeElement	<b>Vocabulary</b>	path = source id, name, description = code Element	<b>Source</b> = sites\all\modules\voc.module <b>Kind</b> = call <b>Calls flow</b> → taxonomy_vocabulary_save

- **Mapping from the element MethodUnit to the element Content:** It is a 1 to 1 transformation. The *Content* element is obtained from a *MethodUnit* located in `sites\all\modules\menu.module`. The name of this *MethodUnit* meets with the *Value* of the *ActionElement* defined as an *ArrayReplace* whos value is `page_callback`
- **Mapping from the element ActionElement to the element Comment:** It is an 1 to 1 transformation. The *Comment* element is obtained from an *ActionElement* located in `sites\all\modules\comment.module`. The attribute *kind* of this *ActionElement* is defined as *call*. The *Calls* flow of this *ActionElement* must point to the *MethodUnit comment\_submit*.
- **Mapping from the element ActionElement to the element Term:** It is an 1 to 1 transformation. The *Term* element is obtained from an *ActionElement* located in `sites\all\modules\term.module`. The attribute *kind* of this *ActionElement* is defined as *call*. The *Calls* flow of this *ActionElement* must point to the *MethodUnit taxonomy\_term\_save*.
- **Mapping from the element ActionElement to the element Vocabulary:** It is an 1 to 1 transformation. The *Vocabulary* element is obtained from an *ActionElement* located in `sites\all\modules\voc.module`. The attribute *kind* of this *ActionElement* is defined as *call*. The *Calls* flow of this *ActionElement* must point to the *MethodUnit taxonomy\_vocabulary\_save*.

### User Concern

Finally, we explain the mappings between the elements of the KDM model an the CMS elements considered within the user concern. It is worth noting that the elements RoleGroup and Permission of the CMS Common Metamodel have not been included in the mappings because their implementation in PHP is not considered by Drupal. Table 5-43 presents the complete list of the mappings.

**Table 5-43. Mappings from the KDM elements to the CMS elements of user concern.**

<b>KDM METAMODEL</b>		<b>CMS COMMON METAMODEL</b>		<b>CONDITION</b>
<b>Element</b>	<b>Properties</b>	<b>Element</b>	<b>Properties &amp; value</b>	
<b>Action Element</b>	source codeElement	<b>Role</b>	path = source name = code Element rolePermission= code Element	<b>Source</b> = <code>sites\all\modules\role.module</code> <b>Kind</b> = call <b>Calls flow</b> → <code>user_role_save</code>

KDM METAMODEL		CMS COMMON METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
ActionElement	source codeElement	Permission	path = source	<b>Source</b> = sites\all\modules\permission.module <b>Kind</b> = newArray This ActionElement must be defined within in a <i>MethodUnit</i> called hook_permission.
Action Element	Source codeElement	User	path = source id ,name, user_login, password, email, regDate = codeElement	<b>Source</b> = \sites\all\modules\user.module <b>Kind</b> = call <b>Calls flow</b> → user_save

- **Mapping from the element ActionElement to the element Role:** It is an 1 to 1 transformation. The *Role* element is obtained from an *ActionElement* located in *sites\all\modules\role.module*. The attribute *kind* of this *ActionElement* is defined as *call*. The *Calls* flow of this *ActionElement* must point to the *MethodUnit user\_role\_save*.
- **Mapping from the element ActionElement to the element Permission:** It is a 1 to 1 transformation. The *Permission* element is obtained from an *ActionElement* located in *sites\all\modules\permission.module*. The attribute *kind* of this *ActionElement* is defined as *newArray* defined within a *MethodUnit* called *hook\_permission*.
- **Mapping from the element ActionElement to the element User:** It is a 1 to 1 transformation. The *User* element is obtained from an *ActionElement* located in *sites\all\modules\user.module*. The attribute *kind* of this *ActionElement* is defined as a *call*. The *Calls* flow of this *ActionElement* must point to the *MethodUnit user\_save*.

## 5.4 Concluding Remarks

In this chapter, we have defined the *ADMigraCMS* method. This method is based on ADM, providing all of the advantages offered by that.

The *ADMigraCMS* method includes three **reengineering stages**: the *reverse engineering stage*, where the knowledge from the code that implements a legacy CMS-based Web application is extracted and represented in a set of models at different abstraction levels, the *restructuring stage* where this knowledge is restructured to be adapted to the features of the target CMS platform and the *forward engineering stage* in which the code that implements the target CMS-based Web application is generated.

These stages are structured in four different **modelling levels** in which the knowledge being migrated is represented from different points of view at different abstraction levels (*specific-platform level* and *independent-platform level*): *Level 0 (L0)* represents the code that implements the CMS-based Web application (PHP code); *Level 1 (L1)* represents the code at platform-specific level using a model conforming to the ASTM metamodel (ASTM\_PHP model); *Level 2 (L2)* specifies the code at platform-independent level with a model conforming to the KDM metamodel (Code model) and *Level 3 (L3)* represents the code in a model conforming to the CMS Common Metamodel (CMS model).

For each modelling level, we have defined a **modelling language** in the form of DSL such as, the PHP DSL (L0), the ASTM\_PHP DSL (L1), the KDM\_CODE DSL (L2) and the CMS DSL (L3).

For the PHP DSL, we have specified a **PHP metamodel** that depicts its abstract syntax. The PHP metamodel captures the syntax elements of the PHP programming language. The concrete syntax of this DSL is defined by the PHP programming language itself.

In reference to the ASTM\_PHP DSL, we have defined an **ASTM\_PHP metamodel** to describe its abstract syntax. This metamodel is the extension of the ASTM standard metamodel with specific elements of the PHP code. The concrete syntax has been defined with a customized graphical notation.

According to the KDM\_CODE DSL, its abstract syntax is defined over the **code** and **action packages** of the **KDM standard metamodel**. Its concrete syntax has been defined with a customized graphical notation.

In accordance to the CMS DSL, we have defined its abstract syntax with the **CMS Common Metamodel** that captures the key elements of the CMS domain. Its concrete syntax has been defined with a customized graphical notation.

The *ADMigracMS* method systemizes the migration process with a set of **automated transformations** which allow the transition between the different modelling levels: L0-to-L1 T2M transformation, L1-to-L2 M2M transformation, L2-to-L3 M2M transformation, L3-to-L2 M2M transformation, L2-to-L1 M2M transformations and L1-to-L0 M2T transformation.

These transformations have been defined at metamodel level. We have specified a set of **mappings** that relates the elements of a source metamodel with the elements of a target metamodel. These mappings can be of three different

types: **one-to-one**, one source element is related to one target element; **one-to-many**, one source element is related to many target elements, **many-to-one**, many source elements are related to one target element and **many-to-many**, many source elements are related to many target elements. For all these mappings, we have defined a set of conditions that a source element must meet to be transformed to a target element. Mainly, these conditions are necessary for the *many-to-one* and *many-to-many* mappings since they determine which source element is mapped. It is evidenced with the definition of the mappings between KDM metamodel and the CMS Common Metamodel, since we have specified a set of different conditions depending on the legacy CMS platform.



## *Chapter 6: The ADMigraCMS Toolkit*

---

---



This chapter presents the implementation of the *ADMigraCMS* toolkit that supports the *ADMigraCMS* method presented in the previous chapter. The *ADMigraCMS* method is composed of a set of DSLs (PHP DSL, ASTM\_PHP DSL, KDM\_CODE DSL and the CMS DSL) to define the models involved in the migration process at each modelling level, and a set of transformations (T2M, M2M and M2T) to automate this process. To explain the implementation of all of these components we carried out a set of activities which are explained below:

The definition of the conceptual architecture and the technical design of the toolkit. In Section 6.1, we present the technical design that specifies the technologies used to implement the artefacts composing the conceptual architecture of the *ADMigraCMS* toolkit.

The implementation of the metamodels. In Section 6.2.1, we present the implementation of the PHP metamodel that represents the abstract syntax of the PHP DSL. This metamodel is necessary for the implementation of the model extractor to implement the T2M transformation considered within the *ADMigraCMS* method. In Section 6.2.2, we present the implementation of the CMS Common Metamodel which has been defined in Chapter 4. This metamodel is the abstract syntax of the CMS DSL as well as the cornerstone of the implementation of the M2M transformations between a Code model and a CMS model.

The implementation of graphical editors. In Section 6.3 we present the process to implement tree-like graphical editors for representing graphically ASTM\_PHP models and Code models. In Section 6.3.2, we explain the implementation of a UML-like graphical editor for representing graphically CMS models.

The implementation of the automated transformations. In Section 6.4, we explain the implementation of the transformation rules that implement the L0-to-L1 T2M transformation which generates an ASMT\_PHP model from PHP code. In Section 6.4.2, we present the implementation of the L1-to-L2 M2M transformation that transforms an ASTM\_PHP model into a Code model. Otherwise, in Section 6.4.5, we present the implementation of the L2-to-L1 M2M transformation that allows the opposite way. In Section 6.4.3 and in Section 6.4.4, the implementation of the L2-to-L3 M2M transformation which generates a CMS model from a Code model and the L3-to-L2 M2M transformation which allows

the opposite way, are presented respectively. In Section 6.4.6, we explain the implementation of the L1-to-L0 M2T that generates the PHP code from the ASTM\_PHP model.

For the implementation of the *ADMigracMS* toolkit, we have followed the incremental and iterative process presented in Section 2.4. We do not present the implementation of the ASTM\_PHP metamodel and the KDM metamodel because their implementation in Ecore can be found and downloaded on the site of the ADM (OMG, 2007). As for the PHP DSL, we have not defined a graphical editor because we are not interested in creating graphical models conforming to the PHP metamodel.

## 6.1 Architecture of the *ADMigracMS* Toolkit

The conceptual architecture of the *ADMigracMS* toolkit is a view of the toolkit at high abstraction level, whereas the technical design specifies the technologies used to implement the artefacts composing the conceptual architecture.

### 6.1.1 Conceptual Architecture of the *ADMigracMS* Toolkit

The conceptual architecture of the *ADMigracMS* toolkit is based on an layered approach (Parnas, 1972), (Kulkarni & Reddy, 2003). Concretely, this toolkit is defined over two different layers, *presentation layer* and *logic layer*, as we can see in Figure 6-1. Each layer represents a certain view of this architecture. The use of different layers allows us to keep the traceability among the different artefacts, the reuse of them and the control of their evolution.

The **logic layer** is the core of the toolkit architecture and represents the business logic of the toolkit. On the one hand, in this layer we implement the modelling languages in form of DSLs to define the models involved in each modelling level (L0, L1, L2 and L3) defined in the *ADMigracMS* method; and on the other hand, we implement the automated transformations which allow the transformations between these modelling levels.

As we can see in Figure 6-1, we implement the PHP DSL defined at the L0 modelling level, the ASTM\_PHP DSL to define an ASTM\_PHP model at L1 modelling level, the KDM\_CODE DSL that specifies a Code model at L2 modelling level and the CMS DSL that defines the CMS model at L3 modelling

level. This implementation is centred in implementing the metamodels which represent their abstract syntax of these DSLs.

As for the transformations, we have implemented the L0-to-L1 T2M transformation that extracts an ASTM\_PHP model from the PHP code, the L1-to-L2 M2M transformation that generates a Code model from a ASTM\_PHP model as well as the L2-to-L1 M2M transformation that allows the reverse process, the L2-to-L3 M2M transformation that generates a CMS model from a Code model as well as its reverse L3-to-L2 M2M transformation, and finally, the L1-to-L0 M2T transformation that generates PHP code from an ASTM\_PHP model.

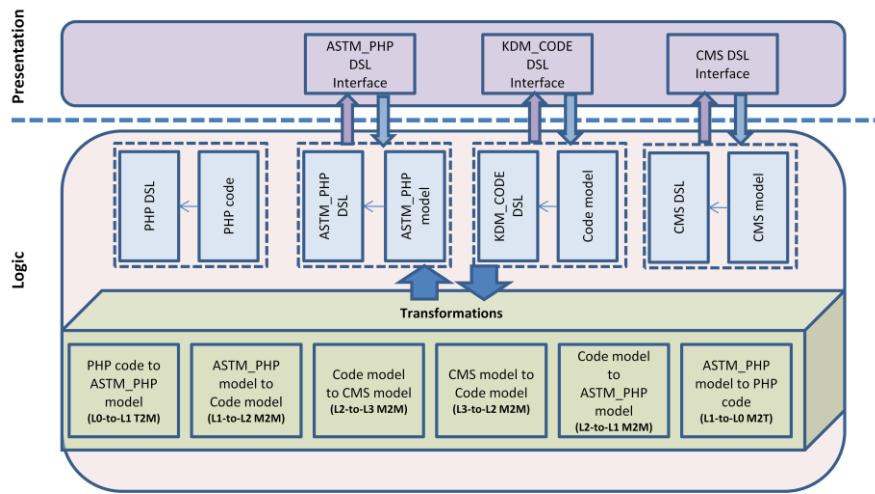


Figure 6-1. Conceptual architecture of the *ADMigraCMS* toolkit.

The **presentation layer** represents the interaction between the toolkit and the user. Thus, in this layer we implement the interfaces in form of graphical editors to edit the models conforming to the DSLs defined in the *ADMigraCMS* method. This implementation is centred in implementing the concrete syntax of the DSLs of the logic layer. As we can see in Figure 6-1, we implement three interfaces: the ASTM\_PHP DSL interface, KDM\_CODE DSL interface and CMS DSL interface.

### 6.1.2 Technical Design of the *ADMigraCMS* Toolkit

After defining the conceptual architecture of the *ADMigraCMS* toolkit, we define its technical design. Therefore, we select the technologies to implement this architecture. Figure 6-2 shows these technologies and the resulting components of

the implementation of the *ADMigraCMS* toolkit. Furthermore, it presents the relationship between the technical design and the layers of the conceptual architecture presented in the previous section.

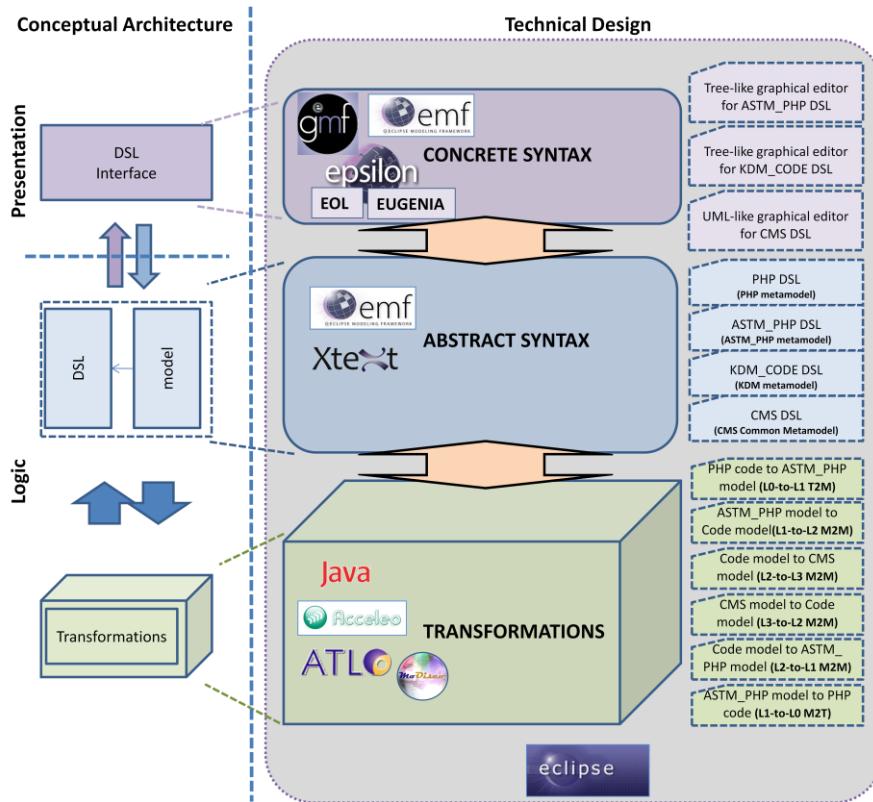


Figure 6-2. Technical design of the *ADMigraCMS* toolkit.

As we can see in Figure 6-2, the technical design is divided in three principal modules: the transformations, the abstract syntax and the concrete syntax.

The **concrete syntax** module corresponds to the presentation layer of the conceptual architecture. It is focused on the implementation of the concrete syntax of the DSLs in the form of graphical editors. These editors allow the graphical definition of models conforming to a concrete DSL. The graphical editors implemented are: two tree-like graphical editors for defining models conforming to the ASTM\_PHP DSL and KDM\_CODE DSL; and one UML-like graphical editor to specify models which conform to the CMS DSL. To implement the tree-like graphical editors we have used the Eclipse Modelling Framework (EMF)

(Steinberg et al., 2008), (Budinsky, 2008), whereas to implement the UML-like graphical editor we have used the Graphical Modelling Framework (GMF) (Eclipse Foundation, 2006). As we will see in Section 6.3.2, the implementation of the UML-like graphical editor is supported by the Epsilon Object Language (EOL) (Kolosovs, Rose, García-Domínguez, & Paige, 2008) and the tool EuGENia (Epsilon, 2012) which allows the semi-automatic implementation of this graphical editor. EOL and EuGENia are part of the project Epsilon (Eclipse Foundation, 2012).

The **abstract syntax** module is related to the logic layer of the conceptual architecture. It considers the implementation of the metamodels that represent the abstract syntax of the named DSLs. As we can see in Figure 6-2, this module is composed of four metamodels. The PHP metamodel is implemented by using the framework Xtext (German company Itemis, 2006), whereas the ASTM\_PHP metamodel, the KDM metamodel and the CMS Common Metamodel are implemented by using EMF as Ecore models. In this PhD Thesis, we present the implementation of the PHP metamodel and the CMS Common Metamodel since the implementation of the ASTM metamodel (which the ASTM\_PHP metamodel is based on) and KDM metamodel can be downloaded from the official website of ADM (<http://adm.omg.org/>).

The **transformations** module is related to the logic layer of the conceptual architecture and it represents the implementation of the transformations which automate the *ADMigraCMS* method. The T2M transformations which extract ASTM\_PHP models from the PHP code are implemented in Java in the form of a model extractor. Otherwise, the M2M transformations which allow the transformation of a target model from a source model are implemented with transformation rules defined with Transformation Language (ATL) (Eclipse Foundation, 2013). Finally, the M2T transformations which generate the PHP code from the ASTM\_PHP models are implemented by a set of transformation rules defined in Acceleo (Musset et al., 2008).

## 6.2 Implementation of the Metamodels

The metamodels defined in the *ADMigraCMS* method represents the abstract syntax of the DSLs proposed. This abstract syntax specifies the elements and their relationships that compose the modelling languages.

In this section, we present the implementation of the PHP metamodel and the CMS Common Metamodel. The former is implemented by using the Xtext framework and the latter is implemented as an Ecore model using EMF.

It is worth noting that the implementation of the ASTM and KDM standard metamodels can be download from the official website of ADM (<http://adm.omg.org/>).

### **6.2.1 PHP Metamodel**

The implementation of the PHP metamodel is carried out by using the grammar language provided by the framework Xtext. Furthermore, this framework allows generating automatically a set of tools and artefacts related to the PHP metamodel: i) an implementation of the PHP metamodel in EMF (Ecore), ii) a parser to recognize the syntax elements from a source PHP code and iii) a textual editor that allows writing and validating code in PHP.

The parser obtained from the implementation of the PHP metamodel in Xtext is required to implement the model extractor that implements the L0-to-L1 T2M transformation to obtain an ASTM\_PHP model from the PHP code.

The grammar language provided by Xtext is composed of a set of rules: 1) *parser rules* to implement the classes of the metamodel, 2) *terminal rules* to produce single atomic terminal tokens, 3) *data type rules* to create instances of new types and 4) *enum rules* to return new types defined as enumerate types. These rules are based on the EBNF standard (ISO/IEC, 1996).

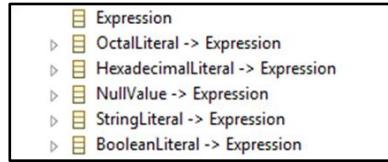
In next sections, we explain the implementation of the classes of the PHP metamodel by using parser rules in Xtext. To present this implementation, we have used a set of tables whose first and second columns determine the name of the class and its attributes, whereas the third column presents the implementation in Xtext (parser rule).

It is worth noting that the implementation of these parser rules has been considered one of the most tedious and time-consuming activities since we had to resolve the left-recursivity conflicts generated among them.

#### **PHP Expressions**

To implement the classes that represent PHP Expressions, we have created an element called *Expression* from which the rest of expressions inherit. To define this class we have implemented accordingly a parser rule named *Expression*.

Figure 6-3 shows an excerpt of the PHP metamodel where the *Expression* class and the rest of elements that inherit from that class.



**Figure 6-3. Classes of the PHP metamodel representing PHP Expressions.**

### Simple Expressions

Table 6-1, presents the parser rules implemented in Xtext to create the classes that represent the simple expressions in PHP.

**Table 6-1. Implementation of the simple expressions in Xtext.**

<i>PHP METAMODEL</i>		<i>XTEXT IMPLEMENTATION (PARSER RULE)</i>
<i>Class</i>	<i>Properties</i>	
<b>Name</b>	nameIdentifier	Name:nameIdentifier=(ID VAR);
<b>Variable</b>	value	Variable:value=(ID VAR);
<b>NumberLiteral</b> <b>StringLiteral</b> <b>FloatLiteral...</b>	value	NumberLiteral: value=INT; StringLiteral: value=STRING; FloatLiteral: value=FLOAT;
<b>ArrayType</b>	params	ArrayType:{ArrayType} (array' OPENBRACE (params+=ArrayEntry (COMA params+=ArrayEntry)* COMA?)? CLOSEBRACE )
<b>ArrayEntry</b>	key value	ArrayEntry:(key=Key ARRAYASSIGN value=Expression) value=Expression;

The class **Name** is implemented with a parser rule called *Name*, whereas the class **Variable** is implemented with a parser rule called *Variable*. The implementation of both classes are similar. They call two terminal rules *ID* and *VAR*. *ID* represents a sequence of alphanumerical characters and underscores, whereas *VAR* represents a dollar symbol (\$) followed by an alphabetical character or underscore (\_) and a set of alphanumerical characters and underscores

The classes **NumberLiteral**, **StringLiteral**, **FloatLiteral**, implemented in Xtext with parser rules. Each parser rule calls a different terminal rule representing a literal. For instance, the parser *NumberLiteral* calls the terminal rule *INT* which is a sequence of numbers.

The class **ArrayType** is implemented with a parser rule called *ArrayType*. Figure 6-4 presents the correspondence between the PHP code for an array definition (*ArrayType*) and its parser rule with Xtext.

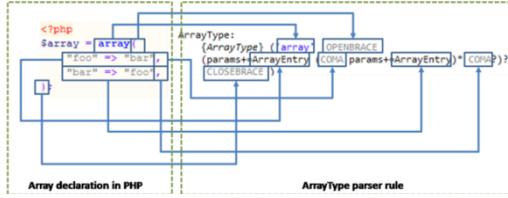


Figure 6-4. ArrayType parser rule.

As we can see in Figure 6-4, the *ArrayType* rule starts with the keyword *array* followed by a couple of parentheses. The parentheses are represented by two terminal rules called *OpenBrace* and *CloseBrace*. Among the parentheses it is possible to specify an optional set of *ArrayEntries* separated by commas. The class **ArrayEntry** is implemented in Xtext with a parser rule called *ArrayEntry* (see Table 6-1) which defines two attributes called *key* and *value* in the PHP metamodel.

### Function Call Expressions and Array Access Expressions

In this section, we explain the implementation of the classes representing *function call expressions* and *array access expressions*. Table 6-2 presents the Xtext implementation of these expressions.

Table 6-2. Implementation of the PHP expressions in Xtext.

PHP METAMODEL		XTEXT IMPLEMENTATION (PARSER RULE)
Class	Properties	
<b>FunctionCall Expression</b>	FunctionIdentifier params	FunctionCallExpression:functionIdentifier=Variable (OPENBRACE (params+=Expression (COMA params+=Expression)*?)? CLOSEBRACE);
<b>ArrayAccess</b>	arrayIdentifier, sus	ArrayAccess:arrayIdentifier=Variable (sus+=Suscript)+;
<b>Subscript</b>	expression	Suscript:open=OPENSQUAREBRACE (expression=Expression)? close=CLOSESQUAREBRACE;

The class **FunctionCallExpressions** is implemented with a parser rule called *FunctionCallExpression*. Figure 6-5 presents the correspondence among the PHP code for a function call expression and its parser rule.

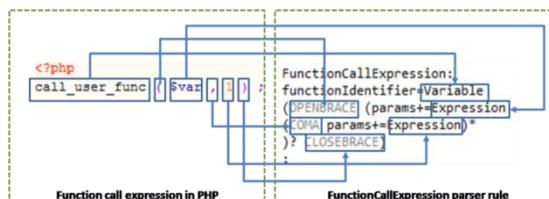


Figure 6-5. FunctionCallExpression parser rule.

As we can see in Figure 6-5, *FunctionCallExpression* rule starts with the definition of the name of the called function. This name is stored within the attribute called *functionIdentifier*. Then, it defines the parameter passing which is defined as a set of expressions separated by commas and contained among a pair of parentheses. As occurred with the *ArrayType* rule, the parentheses are depicted by two terminal rules called *OpenBrace* and *CloseBrace*. Otherwise, the expressions representing the parameters are stored in the attribute *params*.

The class **ArrayAccess** is implemented with a parser rule called *ArrayAccess*. Figure 6-6 presents the correspondence among the PHP code for an array access and its parser rule.

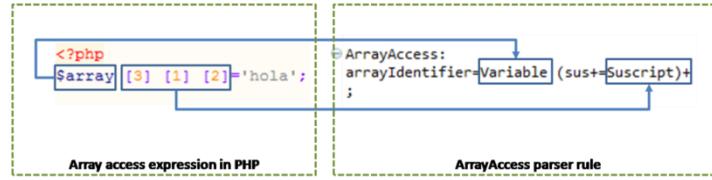


Figure 6-6. ArrayAccess parser rule.

The first element defined in the *ArrayAccess* parser rule is the name of the array being accessed. This name is stored within the attribute *arrayIdentifier*. The array access operator is depicted by two square brackets ([ ]) as we can see in Figure 6-6. Within these square brackets, it is possible to define an expression that evaluates to a value determining the position of the array to be accessed. These accesses are represented by *Subscripts* stored in the attribute *sus*.

The class **Subscript** is implemented with the parser rule *Subscript*. As we can see in Table 6-2, this parser rule specifies three attributes: *open* and *close* which stores the square brackets and *expression* to store the expressions.

### Unary Expressions

In this section, we explain the implementation of the classes representing unary expressions by using parser rules in Xtext. To implement these parser rules we have considered the precedence of operators in PHP, mentioned in Section 5.2.1.1. Table 6-3 presents the Xtext implementation of these unary expressions.

Table 6-3. Implementation of the unary expressions in Xtext.

PHP METAMODEL		XTEXT IMPLEMENTATION (PARSER RULE)
Class	Properties	
NegateOrCast	operator right cast	NegateOrCast returns Expression: {NegateOrCast}(OPENBRACE cast=PRIMITIVE_TYPE CLOSEBRACE) right=Incremen

PHP METAMODEL		XTEXT IMPLEMENTATION (PARSER RULE)
Class	Properties	
		{NegateOrCast}operator=(TILDE MINUS SUPPRESSWARNIN GS)right=Increment;
LogicalNot	exp	LogicalNot returns Expression:{LogicalNot}BANG expr=LogicalNot InstanceOf;
Reference	value_ref	Reference:AMPERSAND value_ref= NameOrFunctionCall;

The class **NegateOrCast** is implemented with a parser rule called *NegateOrCast*. This parser specifies two different definitions. The first one defines a casting unary expression and the second one a unary expression defined with a negation arithmetic operator (-) or a not bitwise operator (~). Figure 6-7 presents the correspondence among the PHP code for a casting unary expression and this parser rule.

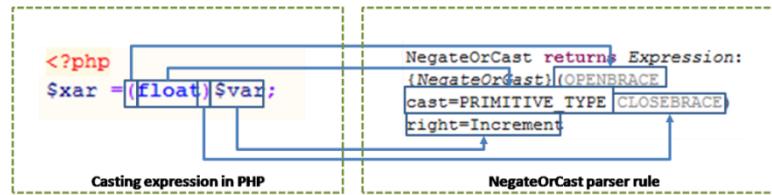


Figure 6-7. NegateOrCast parser rule.

As we can see in Figure 6-7, the *NegateOrCast* rule defines the casting operator and the expression we want to shift. The primitive type of the casting operator is defined by the terminal rule *Primitive\_Type* and it is stored in the attribute called *cast*; otherwise, the pair of parentheses is represented by the terminal rules *OpenBrace* and *CloseBrace*. Finally, the expression is stored in the attribute *right*.

The class **LogicalNot** is implemented with a parser rule called *LogicalNot*. Figure 6-8 shows the correspondence among the not logical expression in PHP and the *LogicalNot* parser rule implemented in Xtext.

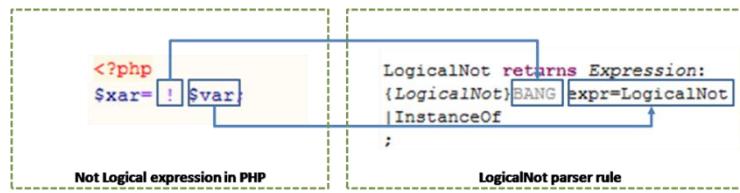


Figure 6-8. LogicalNot parser rule.

As it is shown in Figure 6-8, the *LogicalNot* rule starts with the not logical operator which is represented with the terminal rule *Bang*. This logical operator is followed by an expression which is stored in the attribute *expr*.

The class **Reference** is implemented with a parser rule called *Reference*. Figure 6-9 shows the correspondence among the reference expression in PHP and the *Reference* rule implemented in Xtext.

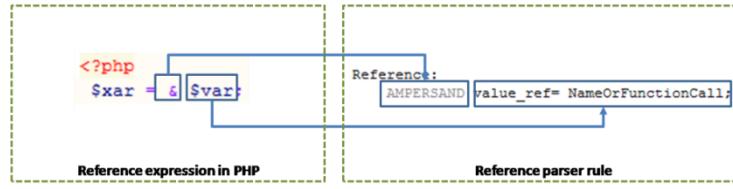


Figure 6-9. Reference parser rule.

As we can see in Figure 6-9, the *Reference* rule starts with the reference operator represented with the terminal rule *Ampersand*. This operator is followed by an expression which is stored in the attribute *value\_ref*.

### Binary Expressions

In this section, we explain the implementation of the classes representing binary expressions. The classes have been implemented by defining parser rules in Xtext. To implement these parser rules we have also considered the precedence of operators in PHP. Table 6-4 presents the Xtext implementation of these binary expressions.

Table 6-4. Implementation of the binary expressions in Xtext.

PHP METAMODEL		XTEXT IMPLEMENTATION (PARSER RULE)
Class	Properties	
<b>Assignment</b>	left operator right	Assignment returns <i>Expression:Ternary</i> ({Assignment.left=current} operator=(EQUALS ASSIGNOPERATOR) right=Ternary)*;
<b>Addition</b>	left operator right	Addition returns <i>Expression: Multiplication</i> ({Addition.left=current} operator=(DOT MINUS PLUS) right=Multiplication)*;
<b>Equality Check</b>	left operator right	EqualityCheck returns <i>Expression:ComparisonCheck</i> ({EqualityCheck .left=current} operator=EQUALITY_OPERATOR right=ComparisonCheck)?;
<b>BitWise Shift</b>	left right operator	BitwiseShift returns <i>Expression:Addition</i> ({BitwiseShift.left=current} operator=SHIFT_OPERATOR right=Addition)*;

PHP METAMODEL		XTEXT IMPLEMENTATION (PARSER RULE)
Class	Properties	
LogicalAnd	left right	LogicalAnd returns Expression: BitwiseOr ({LogicalAnd.left=current} LOGICAL_AND right=BitwiseOr)*;
InstanceOf	left right	InstanceOf returns Expression: NegateOrCast ({InstanceOf.left=current} INSTANCEOF_OPERATOR right=NegateOrCast)?;

The class **Assignment** is implemented by the parser rule *Assignment*. Figure 6-10 shows the correspondence among the assignment expression in PHP and the *Assignment* parser rule implemented in Xtext.

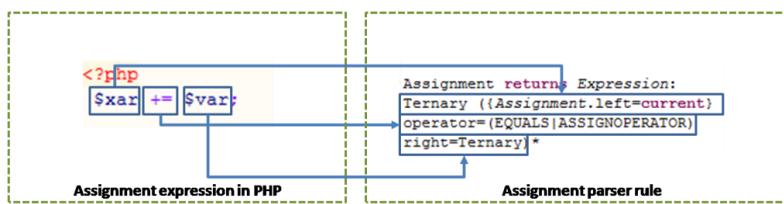


Figure 6-10. Assignment parser rule.

As it is shown in Figure 6-10, this rule is defined with the two operands of the binary expression. The first operand is stored in the attribute *left* and the second one in the attribute *right*. The assignment operator is defined by two terminal rules *Equals* and *AssignOperator*. The basic assignment operator (=) is represented by *Equals*. Otherwise, the combined assignment (the assignment operator (=) along with other operators such as arithmetic operators, string operators or bitwise operators) are represented by the *AssignOperator* terminal rule.

The class **Addition** is implemented by the parser rule *Addition*. Figure 6-11 shows the correspondence among the addition expression in PHP and its parser rule implemented in Xtext.

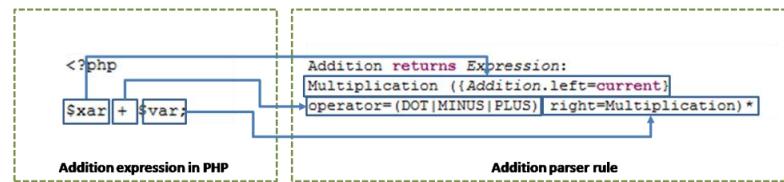


Figure 6-11 . Addition parser rule.

The operands of these binary expression presented in Figure 6-11 are stored in the attributes *left* and *right*. Also, it defines an attribute called *operator*

which stores the operators represented by the following terminal rules *Dot* (.), *Minus* (-) and *Plus* (+).

The class **EqualityCheck** is implemented by the parser rule *EqualityCheck* parser rule. Figure 6-12 shows the correspondence among the equalityCheck expression in PHP and its parser rule implemented in Xtext.

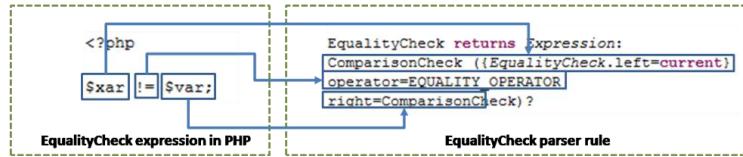


Figure 6-12. EqualityCheck parser rule.

As we can see in Figure 6-12, the left hand side operand is stored in the attribute *left* and the right side operand is stored in the attribute *right*. Otherwise, the operator is stored in the feature called *operator* by means of the terminal rule called *Equality\_Operator*.

Finally, the class **BitWiseShift** is implemented by the parser rule *BitWiseShift* parser rule. Figure 6-13 shows the correspondence among a bitwiseShift expression in PHP and the *BitWiseShift* parser rule implemented in Xtext.

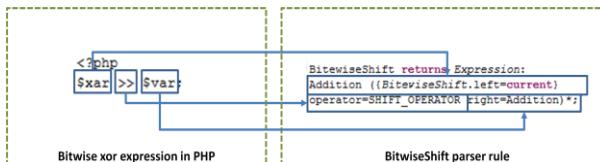


Figure 6-13 . BitwiseShift parser rule.

The *BitWiseShift* parser rule is very similar to the other binary expressions defined; two operands are defined (*left* and *right*) and an operator is represented by the terminal rule *Shift\_Operator*.

The class **LogicalAnd** is implemented by the parser rule presented in Figure 6-14.

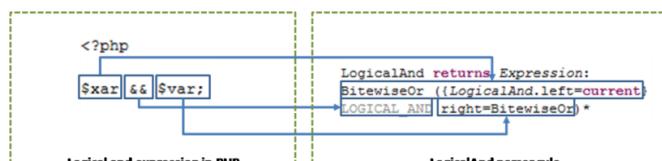


Figure 6-14. LogicalAnd parser rule.

In the parser rule presented in Figure 6-14, we just store the two operands within the attributes *left* and *right*. The operator in this case is represented by the terminal rule *Logical\_And* but it is not stored within any attribute. The same happens with the other parser rules representing these binary expressions.

The class **Instanceof** is implemented by using a parser rule called *InstanceOf*. The correspondence among the instanceof expression defined in PHP and the implementation of its parser rule is presented in Figure 6-15.

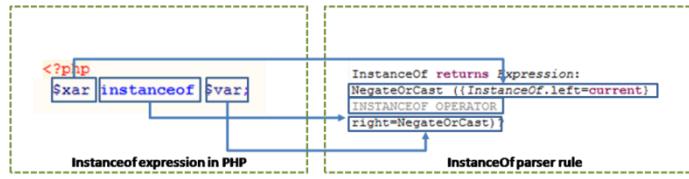


Figure 6-15. *InstanceOf* parser rule.

As we can see in Figure 6-15 and as occurred with the other binary expressions, the left hand side operand is stored in the *left* attribute and the right side operand is stored in the *right* attribute. Otherwise, the operator is represented with the terminal rule *InstanceOf\_Operator* and it is not stored in any attribute

## PHP Statements

In this section, we explain the implementation in Xtext of the classes of our PHP metamodel that represent PHP Statements. We have defined a super class called *Statement* which is the root of the rest of classes implemented. To define this class we have implemented a parser rule in Xtext called *Statement*. Figure 6-16 shows an excerpt of the PHP metamodel where the *Statement* class and the rest of classes are defined.

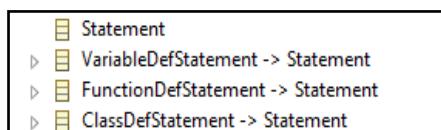


Figure 6-16. Classes of the PHP metamodel representing PHP Statements.

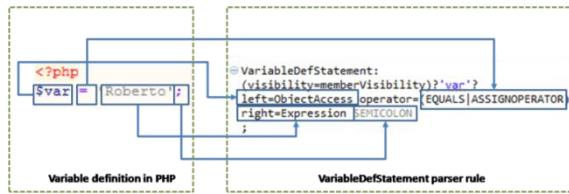
## Definition Statements

In this section, we explain the implementation of the classes representing definition statements. The classes have been implemented by defining parser rules in Xtext. Table 6-5 presents the Xtext implementation of these definition statements.

**Table 6-5. Implementation of the definition statements in Xtext.**

<i>PHP METAMODEL</i>		<i>XTEXT IMPLEMENTATION (PARSER RULE)</i>
<i>Class</i>	<i>Properties</i>	
<b>VariableDef Statement</b>	visibility left right	VariableDefStatement:(visibility=memberVisibility)?'var'? left=ObjectAccess operator=(EQUALS ASSIGNOPERATOR) right=Expression SEMICOLON;
<b>FunctionDef Statement</b>	visibility functionIdentifier params body	FunctionDefStatement:(visibility=memberVisibility)? 'function' functionIdentifier=Variable OPENBRACE (params+=FuncParameters (COMA params+=FuncParameters)*)? CLOSEBRACE OPENCURLYBRACE(body+=Statement)*CLOSECURLYBRA CE;
<b>Func Parameters</b>	var value	FuncParameters:(ref=AMPERSAND)? var=Variable (EQUALS value=Primary)?;

The class **VariableDefStatement** is implemented by the parser rule *VariableDefStatement*. Figure 6-17 shows the correspondence among the this expression in PHP and the *VariableDefStatement* parser rule in Xtext.

**Figure 6-17. VariableDefStatement parser rule.**

As we can see in Figure 6-17, this parser rule starts with the definition of the visibility of the variable. The variables defined within class definitions can specify their level of visibility (private, public and protected). In the Xtext, this visibility is defined by an enum rule called *memberVisibility*. The value of this enumerated type is stored within the attribute called *visibility*. Also, it is possible to use the keyword *var* to declare the variable as public. Hence, we have defined in the parser rule this optional keyword. The structure of variable definitions is the same as a binary expression but ended with semicolon: two operands (the left hand operand and the right hand operand) and an assignment operator. The operands are stored in the attributes *left* and *right* and the assignment operator within the attribute *operator*. As we can see, the assignment operator is defined by two terminal rules *Equals* (=) and *AssignOperator* (+=, \*=).

The class **FunctionDefStatement** is implemented with a parser rule called *FunctionDefStatement*. Figure 6-18 shows the correspondence among a function definition specified in PHP and the parser rule *FunctionDefStatement* implemented in Xtext.

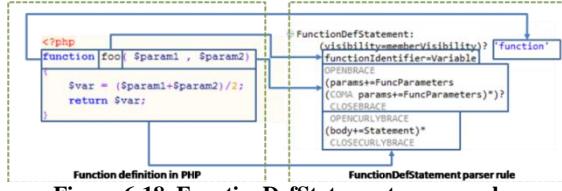


Figure 6-18. FunctionDefStatement parser rule.

As we can see in Figure 6-18, the first element of the parser rule *FunctionDefStatement* is the visibility level defined by the *enum* rule *memberVisibility* as occurred with the variable definition. Then, we define the keyword *function* which is required for the function definition. Following the keyword *function*, it is necessary to define the name which is stored within the attribute *functionIdentifier*. Following the identifier, we need to put a pair of brackets represented by the terminal rules *OpenBracket* and *CloseBracket*. Among the brackets it is possible to specify a set of parameters separated by comas. These parameters are defined by the class *FuncParameters*. The set of parameters are stored in the attribute *params*. After defining the parameters, it is required to define the body of the function. This body is determined by a pair of curly braces depicted by the terminal rule *OpenCurlyBrace* and the terminal rule *CloseCurlyBrace*. The body is composed of a set of statements which are stored within the attribute *body*.

The **FuncParameters** class that represent the parameters passed to the function is implemented by the parser rule *FuncParameter*. Its attribute *ref* defines the mechanism for passing the parameter, by value or by reference. In case of being passed by reference, this attribute stores the string `&`. The attribute *var* is used to store the name of the parameter and the attribute *value* stores the optional initializing value.

### Expression Statements

In this section, we explain the implementation of the classes representing expression statements. The classes have been implemented by defining parser rules in Xtext. Table 6-6 presents the Xtext implementation of these expression statements.

Table 6-6. Implementation of the expression statements in Xtext.

PHP METAMODEL		XTEXT IMPLEMENTATION (PARSER RULE)	
Class	Properties		
FunctionCall StatementExp	exp	FunctionCallStatement:FunctionCallExpression SEMICOLON;	

PHP METAMODEL		XTEXT IMPLEMENTATION (PARSER RULE)
Class	Properties	
Increment Statement	operator right left	IncrementStatement:(operator=INCREMENT_OPERATOR right=Primary SEMICOLON)(left=Primary operator=INCREMENT_OPERATOR SEMICOLON);

As we can see in Table 6-6, the class **FunctionCallStatementExp** is implemented by the parser rule *FunctionCallStatement*. Furthermore, the class **ObjectAccessStatement** is implemented with the parser rule *ObjectAccessStatement*, whereas the class **Member** is implemented by the parser rule *Member*. Finally, the class **IncrementStatement** is implemented by the parser rule *IncrementStatement*. We have decided not to explain in detail these Xtext parser rules because of the similarity with the parser rules that implement the classes *FunctionCallExpression* and *Increment* (respectively).

### Control Statements

In this section, we explain the implementation of the classes representing control statements. The classes have been implemented by defining parser rules in Xtext. Table 6-7 presents the Xtext implementation of these control statements.

Table 6-7. Implementation of the control statements in Xtext

PHP METAMODEL		XTEXT IMPLEMENTATION (PARSER RULE)
Class	Properties	
If Statement	condIf ifStatements listElsif elseStatements	IfStatement:'if OPENBRACE condIf=Expression CLOSEBRACE(ifStatements+=Statement  OPENCURLYBRACE(ifStatements+=Statement)* CLOSECURLYBRACE)(listElsif+=Elsif)*(('else'(elseStatements+=Statement OPENCURLYBRACE(elseStatements+=Statement)* CLOSECURLYBRACE))?:';
Elsif	condElsif elsifStatements	Elsif:'elseif OPENBRACE condElsif=Expression CLOSEBRACE(elsifStatements+=Statement  OPENCURLYBRACE (elsifStatements+=Statement)* CLOSECURLYBRACE);
For Statement	exp1 exp2 exp3 statements	ForStatement:{ForStatement}'for' OPENBRACE(exp1=Expression? SEMICOLONexp2=Expression? SEMICOLONexp3=Expression?) CLOSEBRACE(OPENCURLYBRACE (statements+=Statement)* CLOSECURLYBRACE statements+=Statement);
Return Statement	return	ReturnStatement:('return' return=Expression SEMICOLON);

The class **IfStatement** is implemented with a parser rule called *IfStatement*. The correspondence among the *if* statement defined in PHP and the implementation of its parser rule is presented in Figure 6-19.

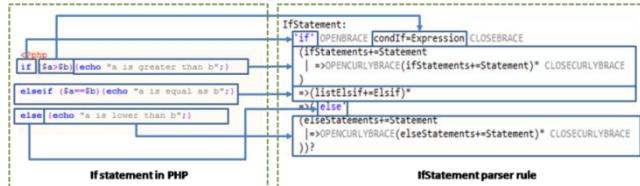


Figure 6-19. IfStatement parser rule.

As we can see in Figure 6-19, the *IfStatement* parser rule starts with the keyword *if* which is required to specify an if statement. This keyword is followed by a pair of brackets represented as *OpenBrace* and *CloseBrace*. Among the brackets it is possible to specify the condition which is represented by an expression and it is stored within the attribute *condIf*. After the brackets, we define the body of the if statement which is defined as a set of statements stored in the attribute *ifStatements*. It is possible to define alternatives of the if statement with the class *Elseif*. The set of *Elseif* is stored within the attribute *listElseif*. Finally, the statements composing the body of the *else* are stored in the attribute *elseStatements*.

The class **Elsif** is defined by the parser rule *Elsif*. Its attribute *condElsif* defines the expression being evaluated and the attribute *elsifStatements* stores the statements composing its body.

The class **ForStatement** has been implemented with a parser rule called *ForStatement*. The correspondence among the function class definition in PHP and the implementation of its parser rule is presented in Figure 6-20.

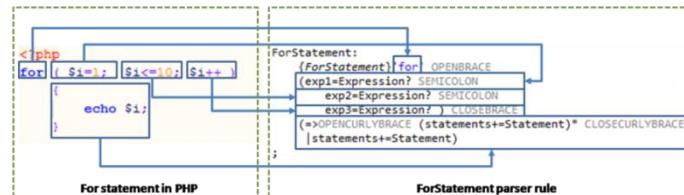


Figure 6-20. ForStatement parser rule.

As we can see in Figure 6-20, the *ForStatement* parser rule starts with the keyword *for* which is required to specify a *for* statement. The keyword is followed by a pair of brackets represented as *OpenBrace* and *CloseBrace*. Among the brackets it is required to define three expressions separated by a semicolon. The first expression is executed at the beginning of the first iteration and it defines the initial value of the variable. This expression is stored with the attribute *exp1*. The second expression is the condition allowing the iteration. It is stored in the

attribute *exp2*. The third expression increments or decrements the value of variable. It is stored in the attribute *exp3*. The body of the for statement is defined by the terminal rules *OpenCurlyBrace* and *CloseCurlyBrace*

The class **ReturnStatement** is implemented by the parser rule *ReturnStatement*. Figure 6-21 shows the correspondence among the return statement in PHP and the implementation of its parser rule.

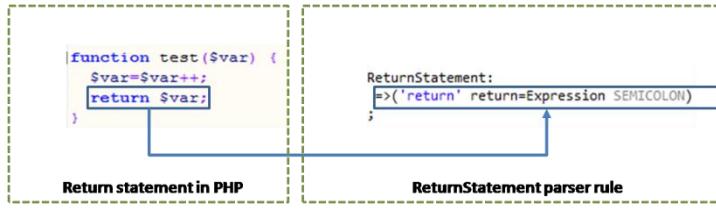


Figure 6-21. ReturnStatement parser rule.

As we can see in Figure 6-21, this rule starts with the keyword *return* and then it is necessary to define an expression which represents the value to return e.g. the name of a variable.

### 6.2.2 CMS Common Metamodel

In this section, we explain the implementation of the CMS Common Metamodel which has been defined in Chapter 4. By using EMF we have defined an Ecore model to implement the CMS Common Metamodel. It is shown in Figure 6-22.

Ecore, the modelling language of EMF, is a simplified implementation of EMOF (Essential MOF) (OMG, 2006). For this reason, due to the XML nature of the underlying storage format of Ecore, any model defined in terms of Ecore has to include a root class. In the case of the CMS Common Metamodel, we have defined the class CMSModel

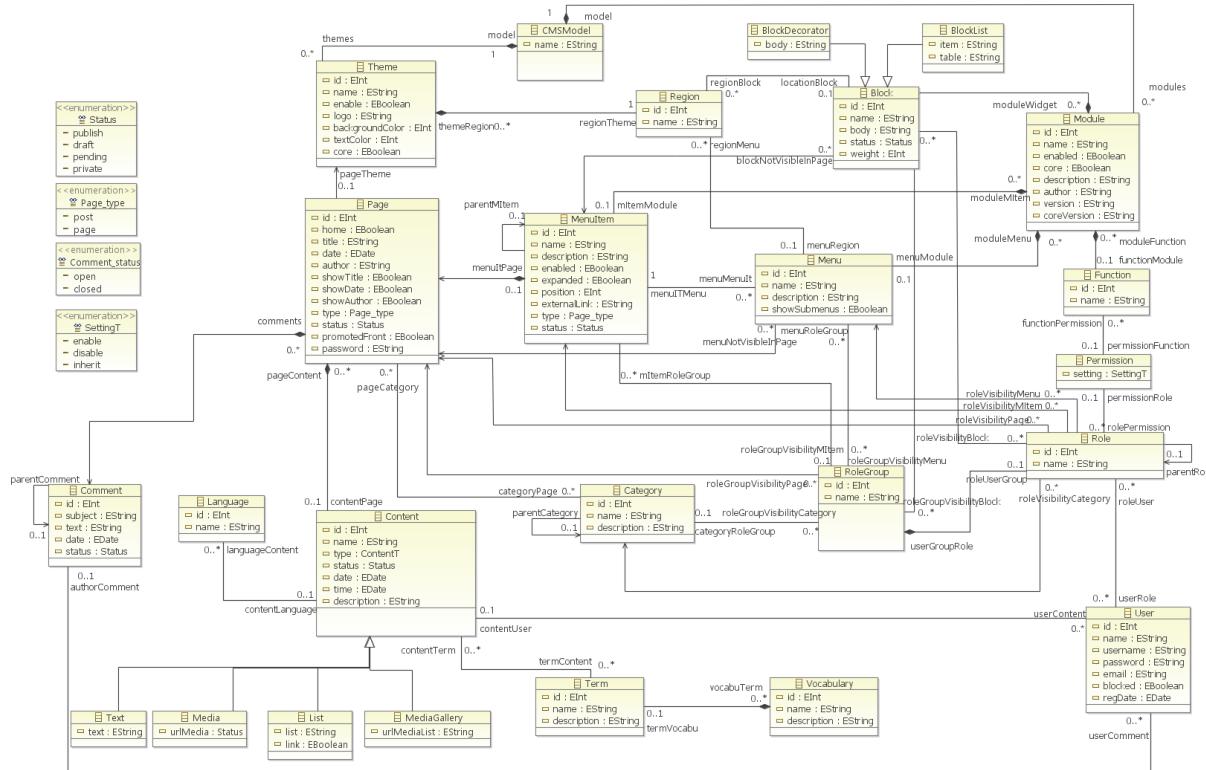


Figure 6-22. Ecore Model for the implementation of the CMS Common Metamodel.

## 6.3 Implementation of the Graphical Editors

In this section, we present the implementation of the graphical editors that support the ASTM\_PHP DSL, the KDM\_CODE DSL and the CMS DSL. These graphical editors allow the creation and edition of graphical models conforming to those DSLs.

We implement a set of graphical editors that must be intuitive, clear and easy-to-use for the developer. Thus, to represent the models conforming to the ASTM\_PHP and KDM metamodels, we have implemented two tree-like graphical editors that represent easily the syntax and semantics of the code being migrated. Otherwise, to create and edit the CMS models, we have implemented a UML-like graphical editor that allows to represent by means of classes and relationships the code in the CMS domain in a user-friendly way.

### 6.3.1 Tree-like Graphical Editors

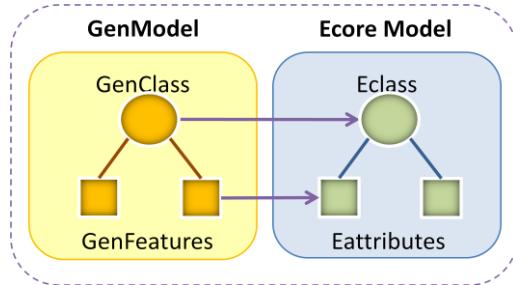
To support the creation and edition of the models conforming to the ASTM\_PHP DSL, we have implemented a graphical editor. The ASTM\_PHP models created by this graphical editor follow a tree-like structure which has been considered the most intuitive manner to represent these models.

For the creation and edition of the Code models, conforming to the KDM\_CODE DSL, we have implemented also a tree-like graphical editor. In the following, we present the process followed to implement these two tree-like graphical editors.

This implementation is based on EMF and is composed of two tasks: 1) the **generation of a generator model (GenModel)** and 2) the **generation of the Java code** that implements the graphical editor.

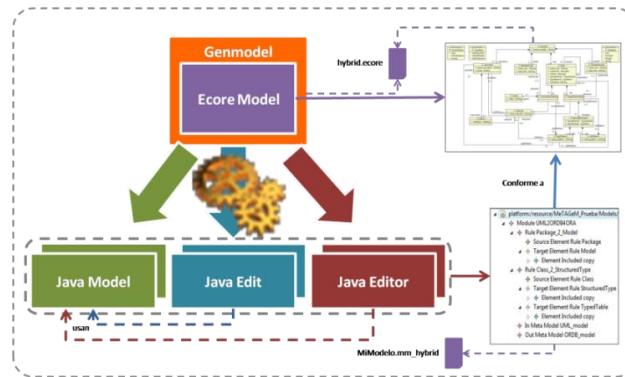
1. **Generation of a generator model (GenModel).** The GenModel is obtained from the implementation of the ASTM\_PHP metamodel as an Ecore model. The GenModel contains the essential data to generate the Java code that implements the tree-like graphical editor (classes, attributes, references, cardinalities among others). As we can see in Figure 6-23, the elements of the GenModel are related to the elements of the Ecore model. The main advantage of defining the Ecore model and the GenModel is that the Ecore model is independent to any information which is only required for the generation of code. However, the main disadvantage is the reference

desynchronization among these two models in case of changes within the Ecore model. To avoid this problem, the classes of the GenModel include methods to recognize these changes occurred in the Ecore assuring the synchronization between them.



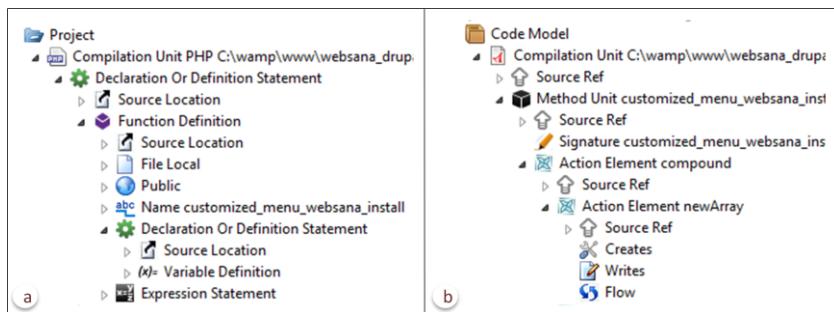
**Figure 6-23.** Relation between the GenModel and the metamodel.

2. **Generation of the Java code.** From the Ecore model and the GenModel obtained previously the Java code implementing the tree-like graphical editor is generated. This Java code is organized in three different and interrelated packages: the model code (Java Model), editing model (Java Edit) and the editor code (Java Editor). The *Java Model* package is the implementation of the metamodel in Java, considered the logic of the metamodel. It defines an API which allows to programmatically access the metamodel, the generation of models conforming to this metamodel, and the serialization of these models in XMI. The *Java Edit* and *Java Editor* packages use the *Java Model* package to generate Java code that implements the user interface of the tree-like graphical editor. The generation of these three packages is represented visually in Figure 6-24.



**Figure 6-24.** Overview of the generation of the Java code.

In Figure 6-25, we show two models represented with the tree-like graphical editors implemented. Figure 6-25.a represents an ASTM\_PHP model whereas the Figure 6-25.b represents a Code model. As we can observe, in both cases we have used the graphical notation defined in the concrete syntax presented in Section 5.2.2.2 for the ASTM\_PHP DSL and the Section 5.2.3.2 for the KDM\_CODE DSL.



**Figure 6-25. a) tree-like graphical editor for ASTM\_PHP DSL, b) tree-like graphical editor for KDM\_CODE DSL.**

### 6.3.2 UML-like Graphical Editor

We present the implementation of a graphical editor to create the models conforming to the CMS DSL. Unlike the graphical editors implemented for the ASTM\_PHP and KDM\_CODE DSLs presented in the previous section, this one represents the models with a UML appearance.

The implementation of this UML-like graphical editor is based on GMF. It is basically defined with three models: a *.gmfgraph* model that defines the graphical elements; a *.gmftool* model that defines the tool palette and a *.gmfmap* model that establishes the correspondences between the graphical elements, the tool palette and the elements of the metamodel. From these three models, GMF generates the Java code which implements the UML-like graphical editor.

Although, GMF is one of the most used frameworks to develop graphical editors, it presents some constraints and drawbacks because of its non-intuitive use, besides it can produce incomplete results, so that the developer must check the implementation of the graphical editor comprehensively to complete it (Epsilon, 2012). To prevent these drawbacks, the Epsilon project (Eclipse Foundation, 2012) proposes EuGENia. EuGENia is a tool that implements automatically efficient graphical editors simplifying the development of graphical editors based on GMF (Kolosovs et al., 2008).

The development process followed for the implementation of the UML-like graphical editor for the CMS DSL by using EuGENia is composed of three tasks:  
**1) Annotation of the CMS Common Metamodel with GMF annotations, 2) Customization of the graphical notation by using EOL and 3) Implementation of the UML-like graphical editor.** This process is presented in Figure 6-26. In next sections, we explain in detail these three tasks.

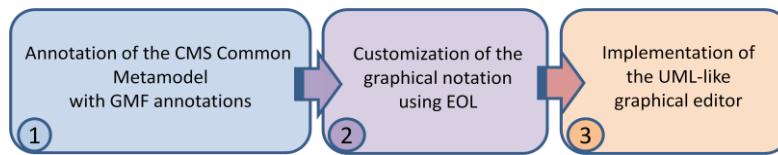


Figure 6-26. Process to define the UML-like graphical editor for the CMS DSL.

### Annotation of the CMS Common Metamodel with GMF Annotations

The first task in the generation of the UML-like graphical editor using EuGENia is the insertion of GMF annotations within the CMS Common Metamodel. These annotations determine which elements will be modelled within the graphical editor and their type (node, link or compartment).

To facilitate this annotation task, we have represented the CMS Common Metamodel in a textual manner. To obtain this textual representation we have implemented it in Emfatic (Bigeardel & Garcia, 2012) by means of a plugin installed in Eclipse. Figure 6-27 shows how to extract the Emfatic description from the Ecore model that implements the CMS Common Metamodel by using this plugin. From the pop-up menu displayed, it is necessary to select the option *Generate the Emfatic Source*.

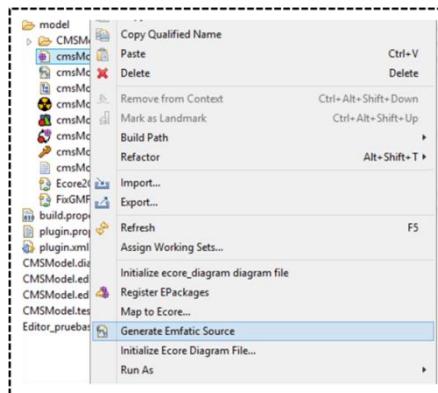


Figure 6-27. Extraction of the Emfatic implementation.

After obtaining the Emfatic implementation, we added the GMF annotations. Below, we present some of the existing GMF annotations that are used in the sample presented later.

- **@gmf.diagram:** indicates the root element of the model.
- **@gmf.node:** determines classes as nodes within the UML-like graphical editor.
- **@gmf.link:** defines references as edges within the UML-like graphical editor.
- **@gmf.compartment:** defines aggregations as containers of other elements within the UML-like graphical editor.

Figure 6-28 shows the CMS Common Metamodel implemented in Emfatic and enriched with the GMF annotations.

```

1  @gmf
2  @namespace(uri="cmsModel", prefix="cms"
3  package cmsModel;
4  @gmf.diagram
5  class CMSCommonModel {
6      attr String name;
7      ref Menu default_menu;
8      val Theme[*] themes;
9      val User[*] user;
10     val RoleGroup[*] roleGroup;
11     val Vocabulary[*] vocabulary;
12     val Category[*] category;
13     val Module[*] #model modules;
14     val Permissions[*] permissions;
15     val Language [*] languages;
16 }
17 @gmf.node(figure="figures.TermFigure",
18 label="name",label.pattern=":{0}")
19 class Term {
20     attr int ~id;
21     attr String name;
22     attr String description;
23     attr int weight;
24     @gmf.link (label=<tagged>")
25     ref Content [*] termContent;
26     @gmf.link (label=<term-parent>")
27     ref Term parentTerm;
28 }
29 @gmf.node(figure=rectangle,
30 label="name", label.pattern=":{0}")
31 class Theme {
32     attr boolean enable;
33     attr String logo;
34     attr int backgroundColor;
35     attr int textColor;
36     attr boolean core;
37     @gmf.compartment
38     val Region[*] #regionTheme themeRegion;
39     ref CMSCommonModel model;

```

**Figure 6-28. Emfatic implementation including GMF annotations.**

As we can see in Figure 6-28, the GMF annotation (`@gmf.node`) defined on lines 17-18 determines the *Term* class as a node. As we can see, this GMF annotation is defined by three attributes *figure*, *label* and *label.pattern*. The attribute *figure* represents the shape of the node (*rectangle*, *ellipse*, *rounded*, *svg* or *polygon*). In this case, we invoke a Java class called *TermFigure* which represents a *jpg* image. The attribute *label* determines the name of the *Term* class. The attribute *label.pattern*, in turn, defines the way the name of the class will be displayed. In this case, the name will be displayed along with the stereotype `<Term>`.

The GMF annotation (`@gmf.link`) added on line 24 defines the reference as an edge between a *Term* object and a *Content* object. This edge has a label with a default text that represents its stereotype (`<tagged>`).

Finally, the GMF annotation (`@gmf.compartment`) used on line 37 defines the aggregation as a container of *Region* objects.

By adding GMF annotations within the metamodel, it is possible to customize the graphical notation of the elements i.e. *border.colour* and *border.colour*, *size*. It entails the pollution of the metamodel with information that is not proper to the abstract syntax (Kolosovs et al., 2010). Hence, to pollute as little as possible the metamodel, we decided to define this graphical notation in a separate file implemented in EOL. The use of separate files allows their reutilization for other GMF projects. In the next section, we explain how to customize the graphical notation of the elements using EOL.

### **Customization of the Graphical Notation Using EOL**

With the customization of the graphical notation using EOL, we complete the definition of the concrete syntax specified for the CMS DSL presented in Section 5.2.4. An EOL project is composed of a set of *polishing transformations* which are executed at runtime. We present them below:

- **Ecore2GenModel.eol:** it alters the generated EMF *genmodel* and accesses the Ecore metamodel.
- **FixGenModel.eol:** it alters the generated EMF *genmodel* after the synchronization a of the EMF *genmodel*.
- **FixGMFGen.eol:** it alters the *gmfgen* model of GMF establishing dependencies to other plugins.

- **Ecore2EMF.eol:** it alters the *gmfgraph*, *gmfmap* and *gmftool* models from GMF to implement the graphical notation of the elements of the DSL using the language EOL.

For the current customization, we have chosen the last two *polishing transformations*. We present their implementation in the next two sections.

#### **Implementation of the FixGMFGen polishing transformation**

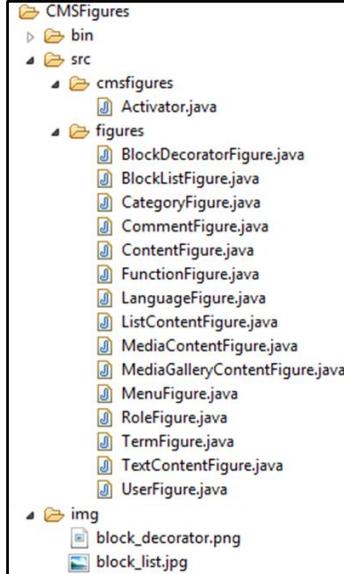
The implementation of this *polishing transformation* allows defining the graphical notation of an element of the CMS DSL by linking an image (.jpg or .png) with the element. It is possible by means of a plugin that is created and invoked by the *FixGMFGen* transformation. To create this plugin (called *CMSFigures*) we need to implement it by using the EOL code presented in Figure 6-29.

```
var plugin := GmfGen!GenPlugin.all.first;
plugin.requiredPlugins.add('CMSFigures');
```

Figure 6-29. Code to create the plugin in *FixGMFGen transformation*.

The *CMSFigures* plugin is created by defining two packages. One package defines the implementation of the classes that generate the images linked to the CMS elements and the other one stores the implementation of the class *Activator*; a class which extends the *AbstractUIPlugin* class and provides methods for loading those images. On the other hand, there is a directory where all these images are stored.

Figure 6-30 shows the physical structure of the *CMSFigures* plugin. The package *figure* contains the implementation of the classes generating the images whereas the package *cmsfigures* contains the implementation of the class *Activator*. Finally, the directory *img* contains the images whose format can be *.png* or *.jpg*.



**Figure 6-30. Physical structure of *CMSFigures* plugin.**

We will use the element *Term* to explain how its graphical notation is defined by the link to an image. As we presented in Figure 6-28, the class *Term* in the Emfatic is associated with the GMF annotation `@gmf.node` to define it as a node. As we can see, its attribute *figure* (`@gmf.node.figure`) defines the graphical notation of this node by the call to the class *TermFigure* which is implemented in the package *figures* of the *FixGMFGen* transformation. The implementation in Java of the class *TermFigure* is presented in Figure 6-31.

As we can see in Figure 6-31, on line 1 the package *figures* is created, whereas on line 3 the class *Activator* (its implementation is explained later in this section) is imported. On the other hand, on line 5 the class *TermFigure* is defined as an extension of the class *ImageFigure*. On lines 6-9 the constructor *TermFigure* is implemented. This constructor calls the method *createImage* from the class *Activator* indicating the location of the image where is stored (*img/term.png*).

```

1 package figures;
2 import org.eclipse.draw2d.ImageFigure;
3 import cmsfigures.Activator;
4
5 public class TermFigure extends ImageFigure {
6     public TermFigure () {
7         super(Activator.imageDescriptorFromPlugin(Activator.PLUGIN_ID,
8             "img/term.png").createImage(), 0);
9     }
10 }
```

**Figure 6-31.** Implementation in Java of *TermFigure* class.

Figure 6-32 is an excerpt of the code that implements the class *Activator*. On line 1, the *cmsfigure* package is created and on lines 3-4 the libraries needed to carry out this implementation are imported. On lines 6-25, the class *Activator* is implemented as an extension of the class *AbstractUIPlugin*. Lines 8-10 define the required attributes, whereas on lines 12-24 the methods composing the class are implemented.

```

1 package cmsfigures;
2
3 import org.eclipse.ui.plugin.AbstractUIPlugin;
4 import org.osgi.framework.BundleContext;
5
6 public class Activator extends AbstractUIPlugin {
7
8     public static final String PLUGIN_ID = "CMSFigures";
9
10    private static Activator plugin;
11
12    public Activator() {
13    }
14    public void start(BundleContext context) throws Exception {
15        super.start(context);
16        plugin = this;
17    }
18    public void stop(BundleContext context) throws Exception {
19        plugin = null;
20        super.stop(context);
21    }
22    public static Activator getDefault() {
23        return plugin;
24    }
25 }
```

**Figure 6-32.** Implementation in Java of the *Activator* class.

### Implementation of the Ecore2EMF Polishing Transformation

The implementation of this *polishing transformation* allows customizing the graphical notation of the elements of the CMS DSL by using the language EOL. This language allows accessing the GMF models (*gmfgraph*, *gmftool* and *gmfmap*) to customize it. This implementation is based on the specification of the concrete syntax presented in Section 5.2.4.

To explain the customization of the graphical notation by using EOL, we will use two classes, the class *Term* and the class *Theme*, both appearing in the Emfatic representation in Figure 6-28. As for the class *Term*, we present the EOL code to customize its associated label and the EOL implementation to customize its reference *termContent* which relates the class *Term* with the class *Content*. According to the class *Theme*, we explain the implementation in EOL of the graphical notation of the node that represents the *Theme* and the customization of its aggregation relationship with the class *Region*.

For the customization of the label of the class *Term* (defined in Figure 6-28 with the GMF annotation *label.pattern* = “<theme>:{0}”), we have implemented the EOL code presented in Figure 6-33.b. Otherwise, in Figure 6-33.a we present the resulting customization of the class *Term*.

<b>&lt;Term&gt;:name</b> <b>ABC</b>	<pre> 1 var termLabel = GmfGraph!Label.all. 2 selectOne(l l.name = 'TermLabelFigure'); 3 termLabel.font = new GmfGraph!BasicFont; 4 termLabel.font.style = GmfGraph!FontStyle#BOLD; 5 termLabel.font.height=9; 6 7 var diagramLabel = GmfGraph!DiagramLabel.all. 8 selectOne(l l.name='TermLabel'); 9 diagramLabel.elementIcon=false; </pre>
--	--

Figure 6-33. a) Customized graphical notation of the Term element b) EOL code to customize the label associated to the class Term.

This customization is focused on the font and on the icon associated to the label.

- **Customization of the font:** On line 4 we have defined the style of the font as bold, otherwise on line 5, we have defined the size of this font with 9 points.
- **Customization of the icon:** The label can be displayed along with an icon that depicts the element to which is associated. On line 9, we have decided not to display any icon with the label.

As for the customization of the reference *termContent* of the class *Term* (defined in Figure 6-28 with the GMF annotation @gmf.link (link=<tagged>)), which allows the association of a *Term* with a *Content*, we have implemented the EOL code presented in Figure 6-34.b. Otherwise, in Figure 6-34.a we present the resulting customization of the this reference.

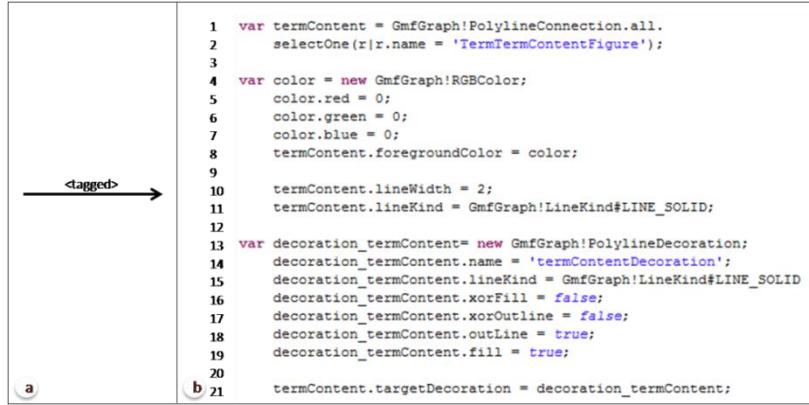


Figure 6-34. a) Customized graphical notation of the termContent link, b) EOL code to customize the reference termContent of the class Term.

With the code presented in Figure 6-34, we customize the colour, the width, the line style and decorations of the link that represents this reference. In this case, the label associated to this reference is not customized, so that it is displayed with the default features.

- **Customization of the colour:** On lines 4-8, we define the colour of the link. As we can see, to define the colour it is necessary to create an object *RGBColour*.
- **Customization of the width and line style:** On lines 10-11, the width of the link is defined with the value of 2, as well as the style of the link is defined as a solid line.
- **Customization of the decoration:** The decoration defined for this link is an arrow whose customization is presented on lines 13-21.

After explaining the implementation in EOL of the graphical notation of the class *Term* (its label and one of its references), we explain the customization of the class *Theme* and its aggregation relationship called *themeRegion*.

For the customization of the class *Term* (defined in Figure 6-28 with the GMF annotation @gmf.node), we have defined the EOL coded presented in Figure 6-35.b. The final graphical notation for the class *Theme*, is presented in Figure 6-35.a.

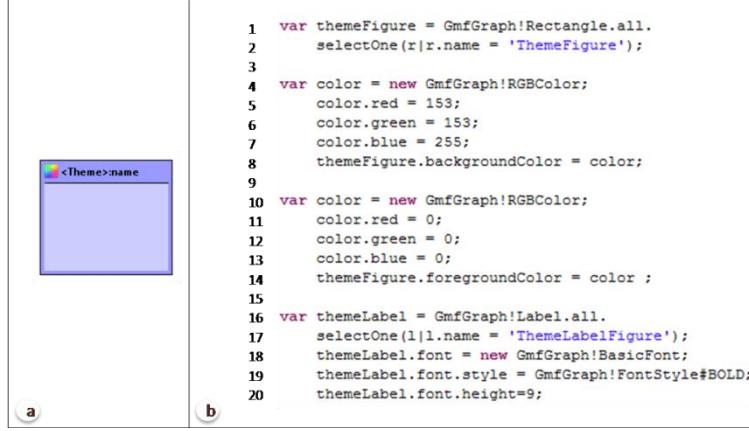


Figure 6-35. a) Customized graphical notation of the *Theme* class, b) EOL code to customize the class *Theme*.

This customization is centred in three aspects, the customization of the background colour, the border of the figure and the customization of the font of the label associated to the *Theme*.

- **Customization of the background colour:** On lines 4-8, we define the colour of the background of the figure. As occurred previously, it is necessary to create a *RGBColour* object.
- **Customization of the border:** On lines 10-14, the colour of the border of this figure is specified. For that, we defined another *RGBColour* object.
- **Customization of the font of the label:** On lines 17-20, we customize the font of the label. We define its style and its size.

Finally, in Figure 6-36.b we present the EOL code for the customization of the aggregation relationship *themeRegion* defined in the class *Theme* (defined in Figure 6-28 with the GMF annotation @gmf.compartment). This aggregation relationship defines the area where the *Region* elements will be located within the *Theme* element. This area is marked in red in Figure 6-36.a.



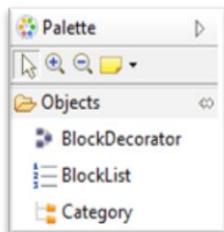
**Figure 6-36. a)** Customized graphical notation of the compartment of the Theme class, **b)** EOL code to customize the compartment of the class Theme.

This customization is focused on the background colour and the border colour.

- **Customization of the background colour:** On lines 4-8, we define the colour of the background of for this area.
- **Customization of the border:** On lines 10-14, the colour of the border of this area is specified.

Apart from the customization of the graphical notation of the elements of the CMS DSL that will be created with our UML-like graphical editor, it is possible to customize the toolbar of this editor.

We have customized this toolbar assigning to each type of element an specific icon replacing the GMF default icon (◆). For instance, for the class Term we have selected the icon ↗, for the class Theme the icon 🌈 and finally for the reference *termContent* the icon ➔. Figure 6-37, shows this toolbar.



**Figure 6-37.** Toolbar of the UML-like graphical editor.

### Implementation of the UML-like Graphical Editor

Finally, EuGENia launches a set of automated transformations which allow the generation of the UML-like graphical editor implemented in Java. It uses the annotations defined within the metamodel and the EOL transformations defined previously. Figure 6-38 is an overview of this graphical editor. As we can observe the toolbar is structured in *Objects* and *Connections*.

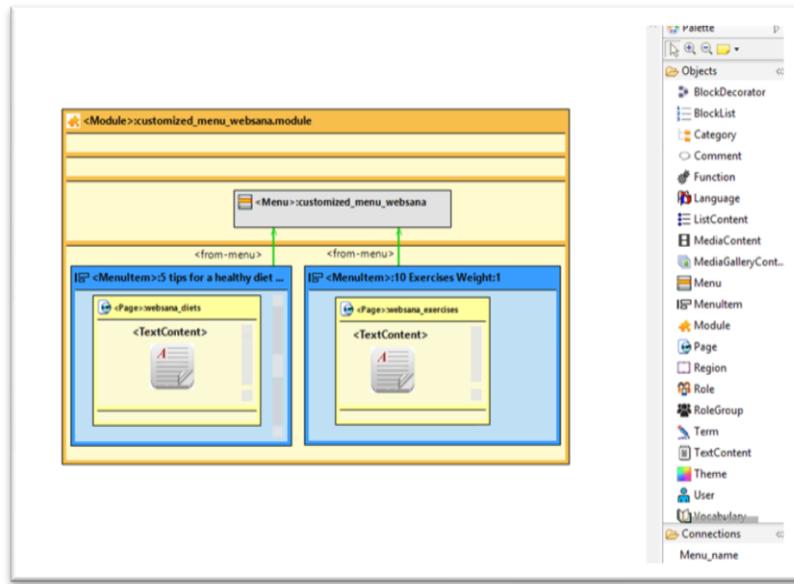


Figure 6-38. UML-like graphical editor for the CMS DSL.

### 6.4 Implementation of the Automated Transformations

In this section, we explain the implementation of the automated transformations defined between the different modelling levels proposed by the *ADMigaCMS* method.

The L0-to-L1 T2M transformation is implemented as a model extractor defined in Java; all the M2M transformations are implemented by transformation rules defined in ATL and finally, the L1-to-L0 T2M transformation is implemented with transformation rules implemented in Acceleo.

Unlike we did in the Chapter 5 where we specified the automated transformations by definig the mappings between the different modelling levels, in

the following sections we present the implementation of each automated transformation.

#### 6.4.1 L0-to-L1 T2M Transformation

The L0-to-L1 T2M transformation of the *ADMigracMS* method allows the extraction of an ASTM\_PHP model from the PHP code. The mappings of these transformations have been defined in Section 5.3.1.

These transformations have been implemented by a custom model extractor defined in Java. This implementation is based on a PHP parser and on an ASTM\_PHP API. The former recognizes the PHP syntax from a PHP code and the latter creates the elements of the ASTM\_PHP model. Figure 6-39 shows graphically this process.

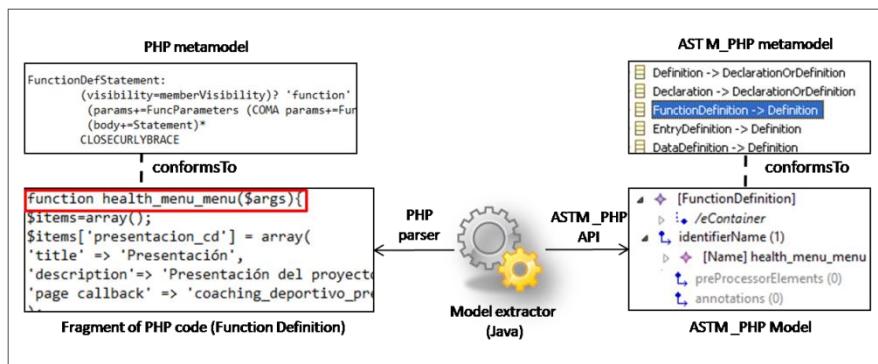


Figure 6-39. Implementation of the model extractor.

The PHP parser is obtained automatically from the PHP metamodel by using the framework Xtext. Otherwise, the ASTM\_PHP API is generated automatically from the GenModel model obtained during the implementation of the tree-like graphical editor of the ASTM\_PHP DSL (see Section 6.3). Both the PHP parser and the ASTM\_PHP API are implemented in Java.

To implement this model extractor in Java we have created three classes: *Principal*, *EventHandler* and *Extractor*:

- The **Principal** class generates a user interface (UI) which allow to select the directory where the PHP code is located and to launch the T2M transformation.
- The **EventHandler** class manages the events launched from the UI.

- The **Extractor** class contains the logic used to map the PHP elements from the code to ASTM\_PHP elements in the ASTM\_PHP model.

In next sections, we explain the implementation of these classes. In the following, we explain the implementation of the *Principal* class, the *EventHandler* class implementation and the implementation of the *Extractor* class.

### Implementation of the Principal Class

As we mentioned previously, this class creates the UI which allows selecting a PHP file and to launch the model extractor to obtain the ASTM\_PHP model. Figure 6-40.a presents this UI and Figure 6-40.b the header for this *Principal* class. As we can see, this class extends the *JFrame* class which allows building a window.

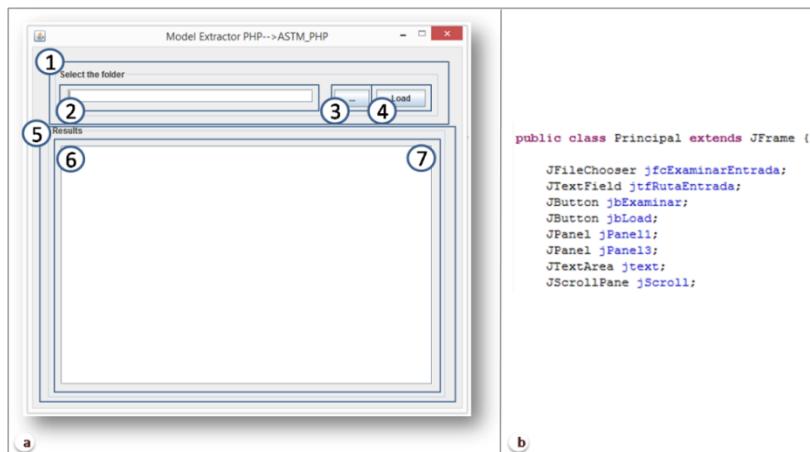
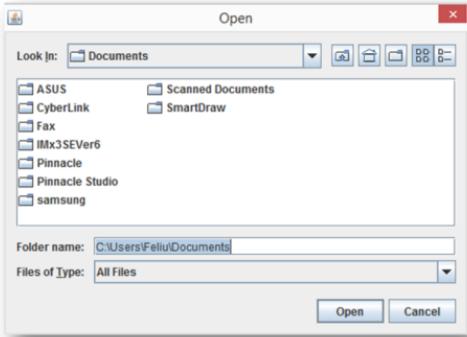


Figure 6-40. a) Model extractor UI, b) Implementation of the *Principal* class.

As we can see in Figure 6-40.b, in the implementation of the *Principal* class a set of attributes are defined. Below, we present them:

- **JfcExaminarEntrada:** its type is *JFileChooser* and it represents the window displayed to select the directory where the PHP files are located and from which we want to extract the ASTM\_PHP model (see Figure 6-41).



**Figure 6-41.** Resulting window for the *JFileChooser* object.

- **JbPanel1:** its type is *JPanel* and it represents the container of the other UI elements (see Figure 6-40.1).
- **JtfRutaEntrada:** its type is *JTextField* and it represents the input where the path and the name of the selected directory is displayed (see Figure 6-40.2).
- **JbExaminar:** its type is *JButton* and it represents the *Examine* button which launch the *JFileChooser* (see Figure 6-40.3).
- **JbLoad:** its type is *JButton* and it represents the *Load* button which launch the logic of the model extractor (see Figure 6-40.4).
- **JbPanel3:** its type is *JPanel* and it represents the container which contains other user interface elements (see Figure 6-40.5).
- **Jtext:** its type is *JTextArea* and it represents the text area where the results of launching the model extractor are displayed (see Figure 6-40.6).
- **Jscroll:** its type is *JScrollPane* and it represents the scroll which appears when the results exceed the previous text area (see Figure 6-40.7).

After defining the elements of this UI, we create an instance of the class *EventHandler* which we assign it to the buttons *Examine* and *Load* by means of an *ActionListener*, as we can see in Figure 6-42. When these buttons are clicked the object *EventHandler* is activated.

```
EventHandler eventHandler =new EventHandler(this);
jbExaminar.addActionListener(eventHandler);
jbLoad.addActionListener(eventHandler);
```

**Figure 6-42.** Adding an *ActionListener* object to the buttons.

### Implementation of the EventHandler Class

The *EventHandler* class processes the events launched by the two buttons defined in the previous UI and from these events allows the *Extractor* object to parser the PHP elements to ASTM\_PHP elements.

Figure 6-43 shows an excerpt of the Java code implementing this class. We observe on line 1 that the event launched by the button *Load* is managed by a condition and on line 4 we create an instance of the class *Extractor* which contains all the required methods to transform the PHP elements into ASTM\_PHP elements.

```

1 if (evento.getSource() == ventana.jbLoad) {
2     if (projectDirectory != null) {
3
4         Extractor test = new Extractor();
5         GastmPackage.eINSTANCE.eClass();
6
7         Project targetModel = test.gastmFactory.createProject();
8         test.getModuleFiles(projectDirectory);

```

**Figure 6-43. Excerpt of the EventHandler class implementation.**

At the end of the execution of this class, the ASTM\_PHP model is resulted in XMI (OMG, 2011). If it has been created correctly the pop-up presented in Figure 6-44 is displayed to the user.



**Figure 6-44. Windows displayed when ASTM\_PHP model is created correctly.**

On the other hand, if any error occurred during the T2M transformation a set of pop-ups can be displayed depending on the error. Figure 6-45 shows the pop-up in the case any PHP is not found to start with the T2M transformation.



**Figure 6-45. Windows displayed when the Php files are not found.**

## Implementation of the Extractor Class

This class is composed of a set of methods which allow to map the PHP elements to elements of the ASTM\_PHP model. In Table 6-8, we present briefly all these methods. We have classified them in two types:

- The **generator**: This type corresponds to all those methods which create an element in the ASTM\_PHP model.
- The **processor**: This type corresponds to those methods which enable the matching between the PHP elements and the elements of the ASTM\_PHP model.

**Table 6-8. Implementation of L0-to-L1 T2M transformation.**

TYPE	METHOD	DESCRIPTION
Generator	createDefinition	It generates elements of type <i>DefinitionObject</i> of the ASTM_PHP metamodel.
	createSource Location	It generates elements of type <i>SourceLocation</i>
	createName	It generates elements of type <i>Name</i> .
	createNamed TypeReference	It generates elements of type <i>NamedTypeReference</i> .
	createUnnamed TypeReference	It generates elements of type <i>UnnamedTypeReference</i> .
	getBinary Operator	It generates elements of type <i>BinaryOperator</i> .
	getUnary Operator	it generates elements of type <i>UnaryOperator</i> .
	getCastType	It generates elements of type <i>TypeReference</i> .
	getIncrement Operator	it generates elements of type <i>UnaryOperator</i> depending on the increment operator.
	getFormalParametersDefinition	it generates elements of type <i>FormalParameterDefinition</i> .
	getObjectAccess	it generates elements of type <i>ObjectAccess</i> .
	getAccessKind	it generates elements of type <i>AccessKind</i> .
	getKey:	it generates elements of type <i>Key</i> .
	getAssign Operator:	it generates an element of type <i>BinaryOperator</i> depending on the assign operator.
Processor	processBinary Expression	This method processes the binary expressions from the PHP code and transforms them into <i>BinaryExpression</i> elements within the ASTM_PHP model.
	processUnary Expression:	This method processes the unary expressions from the PHP code and transforms them into <i>UnaryExpression</i> elements within the ASTM_PHP model.
	Process Expression:	This method processes the different expressions from the PHP code and transforms them into elements within the ASTM_PHP model.
	Process Statement	This method processes the different sentences found in the PHP code used as input and transforms them into <i>Statement</i> elements into the ASTM_PHP model.
	getModulesFiles	It lists all the files with .module extension located in a certain path. These .module files contain the PHP code of the Drupal-based Web applications.

Figure 6-46 presents an excerpt of the Java code implementing the method *ProcessStatement*. This fragment of code map a *VariableDefStatement* (on line 1) from the PHP code to an *ExpressionStatement* (on lines 19-20) in the ASTM\_PHP model.

```

1  if (statementPhp instanceof VariableDefStatement){
2      VariableDefStatement assignment=(VariableDefStatement)statementPhp;
3      astm_php.gastm.Expression nameExpression=
4          processExpression(assignment.getLeft(),file);
5      astm_php.gastm.Expression expRightProcess=
6          processExpression(assignment.getRight(),file);
7      String operatorAssignment.getOperator();
8
9      astm_php.gastm.BinaryExpression binaryExp=
10         gastmFactory.createBinaryExpression();
11     binaryExp.setLocationInfo(createSourceLocation(file));
12     binaryExp.setExpressionType(createNamedTypeReference(file, false));
13     binaryExp.setLeftOperand(nameExpression);
14     binaryExp.setRightOperand(expRightProcess);
15     astm_php.gastm.BinaryOperator binaryOperator=
16         getBinaryOperator(operator,file);
17     binaryExp.setOperator(binaryOperator);
18
19     astm_php.gastm.ExpressionStatement statement=
20         gastmFactory.createExpressionStatement();
21     statement.setLocationInfo(this.createSourceLocation(file));
22     statement.setExpression(binaryExp);
23
24 }
```

Figure 6-46. Excerpt of *processStatement* method.

#### 6.4.2 L1-to-L2 M2M Transformation

In this section, we explain the implementation in ATL of the L1-to-L2 M2M transformation which allow transforming an ASTM\_PHP model to a Code model. The specification of the mappings for these M2M transformation has been presented in Section 5.3.2.

We have defined a set of transformation rules implemented in ATL. ATL specifies three types of transformation rules: *mapped rules*, *called rules* and *lazy rule*.

- The **mapped rules** are those which are launched at the time that the M2M transformation is executed. This type of rules defines a *source pattern* which represents an element from the source model and the *target pattern* the element generated in the target model.
- The **called** and **lazy rules** are launched from a mapped rule. The main difference between both is that the former do not need the definition of a *source pattern*, whereas the latter does.

These transformation rules are unidirectional, it means that the source models are read-only and the target models are write-only.

Some of the transformation rules implementing this M2M transformation are presented in Table 6-9. The rest of transformation rules can be seen in the complete implementation of the *ADMigraCMS* toolkit in the CD attached.

**Table 6-9. Implementation of L1-to-L2 M2M transformation.**

TYPE	NAME RULE	DESCRIPTION
Helper	getName	It takes the string from a <i>Name</i> instance.
	getExportKind	It maps the values of the enumerated type <i>AccessKind</i> to the the enumerated type <i>ExportKind</i>
	getFunctionSignature	It returns the <i>Signature</i> element from the name of a <i>MethodUnit</i>
	getParameterUnit	It returns the <i>ParameterUnit</i> elements from a <i>Signature</i> .
	getStorableUnit	It returns a <i>StorableUnit</i> whose name meets a certain string.
Called Rules	createSignature	It creates a <i>Signature</i> element from a string and a set of <i>FormalParameterDefinition</i> elements.
	createParameterUnit	It creates a <i>ParameterUnit</i> element from a <i>FormalParameterDefinition</i> element.
	createSourceFile	It creates a <i>SourceFile</i> .
	createSourceRegion	It creates a <i>SourceRegion</i> .
	CreateSourceRef	It creates a <i>SourceRef</i> .
Matched rules	Project2CodeMode1	It creates an <i>CodeModel</i> element from a <i>Project</i> element.
	CompilationUnitPHP2CompilationUnitKDM	It creates an <i>CompilationUnit</i> element from a <i>CompilationUnitPHP</i> element.
	ClassDefinition2Action	It creates a <i>ClassUnit</i> from a <i>DeclarationOrDefinitionStatement</i> element.
	functionDefMember2MethodUnit	It creates a <i>MethodUnit</i> from a <i>FunctionDefinition</i> element which is a member of a class.
	functionDef2MethodUnit	It creates a <i>MethodUnit</i> from a <i>FunctionDefinition</i> element.

This implementation is defined as a module called *astm2kdm*, as we can see in Figure 6-47. The source model (*IN*) defined at PSM level conforms to the ASTM\_PHP metamodel and the target model (*OUT*) defined at PIM level conforms to the KDM metamodel.

```
module astm2kdm;
create OUT: kdm from IN: astm_php;
```

**Figure 6-47. Astm2Kdm module.**

The implementation of the matched rule *functionDef2MethodUnit* is presented in Figure 6-48. This rule implements the mapping defined between the

ASTM\_PHP class *DeclarationOrDefinitionStatement* and the KDM class *MethodUnit*, presented in Table 5-35. This mapping is defined with the condition (*If declOrDefn = FunctionDefinition*).

```

1 rule functionDef2MethodUnit {
2   from
3     funcdef: astm!DeclarationOrDefinitionStatement
4     (funcdef.isFunctionDefinition)
5   to
6     meth: kdm!MethodUnit (
7       name <- funcdef.declOrDefn.identifierName.getName.
8       codeElement <- thisModule.createSignature(funcdef.
9       source <- thisModule.createSourceRef(funcdef.declO
10      export <- funcdef.declOrDefn.accessKind.getExportK
11      codeElement<-funcdef.declOrDefn.body

```

**Figure 6-48.** *functionDef2MethodUnit* matched rule in ATL.

As we can see in Figure 6-48, the *source pattern* of this matched rule is defined on the class *DeclarationOrDefinitionStatement* (on line 3) and the *target pattern* on the class *MethodUnit* (on line 6). The condition is implemented with a call to the helper *isFunctionDefinition* (on line 4). As we can see in the *target pattern*, a *Signature* is created (on line 8) by calling the called rule *createSignature*.

#### 6.4.3 L2-to-L3 M2M Transformation

We present the implementation in ATL of the L2-to-L3 M2M transformations of the *ADMigracMS* method. This transformation generates a CMS model from a Code model. The main objective of this transformation is to represent the PHP code in the CMS domain. The specification of the mappings for these M2M transformations have been presented in Section 5.3.3.

The implementation in ATL of this M2M transformation is composed of a set of transformation rules. A set of them are presented in Table 6-10. The rest of transformation rules can be seen in the CD attached.

**Table 6-10. Implementation of the L2-to-L3 M2M transformation.**

TYPE	NAME RULE	DESCRIPTION
Helper	Model	It returns the root element of the CMS model.
	getProject	It returns the element Project from a KDM model.
	getAllDirectories	It returns a list of directories from the element Project of the Inventory model.
	Project	
	getSpecific	It returns a specific directory from the element Project of the DirectoryProject.
	DirectoryProject	

	getAllDirectories Directory	It returns a list of directories from a directory of the Inventory model.
Matched rules	InventoryModel2DrupalCMSModel	This rule matches the root element of the Inventory model to the root element of the CMS model.
	MethodUnit2Menu	This rule matches a method unit to a Menu.
	SourceFile2Module	This rule matches a SourceFile to a Module.
	ArrayReplace2Block	This rule matches an ActionElement with kind=ArrayReplace to a Block.
	ArrayReplace2MenuItem	This rule matches an ActionElement with kind=ArrayReplace to a MenuItem.

This implementation is defined as a module called *kdm2cms*, as we can see in Figure 6-49. The source model (*code*) conforms to the KDM metamodel and the target model (*cms*) conforms to the CMS Common Metamodel (*CMSCM*).

```
module kdm2cms;

create cms : CMSCM from code:KDM,
```

Figure 6-49. *Kdm2cms* module.

In the following, we present the implementation of the matched rule *methodUnit2Menu* whose code is presented in Figure 6-50. This rule implements the mapping defined between the class *MethodUnit* of the Code model and the class *Menu* of the CMS model.

As we can see in Figure 6-50, the *source pattern* refers to a *MethodUnit* and applies a condition called *isCreatingMenu* (on line 4). In this case, *IsCreatingMenu* is a helper that returns true if the *MethodUnit* contains a call to the method *menu\_save()* (this condition is defined in Table 5-39). On the other hand, the *target pattern* creates the *Menu* element and initializes it (on line 6). To initialize its attribute *name* we use the helper *getMenuName*, as well as to initialize the attributes *description* and *title*, we call the helpers *getMenuDescription* and *getMenuTitle* respectively.

```
1 rule MethodUnit2Menu {
2   from
3     kdm : KDM!MethodUnit
4     (kdm.isCreatingMenu())
5   to
6     cms : CMSCM!Menu(
7       name<-kdm.getMenuName,
8       event<-kdm.name,
9       id<-thisModule.count,
10      description<-kdm.getMenuDescription,
11      title<-kdm.getMenuTitle
12    )
13   do{
14     thisModule.count<-thisModule.count+1;
```

Figure 6-50. *MethodUnit2Menu* matched rule in ATL.

#### 6.4.4 L3-to-L2 M2M Transformation

This transformation generates a Code model from a CMS model. This Code model defines the code that will be run in the target CMS-based Web application. The implementation in ATL of this M2M transformation is composed of a set of transformation rules. A set of them are presented in Table 6-11, the rest of them can be consult in the CD attached.

**Table 6-11. Implementation of the L3-to-L2 M2M transformation.**

TYPE	NAME RULE	DESCRIPTION
Helper	getSignature	Return the Signature of a MethodUnit
	createVariableName	Returns a string which is the name of the variable.
	getParameterUnit	Returns an string which is the name of the parameter
	getMenuItemsFromMenu	Returns a sequence of menu items from a Menu
	getArrayReplace	Returns an ActionElement with the attribute kind defined as arrayReplace
Called rules	getCreateNavMenu	It creates an ActionElement with the attribute kind defined as call and whose name is wp_create_nav_menu
	createSourceRef	It creates an element SourceRef
	getNewArray	It creates an ActionElement with the attribute kind defined as new array
	getUpdateNavMenuItem	It creates an ActionElement with the attribute kind defined as call and whose name is wp_update_nav_menu_item
	getInsertPost	It creates an ActionElement with the attribute kind defined as call and whose name is wp_insert_post
Matched rules	Menu2MethodUnit	From the Menu element creates a MethodUnit element
	MenuItem2ActionElement	From the MenuItem element creates a ActionElement element
	Page2ActionElement	From the Page element creates a ActionElement element
	Module2CompilationUnit	From the Module element creates a CompilationUnit element
	cmsModel2CodeModel	From the CMSModel element creates a Model element

The implementation in ATL of this transformation is defined as a module called *cms2kdm*, as we can see in Figure 6-51. The source model (*cms*) conforms to the CMS Common Metamodel (*CMSCM*) and the target model (*kdm*) conforms to the KDM metamodel.

```
module cms2kdm;
create kdm:KDM from cms:CMSCM;
```

**Figure 6-51. cms2kdm module.**

In the following and considering the implementation of the matched rule explained in the previous section, we present the implementation in ATL of the

matched rule *menu2MethodUnit* whose code is presented in Figure 6-52. This rule implements the mapping defined between a the class *Menu* of the CMS model and the class *MethodUnit* of the Code model. This mapping is defined in Table 13-10. If the reader is interested in consulting the rest of these transformation rules, he/she can find them in the CD attached.

```

1 rule menu2MethodUnit{
2   from cms:CMSCM!Menu
3   to
4     kdm:KDM!MethodUnit(
5       name<-'create_menu_'+cms.name,
6       codeElement<-thisModule.getCreateNavMenu(cms,kdm),
7       source<-thisModule.createSourceRef(cms.getPathModuleElement),
8       codeElement<-cms.getMenuItemsfromMenu->collect(el|el.getPagefromMenuItem),
9       codeElement<-cms.getMenuItemsfromMenu->collect(el|el.getPagefromMenuItem)
10      ->collect(e12| thisModule.getInsertPost(e12, kdm)),
11      codeElement<-cms.getMenuItemsfromMenu,
12      codeElement<-cms.getMenuItemsfromMenu->
13      collect(el|thisModule.getUpdateNavMenuItem(el,kdm))

```

**Figure 6-52. *menu2ActionElement* matched rule in ATL.**

As we can see in Figure 6-52, the *source pattern* of the matched rule is defined on the class *Menu* (on line 2) and the *target pattern* on the class *MethodUnit* (on line 4). To create the *MethodUnit* we call a set of called rules such as *getCreateNavMenu* (on line 6) or *CreateSourceRef* (on line 7) as well as helpers such as *getMenuItemsfromMenu* (on lines 8-9).

#### 6.4.5 L2-to-L1 M2M Transformation

This transformation generates an ASTM\_PHP model defined at PSM level from a Code model at PIM level. We decrease the abstraction level since next task is the generation of the PHP code of the target CMS-based Web application. The implementation in ATL of this M2M transformation is composed of a set of transformation rules. A set of them are presented in Table 6-12, the rest can be seen in the CD attached.

**Table 6-12. Implementation of L2-to-L1 M2M transformation.**

TYPE	NAME RULE	DESCRIPTION
Helper	isBinaryExpression	Returns a Boolean value. It asses whether the expression is of type <i>BinaryExpression</i> .
	isMethodUnitStatement	Returns a Boolean value. It asses whether the statement is defines a <i>MethodUnit</i> .
	getCompoundType	Returns a sequence of <i>ActionElements</i> whose attribute kind is defined as compound.
	getActualParameters	Returns a sequence of <i>ByValueActualParameterExpression</i> elements.
	getLiterals	Returns an Expressio of type <i>Literal</i> .

	getName	It takes the string from a <i>Name</i> instance (nameString).
Called Rules	getBinaryExpression	Returns a <i>BinaryExpression</i> element.
	getCodeElementList	Returns a sequence of elements stored in the attribute codeElement of an <i>ActionElement</i> .
Lazy rules	getVariableDefinition	It generates a <i>VariableDefinition</i> from a <i>StorableUnit</i> .
	getFunctionDefinition	It generates a <i>FunctionDefinition</i> from a <i>MethodUnit</i> .
	getFormalParameter	It generates a <i>FormalParameterDefinition</i> from a <i>ParameterUnit</i> .
Matched rules	calls2FunctionCallExpression	It generates a <i>FunctionCallExpression</i> from an <i>ActionElement</i> whose attribute kind is defined as calls.
	Assign2VariableDefinition	It generates a <i>DeclarationOrDefinitionStatement</i> from an <i>ActionElement</i> whose attribute kind = assign
	methodUnit2Statement	It generates a <i>DeclarationOrDefinitionStatement</i> from an <i>MethodUnit</i> .

The implementation in ATL of this transformation is defined as a module of the toolkit called *kdm2astm*, as we can see in Figure 6-53. The source model (*kdm*) at PIM level conforms to the KDM metamodel and the target model (*astm*) defined at PSM level conforms to the ASTM\_PHP metamodel.

```
module kdm2astm;
create astm:ASTM from kdm:KDM;
```

Figure 6-53. *kdm2astm* module.

Considering the implementation of the matched rule explained in the previous section, we present the implementation in ATL of the matched rule *methodUnit2Statement*. This implementation is shown in Figure 6-54.

```
1 rule methodUnit2Statement{
2   from
3     kdm:KDM!MethodUnit
4     (kdm.isMethodUnitStatement)
5   to
6     astm:ASTM!DeclarationOrDefinitionStatement(
7       declOrDefn<-thisModule.getFunctionDefinition(kdm),
8       locationInfo<-thisModule.getLocationInfo(kdm)
9     )
```

Figure 6-54. *methodUnit2Statement* matched rule in ATL.

As we can see in Figure 6-54, the *source pattern* of the matched rule is defined on the class *MethodUnit* (on line 3) and the *target pattern* on the class *DeclarationOrDefinitionStatement* (on line 6). Also, there is a condition implemented with a helper that in this case is called *isMethodUnitStatement* (on line 4). The function definition is created with the call to the lazy rule

*getFunctionDefinition* (on line 7) whose ATL implementation is presented in Figure 6-55.

```

1 lazy rule getFunctionDefinition{
2   from
3     kdm:KDM!MethodUnit
4   to
5     astm:ASTM!FunctionDefinition(
6       locationInfo<-thisModule.getLocationInfo(kdm),
7       storageSpecifiers<-thisModule.getFileLocal(kdm),
8       accessKind<-thisModule.getPublic(kdm),
9       identifierName<-thisModule.getName(kdm.name, kdm),
10      opensScope<-thisModule.getOpenScope(),
11      body<-kdm.codeElement,
12      formalParameters<-
13        if kdm.getCodeElementList(KDM!Signature).notEmpty() then
14          kdm.getCodeElementList(KDM!Signature)->first().parameterUnit
15          ->collect(e1|thisModule.getFormalParameter(e1))
16        else
17          Sequence{}
18      endif

```

Figure 6-55. *getFunctionDefinition* lazy rule in ATL.

The *source pattern* of this lazy rule is defined on the class *MethodUnit* (on line 3) and the *target pattern* on the class *FunctionDefinition* (on line 5). The parameters defined within the *Signature* are mapped to the attribute *formalParameters* of the *FunctionDefinition* class.

#### 6.4.6 L1-to-L0 M2T Transformation

Finally, we explain the implementation of the L1-to-L0 M2T transformations of the *ADMigraCMS* method, which generates the PHP code that implements the target CMS-based Web application from the ASTM\_PHP model.

On the one hand, the specification of the mappings for these M2T transformation has been presented in Section 5.3.1. On the other hand, the implementation of this transformation has been carried out using the open-source language, Acceleo (Musset et al., 2008). Acceleo provides the mechanism for the code generation from models conforming to any kind of metamodel.

The implementation of the code generator is defined on a template-based approach. A template is a text containing a set of structures where the text will be computed from elements provided by the inputs models. Some of the structures composing a template are presented below:

- **If structure:** Acceleo template can use if structures to specify blocks of code that should be executed only if a given condition is fulfilled.
- **For structure:** A for structure is also available to iterate on collections. For structures are also defining a variable named "i". This variable is the iterator of the block. This variable is only accessible in the body of the *for* block and

in the separator expression. In the body expression, this variable would always have the value "1" and in the after expression, the value would always be the size of the expression initializing the iteration. Since this variable is representing the index of the element in the collection initializing the iterator of the loop, its value will always start at 1.

- **Let structure:** A let block can let you define a variable inside of your template.

Figure 6-56 presents the template *generateStatement*, a fragment of the implementation in Acceleo of the L1-to-L0 M2T transformation. This code generates the PHP code for an array definition from the ASTM\_PHP model.

```

1 [template private generateStatement(aStatement : Statement)]
2 [if (aStatement.oclIsTypeOf(ExpressionStatement))]
3 [generateExpression(aStatement.oclAsType(ExpressionStatement).expression)/]
4 [elseif(aStatement.oclIsTypeOf(DeclarationOrDefinitionStatement))]
5 [generateDeclaration(aStatement.oclAsType(DeclarationOrDefinitionStatement).declOrDefn)/]
6 [elseif(aStatement.oclIsTypeOf(ReturnStatement))]
7 [generateReturn(aStatement.oclAsType(ReturnStatement))/]
8 [/if]
9 [/template]
```

Figure 6-56. *GenerateStatement* template in Acceleo.

In the template *generateStatement* we can see how a *Statement* element from the ASTM\_PHP model (on line 1) is assessed with an *if structure* to known its type. In this case, the statement is of *ExpressionStatement* type (on line 2), where the template *generateExpression* is called. The Acceleo code implementing this template is presented in Figure 6-57.

```

1 [template private generateExpression(aExpression : Expression)]
2 [if (aExpression.oclIsTypeOf(FunctionCallExpression))]
3 [generateFunctionCallExp(aExpression.oclAsType(FunctionCallExpression))/]
4 [elseif (aExpression.oclIsKindOf(Literal))]
5 [generateLiteral(aExpression.oclAsType(Literal))/]
6 [elseif (aExpression.oclIsTypeOf(BinaryExpression))]
7 [generateBinaryExp(aExpression.oclAsType(BinaryExpression))/]
8 [elseif(aExpression.oclIsTypeOf(ObjectAccess))]
9 [generateObjectAccess(aExpression.oclAsType(ObjectAccess))/]
10 [elseif(aExpression.oclIsTypeOf(IdentifierReference))]
11 [generateIdRef(aExpression.oclAsType(IdentifierReference))/]
12 [elseif(aExpression.oclIsTypeOf(DuplaArray))]
13 [generateDuplaArray(aExpression.oclAsType(DuplaArray))/]
14 [elseif(aExpression.oclIsTypeOf(CollectionExpression))]
15 [generateArray(aExpression.oclAsType(CollectionExpression))/]
16 [elseif(aExpression.oclIsTypeOf(ClassAccess))]
```

Figure 6-57. *GenerateExpression* template in Acceleo.

The *generateExpression* template is used to assess the type of *Expression* defined within the *ExpressionStatement*. For that, an *if structure* is define, as we

can see in Figure 6-57. In this case, the expression is a *BinaryExpression* (on lines 6-7) where the template *generateBinaryExp* is called. The code of this template is presented in Figure 6-58.

```

1 [template private generateBinaryExp(aExpression : BinaryExpression)]
2 [generateExpression(aExpression.leftOperand)/]
3 [generateOperator(aExpression.operator)/]
4 [generateExpression(aExpression.rightOperand)/]
5 [/template]
```

**Figure 6-58.** *GenerateBinaryExp* template in Acceleo.

The *generateBinaryExp* is defined to generate the PHP code for a *BinaryExpression*. In this case, this *BinaryExpression* represents an array definition. As we can see in Figure 6-58, three template calls are defined to generate this code. The first one is a call to the *generateExpression* template (on line 2) to generate the PHP code for the left operand which in this case is an *IdentifierReference*; the second one is a call to the *generateOperator* template (on line 3) for the operator that in this case is an *Assign* operator and finally, the third one is another call to the *generateExpression* template (on line 4) for the right operand which is a *CollectionExpression* (array definition). In Figure 6-59, we present the code of the *generateIdRef* template called from the *generateExpression* (on line 11 in Figure 6-57) template for generating the PHP code of an *IdentifierReference* (left operand).

```

1 [template private generateIdRef(aExpression : IdentifierReference)]
2 [aExpression.name.nameString/]
3 [/template]
```

**Figure 6-59.** *GenerateIdRef* template in Acceleo.

The *generateIdRef* prints in the PHP code the name of the *IdentifierReference* element which is stored in its attribute *nameString*. In Figure 6-60, we present the *generateOperator* template.

```

1 [template public generateOperator(binary:BinaryOperator)]
2 [if (binary.oclIsTypeOf(Or))] Or
3 [elseif (binary.oclIsTypeOf(Assign))] =
4 [else] +
5 [/if]
6 [/template]
```

**Figure 6-60.** *GenerateOperator* template in Acceleo.

The *generateOperator* template prints the operator of the *BinaryExpression*. In this case, it is an assign operator so that the operator = is

printed in the PHP code. Finally, Figure 6-61 shows the *generateArray* template called from the *generateExpression* template (on line 19 in Figure 6-57) for generating the PHP code of an *CollectionExpression* (right operand).

```

1 [template private generateArray(aExpression : CollectionExpression)]
2 array(
3 [for (exp:Expression|aExpression.expressionList) separator(',') ]
4 [generateExpression(exp)/]
5 [/for])
6 [/template]
```

Figure 6-61. *GenerateArray* template in Acceleo.

The *generateArray* template prints the PHP code for an array definition. Thus, the token *array* and an *openbrace* (on line 2) is printed followed by a set of expressions separated by commas defining the array entries. These expressions are managed by a *for structure* (on lines 3-5). Each expression stored in the attribute *expressionList* (on line 3) of the *CollectionExpression* calls the *generateExpression* template.

## 6.5 Concluding Remarks

In this chapter, we have presented the implementation of the *ADMigracMS* toolkit that supports the *ADMigracMS* method.

This toolkit provides the *ADMigracMS* method with a set of **graphical editors** as well as it implements the **automated transformations**.

We have implemented two tree-like graphical editor for the ASTM\_PHP DSL and the KDM\_CODE DSL. To implement these **tree-like graphical editors** we have used EMF. These graphical editors allow developers to create and edit graphical models conforming to these two DSLs.

Otherwise, we have implemented a **UML-like graphical editor** for the CMS DSL. To carry out this implementation, we have needed to implement the **CMS Common Metamodel** as an Ecore model using EMF. To generate the code that implements this graphical editor, we have used EuGENia. This tool has allowed us to implement semi-automatically a graphical editor based on GMF.

As for the implementation of the automated transformations, we can say that the **L0-to-L1 T2M transformation** has been implemented as a **model extractor** defined in Java. For this implementation, we needed a parser to read code written in PHP. This parser has been obtained from the implementation of

the **PHP metamodel** using the Xtext framework. This implementation has been defined with a set of **parser rules**. Otherwise, we needed an **API of the ASTM\_PHP metamodel** to generate the elements of the ASTM\_PHP model. This API has been obtained from the implementation in Ecore of the ASTM\_PHP metamodel.

All the **M2M transformations** have been implemented by a set of transformation rules defined in **ATL**.

Finally, the **L1-to-L0 M2T transformation** has been implemented as a model generator implemented with **Acceleo**.



## *Chapter 7: Validation*

---



In this chapter we present the validation of the research presented in this PhD Thesis according to the validation method described in Section 2.3.2.

This validation has been carried out considering two case studies: one is based on a CMS-based Web application that manages a coaching centre (Coaching Web) and the other one is a CMS-based Web application for a wellness and nutrition centre (Websana).

In Section 7.1, we explain the validation method followed to execute the validation of this PhD Thesis. In Section 7.2, we explain the process to select the case studies used in this validation. In Section 7.3, we present the sources considered to execute the validation and the questions used to retrieve data from the execution of this validation. Section 7.4 presents the design and the execution of the case studies and finally, Section 7.5 presents the final conclusions extracted from the data retrieved from the previous questions.

## 7.1 Validation Method

The validation method applied for this PhD Thesis is composed of four tasks as we explained in Section 2.3.2. Figure 7-1 shows the tasks of this validation method.

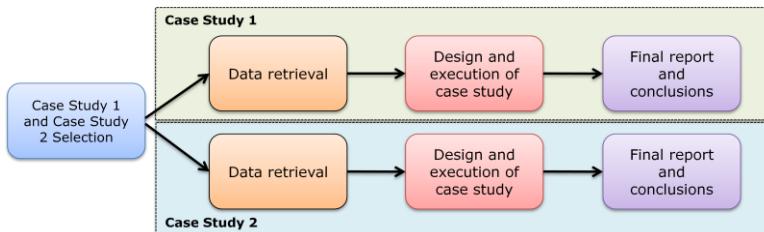


Figure 7-1. Tasks of the validation method.

- **Case study 1 and case study 2 selection:** during this task we have selected the two case studies used to carry out this validation.
- **Data retrieval:** for each case study we define the sources from which we extract the data during the design and execution task. These sources are the PHP code implementing the CMS-based Web applications being migrated. Moreover, we define a set of questions which allow this data retrieval.

- **Design and execution of the case study:** for each case study, we execute the migration process.
- **Final report and conclusions:** at the end of the execution task, we gather all the data and information extracted from the questions defined previously and we reach to the final conclusions and write them in a final report.

Executing this validation method, we are interested in validating the following points:

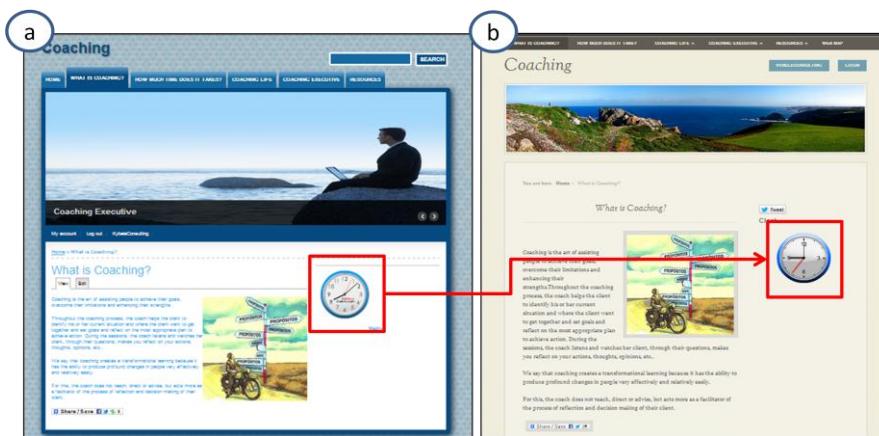
- The **correct specification of the metamodels.** We want to validate the three metamodels defined in the *ADMigraCMS* method: PHP metamodel, ASTM\_PHP metamodel and the CMS Common Metamodel. To validate them we have executed the automated transformations implemented in Chapter 6 and we have edited the corresponding models, according to those metamodels, with the graphical editors implemented for this purpose.
- The **proper performance of the graphical editors.** We are interested in validating the graphical editors implemented for the *ADMigraCMS* toolkit. We have validated them representing, editing and restructuring the models resulting from the different automated transformations.
- The **correct execution of the automated transformations.** We want to validate the correct execution of the different automated transformations defined within the *ADMigraCMS* method. We validate the T2M transformation by means of the execution of the model extractor implemented in Java. We prove whether the ASTM\_PHP model generated from the PHP code is correct according to the specification of this PHP code. Otherwise, we validate the M2M transformations by the execution of the transformation rules implemented in ATL. For instance, we validate the correctness of the Code model generated from the ASTM\_PHP model by executing the corresponding transformation rules. Finally, we validate the M2T transformation implemented in Acceleo by checking whether the PHP code generated is correct regarding the ASTM\_PHP model from which the code is generated.

To validate all these points, we have used the case studies explained in the following sections.

## 7.2 Case Study 1 and Case Study 2 Selection

The first case study is the **migration of Coaching Web**, a CMS-based Web application implemented in *Wordpress* and it is wanted to be migrated *to Drupal*. *Coaching Web* manages a coaching centre providing users with information about the different types of coaching (Whitworth, 2007) (family coaching, sport coaching or executive coaching) as well as it allows users to see explanatory videos about the process of coaching, and make an appointment with the centre. This case study has been used to validate all the points described in the previous section.

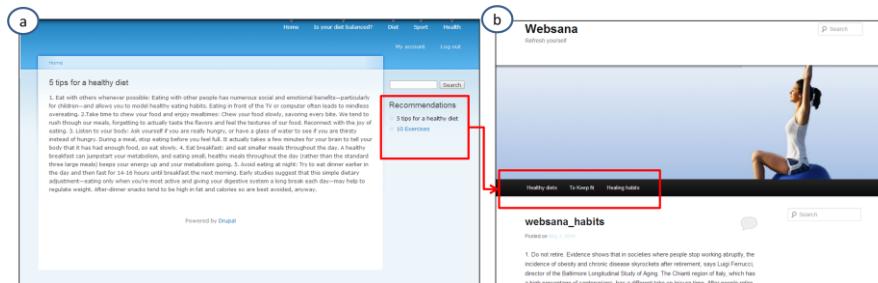
This case study is centred in the migration of *Blocks*, one of the elements belonging to the *Presentation concern* of the CMS Common Metamodel. We want to migrate the PHP code that implements a *block* that displays a clock from Wodpress to Drupal. Figure 7-2.a presents the implementation of *Coaching Web* in *Wordpress* and Figure 7-2.b in *Drupal*. The *block* migrated, which displays a clock, is marked in red.



**Figure 7-2. a) Implementation of Coaching Web in Wordpress, b) Implementation of Coaching Web in Drupal.**

The second case study is the **migration of Websana**, a CMS-based Web application implemented in *Drupal* and it is wanted to be migrated *to Wordpress*. *Websana* is the Web of a wellness and nutrition centre. It provides users with information about diets, exercices and recommendations about healthy habits. Moreover, it allows users to order nutrition packages and make reservations in their spa facilities. As occurred with the *Coaching Web*, *Websana* has been used to validate all the points described in the previous section.

This case study is focused on the migration of the elements menu, menu item, page and content of the *Navigation* and *Content concern* of the CMS Common Metamodel. The PHP code that implements a menu with its menu items, pages and contents in *Drupal* is migrated to *Wordpress*. Figure 7-3.a shows the implementation of *Websana* in *Drupal* and Figure 7-3.b in *Wordpress*. The menu which has been migrated is marked in red.



**Figure 7-3. a) Implementation of Websana on Drupal, b) Implementation of Websana on Wordpress.**

### 7.3 Data Retrieval

According to Yin (Yin, 2003), this task provides an overview of the case studies using multiple sources of information and defining a set of questions which must be resolved with the execution of the case studies. From the resolution of these questions we extract the final conclusions. The sources of information considered for our case studies are the files implemented in PHP which define the elements (menu, menu items, pages and blocks) being migrated.

#### Source of Information for the Migration of Coaching Web

The block migrated during the migration of *Coaching Web* displays an analogical clock at the right side of the user interface. In Figure 7-4, we present the PHP code that implements this block in *Wordpress*. This code, stored in a php file called *digital\_clock.php*, is migrated by using the *ADMigraCMS* toolkit.

```

1 add_action('widgets_init','func_clock_websana');
2 function func_clock_websana(){
3     register_widget('clock_websana');
4 }
5 class clock_websana extends WP_Widget{
6     function clock_websana(){
7         parent::WP_Widget(false,$name = 'clock');
8     }
9     function widget($args,$instance){
10        extract($args);
11        $message=<script src="http://24timezones.com/is/swfobject.js" language="javascript">
<script src="http://24timezones.com/timescript/maindata_js.php?city=1321702" lang-
<table><tr><td><div id="flash_container_tt51de9bb993bdb"></div><script type="text/
var flashMap = new SWFObject("http://24timezones.com/timescript/clock_final.swf");
flashMap.addParam("movie", "http://24timezones.com/timescript/clock_final.swf");
flashMap.addParam("quality", "high");
flashMap.addParam("wmode", "transparent");
flashMap.addParam("flashvars", "color=0099FF&logo=1&city=1321702");
flashMap.write("flash_container_tt51de9bb993bdb");
</script></td></tr><tr><td style="text-align: center; font-weight: bold"><a href=
12        echo($message);
13    }
14 }

```

Figure 7-4. PHP code implementing the *Block* in Wordpress.

The PHP code in Figure 7-4 is composed of a function call to the function *add\_action* (on line 1) which displays the clock on the UI, a function definition called *func\_clock\_websana* which registers the block in the database and finally, a class definition which extends from *WP\_Widget* that defines the block. Within the body of this class definition two function definitions are specified, *clock\_websana* the constructor and *widget* which defines the code implementing the clock.

Figure 7-5 shows the PHP code that implements this block in *Drupal*. This code is the result of the execution of the *ADMigraCMS* toolkit from the PHP code presented in Figure 7-4.

```

1 function clock_install() {
2     $nid = db_insert('block_custom')
3     ->fields(array(
4         'body' => ' <script src="http://24timezones.com/is/swfobject.js">
<script src="http://24timezones.com/timescript/maindata_js.php?city=1321702" lang-
<table><tr><td><div id="flash_container_tt51de9bb993bdb"></div><script type="text/
var flashMap = new SWFObject("http://24timezones.com/timescript/clock_final.swf");
flashMap.addParam("movie", "http://24timezones.com/timescript/clock_final.swf");
flashMap.addParam("quality", "high");
flashMap.addParam("wmode", "transparent");
flashMap.addParam("flashvars", "color=0099FF&logo=1&city=1321702");
flashMap.write("flash_container_tt51de9bb993bdb");
</script></td></tr><tr><td style="text-align: center; font-weight: bold"><a href=
15         'info' => 'clock',
16         'format' =>'full_html',
17     )->execute();
18 }

```

Figure 7-5. PHP code implementing the *Block* in Drupal.

### Source of Information for the Migration of Websana

The menu migrated during the migration of *Websana* is called *Recommendations* and it is composed of two menu items. One menu item is called *5 tips for a healthy diet* and links to a page about healthy diets, and the other one

is called *10 exercises* leading to a page about physical exercises to keep fit. The PHP code that implements these elements on *Drupal* is defined in a file called *customized\_menu\_websana.module* located in the directory *sites/all/modules* within the *Drupal* project. In Figure 7-6, we present the PHP code implementing the menu.

```

1 function customized_menu_websana_install() {
2     $menu = array( 'menu_name' => 'customized_menu_websana',
3                   'title' => 'Customized Menu',
4                   'description' => 'Further information
5                                     about diets and excercises';
6     menu_save($menu);
7 }
```

**Figure 7-6. PHP code implementing the *Menu* in Drupal.**

As we can see in Figure 7-6, the function which creates the *menu* is called *customized\_menu\_websana\_install* (on line 1). The features of this menu are defined with an array definition (on lines 2-5). Finally, the function call *menu\_save* creates and stores in the database the menu. Figure 7-7 shows the PHP code to implement the menu items which are located in the menu created.

```

1 function customized_menu_websana_menu() {
2     $items=array();
3     $items['diets'] = array(
4         'title' => '5 tips for a healthy diet',
5         'description'=> 'Recommendations of balanced diets',
6         'page callback' => 'websana_diets',
7         'access callback' => TRUE,
8         'weight' => 0,
9         'type' => MENU_NORMAL_ITEM,
10        'menu_name' =>'customized_menu_websana');
11
12    $items['exercises'] = array(
13        'title' => '10 Exercises',
14        'description'=> 'Recommendations of exercises to keep fit',
15        'page callback' => 'websana_exercises',
16        'access callback' => TRUE,
17        'weight' => 1,
18        'type' => MENU_NORMAL_ITEM,
19        'menu_name' =>'customized_menu_websana');
20
21    return $items;
22 }
```

**Figure 7-7. PHP code implementing the *MenuItem* in Drupal.**

As it is shown in Figure 7-7, the function *customized\_menu\_websana\_menu* creates these menu items. The features of each menu item are defined with array definitions. On lines 3-10, the menu item *5 tips for a healthy diet* is defined. Some of its features are specified, such as the title (on line 4), the description (on line 5) or the page linked by the menu item (on line 6).

In Figure 7-8, we present the PHP code that built the page linked to the menu item *5 tips for a healthy diet*.

```

1 function websana_diets () {
2 return "
1. Eat with others whenever possible: Eating with other p
emotional benefits—particularly for children—and allows y
Eating in front of the TV or computer often leads to mind
2. Take time to chew your food and enjoy mealtimes: Chew
We tend to rush through our meals, forgetting to actually
of our food. Reconnect with the joy of eating.
3. Listen to your body: Ask yourself if you are really hu
instead of hungry. During a meal, stop eating before you
brain to tell your body that it has had enough food, so e
4. Eat breakfast: and eat smaller meals throughout the da
jump start your metabolism, and eating small, healthy mea
(rather than the standard three large meals) keeps your e
5. Avoid eating at night: Try to eat dinner earlier in the
Early studies suggest that this simple dietary adjustment
long break each day—may help to regulate weight. After-d
3 )

```

**Figure 7-8. PHP code implementing the *Page* in Drupal.**

As we can see in Figure 7-8, on line 2 there is a return sentence returning the text content displayed in the page.

In the following, we present the PHP code implementing the menu with the menu items and the pages in Wordpress resulting at the end of the migration process executed by the *ADMigracMS* toolkit. This PHP code is defined in the file *customized\_menu\_websana.php* stored in the directory *plugins* in the Wordpress project. Figure 7-9 shows this PHP code. As we can see, the function called *create\_menu\_customized\_menu\_websana* defines the code to implement the menu, menu items and pages.

```

1 function create_menu_customized_menu_websana(){
2     $var_0=wp_create_nav_menu('customized_menu_websana');
3     $var_1=$var_0;
4     $var_2=array('post_title'=>'websana_diets',
5     'post_type'=>'post',
6     'post_status'=>'publish',
7     'post_content'=>'1. Eat with others whenever possible: Eating with o
emotional benefits—particularly for children—and allows you to model h
Eating in front of the TV or computer often leads to mindless overeating.
8
9     $var_3=array('post_title'=>'websana_exercises',
10    'post_type'=>'post',
11    'post_status'=>'publish',
12    'post_content'=>'1. When starting: Always remember that you should be
to set aside time for your exercise and set a goal you want to reach.
Stay within reason, though, as overdoing can actually damage your body.
13
14     $var_4=array('menu-item-title'=>'5 tips for a healthy diet',
15     'menu-item-description'=>'Recommendations of balanced diets',
16     'menu-item-position'=>0, 'menu-item-type'=>'post_type',
17     'menu-item-status'=>'publish',
18     'menu-item-object'=>'post',
19     'menu-item-object-id'=>$var_8_page);
20     $var_5=array('menu-item-title'=>'10 Exercises',
21     'menu-item-description'=>'Recommendations of exercises to keep fit',
22     'menu-item-position'=>1,
23     'menu-item-type'=>'post_type',
24     'menu-item-status'=>'publish',
25     'menu-item-object'=>'post',
26     'menu-item-object-id'=>$var_9_page);
27
28 }
29 register_activation_hook(__FILE__,'create_menu_customized_menu_websana');

```

**Figure 7-9. PHP code implementing the *Menu*, *MenuItem* and *Page* in Wordpress.**

As we can see in Figure 7-9, the function call `wp_create_nav_menu` (on line 2) creates the menu. The name is *Customized Menu* passed as a parameter.

The pages are defined with array definitions. On lines 3-6, the page about healthy diets is defined. Some of these features are title (on line 3), the type (on line 4), the status (on line 5) or the content of the page (on line 6). Once the features of the page are defined, the page is created and inserted in the database with the function call `wp_insert_post` (on line 11).

As it is shown in Figure 7-9, the menu items are also defined with an array definition. On lines 13- 18, the array storing the features of the menu item *5 tips for a healthy diet* is defined. The features specified for the menu item, are the description (on line 14) and the position (on line 15) among others. The function call `wp_update_nav_menu_item` creates the menu item and insert it into the database.

Finally, on line 29 we can see how the function `register_activation_hook` launches the function `create_menu_customized_menu_websana` which is passed as a parameter and execute the creation of all the elements explained.

### Questions to Assess the Validation

we present the following questions which must be resolved with the execution of the case studies and allow us to assess this validation and extract the final conclusion is Section 7.5. These questions are called Case Study Questions (CSQ).

- **CSQ1** – Is the model extractor able to recognize all the PHP sentences and expressions from the legacy PHP code?
- **CSQ2** – Is the ASTM\_PHP model able to represent at PSM level the syntax of the legacy PHP code? Is it possible to represent the ASTM\_PHP model with the tree-like graphical editor we have implemented?
- **CSQ3** – Is the transformation from the ASTM\_PHP model into the Code model possible? Is it possible to represent the Code model with the tree-like graphical editor we have implemented?
- **CSQ4** – Is the transformation from the Code model to the CMS model at possible? Is it possible to represent the generated CMS model with the UML-like graphical editor we have implemented?
- **CSQ5** – Is it possible to generate the target Code model from the restructured CMS Model?

- **CSQ6** – Is it possible to generate the target ASTM\_PHP model from the target Code model?
- **CSQ7** – Is the PHP code which implements the module interpreted correctly by the Wordpress platform?

## 7.4 Design and Execution of the Case Study

The design of a case study is focused on defining a global vision of the tasks involved. The execution of a case study consists on the application of these tasks on a concrete scenario to validate an approach and to retrieve data to reach to interesting conclusions. In this dissertation, we present one out of the two case studies used for the validation of our approach. Concretely, in this section, we present the case study focused on the migration of Websana. Figure 7-10 shows a global overview of the tasks involved in the execution of this case study using the *ADMigraCMS* toolkit.

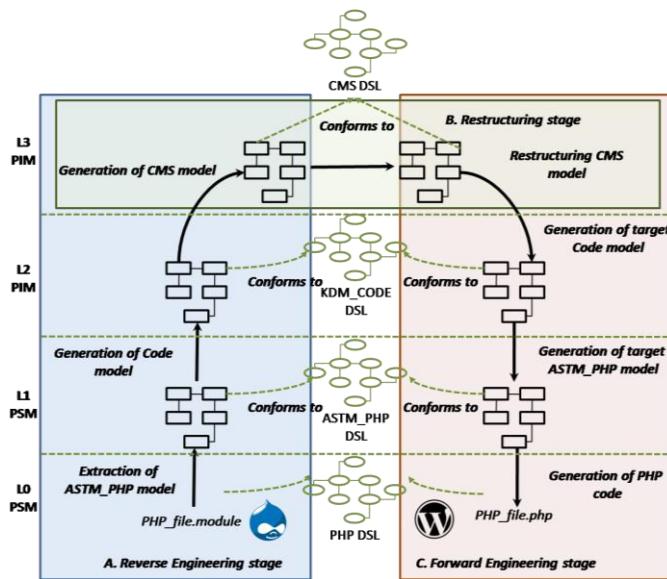


Figure 7-10. Overview of the tasks executed in the case studies.

- **Extraction of ASTM\_PHP model:** This task is defined to extract an ASTM\_PHP model from the PHP code that implements the module in Websana that implements the menu on *Drupal*. This task allows us to validate the correct specifications of the PHP metamodel and the ASTM\_PHP metamodel, the proper performance of the tree-like graphical

editor for ASTM\_PHP DSL as well as the correct execution of the L0-to-L1 T2M transformation implemented as a model extractor defined in Java.

- **Generation of Code model:** In this task the Code model is generated from the ASTM\_PHP model. This task allows us to validate the proper performance of the tree-like graphical editor that supports the KDM\_CODE DSL as well as the correct execution of the L1-to-L2 M2M transformation implemented in ATL.
- **Generation of CMS model:** The task is focused on the generation of the CMS model from the Code model. This task allows us to validate the correct specification of the CMS Common Metamodel, the proper performance of the UML-like graphical editor for CMS DSL as well as the correct execution of the L2-to-L3 M2M transformation implemented as a model extractor defined in Java.
- **Restructuring CMS model:** This task is based on restructuring the CMS model considering the features of Wordpress platform. Apart from restructuring the software artefacts extracted from the legacy CMS-based Web application, it is possible to insert new elements which will be implemented in the following steps. This task allows the validation of the correct specification of the CMS Common Metamodel as well as the proper performance of the UML-like graphical editor for CMS DSL.
- **Generation of target Code model:** This task generates the target Code model from the restructured CMS model. This Code model represents at PIM level the code which will be executed within the Wordpress platform. This task allows the validation of the correct execution of the L3-to-L2 M2M transformation implemented in ATL.
- **Generation of target ASTM\_PHP model:** In this task the target ASTM\_PHP model is generated from the target Code model. This ASTM\_PHP model represents the PHP code implementing the software artefacts in Wordpress at PSM level. This task allows the validation of the correct execution of the L2-to-L1 M2M transformation implemented in ATL.
- **Generation of PHP code:** The last task is the generation of the PHP code from the target ASTM\_PHP model. This code is the implementation in Wordpress of the migrated elements. This task allows the validation of the correct execution of the L1-to-L0 M2T transformation implemented in Acceleo.

Table 7-1 shows how these different tasks involved in the migration of Websana cover the points presented in Section 7.1 that we are interested in to validate.

**Table 7-1. Coverage of the interesting points to validate with the Websana case study.**

	Extraction of ASTM_PHP model	Generation of Code model	Generation of CMS model	Restructuring CMS model	Generation of target Code model	Generation of target ASTM_PHP model	Generation of PHP code
Correct specification PHP metamodel	✓						
Correct specification ASTM_PHP metamodel	✓						
Correct specification CMS Common Metamodel			✓	✓			
Proper performance of the tree-like graphical editor for ASTM_PHP DSL	✓						
Proper performance of the tree-like graphical editor for KDM_CODE DSL		✓					
Proper performance of the UML-like graphical editor for CMS DSL			✓	✓			
Correct execution of the L0-to-L1 T2M transf.	✓						
Correct execution of the L1-to-L2 M2M transf.		✓					
Correct execution of the L2-to-L3 M2M transf.			✓				
Correct execution of the L3-to-L2 M2M transf.					✓		
Correct execution of the L2-to-L1 M2M transf.						✓	
Correct execution of the L1-to-L0 M2T transf.							✓

In the following, we present the migration of Websana using the *ADMigracMS* toolkit. In Section 7.4.1, we explain the extraction of the ASTM\_PHP model, in Section 7.4.2 we explain the generation of the Code model from the ASTM\_PHP model, in Section 7.4.3 we present the generation of the

CMS model from the Code model, in Section 7.4.4 we show the restructuring of the CMS model, in Section 7.4.5 we explain the generation of target Code model from the restructured CMS model, in Section 7.4.6 we present the generation of the target ASTM\_PHP model from the target Code model and finally, in Section 7.4.7, we present the generation of PHP code from the target ASTM\_PHP model. This code implements the menu, its menu items and their pages an contents of Websana in Wordpress.

#### 7.4.1 Extraction of the ASTM\_PHP Model

From the PHP code implementing the menu, menu items and pages of Websana based on *Drupal*, we extract a ASTM\_PHP model. Figure 7-11 marks this task within the whole migration process defined for the case study of *Websana*.

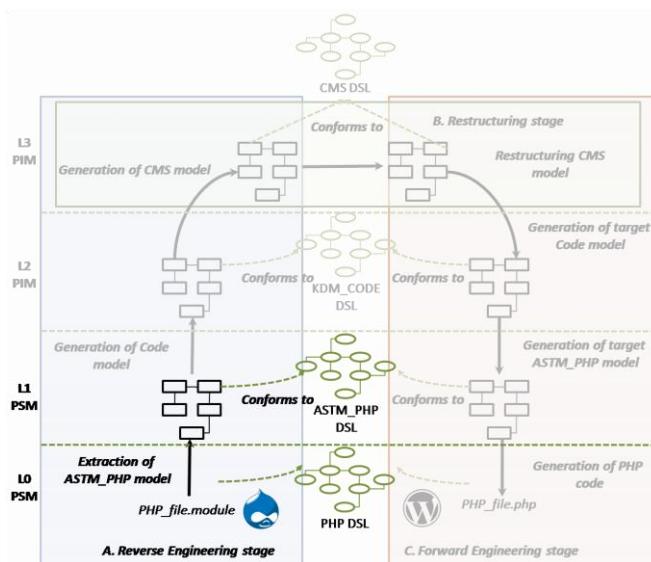


Figure 7-11. Extraction of the ASTM\_PHP model.

This task allows us to validate the points presented below:

- The correct specification of the PHP metamodel.
- The correct specification of the ASTM\_PHP metamodel.
- The proper performance of the tree-like graphical editor for ASTM\_PHP DSL.

- The correct execution of the L0-to-L1 T2M transformation implemented as a model extractor defined in Java.

In the following, we present the transformation of the PHP code that implements the menu in *Drupal* to the elements of the ASTM\_PHP model. Concretely, we present the T2M transformation between the PHP code that implements the menu items composing the menu (see Figure 7-7) to the ASTM\_PHP model. The transformation of the rest of the code is presented in Appendix F.

To support the explanation of this T2M transformation, we use the Table 7-2 that sums up this process. The first column of this table shows the fragment of PHP code being migrated, the second and third columns present the PHP element and its attributes that represents the fragment of PHP code, the fourth and fifth columns present the ASTM\_PHP element and its elements generated from the fragment of PHP code.

**Table 7-2. Transformations from the PHP code to the ASTM\_PHP model.**

PHP CODE	PHP	ATTRIBUTES	ASTM_PHP	ATTRIBUTES
Function customized_me nu_websana_m enu()	Function Def Statemen t	Visibility = public functionIdentifier = Name params = null body = VariableDefStatemen+ VariableDefStatemen+ VariableDefStatement + ReturnStatement	DeclarationOrD efinitionStateme nt	declOrDefn: FunctionDefinition  accessKind = public identifierName=Name formalParameters = null body = DeclarationOrDefinitio nStatement + ExpressionStatement + ExpressionStatement + ReturnStatement
\$items = array();	Variable DefState ment	Visibility = public Left = Name Right = arrayType	DeclarationOrD efinitionStateme nt	declOrDefn = VariableDefinition  accesskind = public identifierName = Name initialValue = CollectionExpression.
Array ()	Array Type	Params = null	Collection Expression	expressionList = null.
\$items['diets'] = array('title' =>'5 tips for a healthy diet'...)	Variable Def Statemen t	Visibility = public Left = ArrayAccess Right = ArrayType	Expression Statement	Expression = BinaryExpression  Operator = Assign leftOperand =ArrayAccessPHP rightOperand = CollectionExpression
\$items ['diets']	Array Access	arrayIdentifier = Variable sus = Suscript	Array AccessPHP	arrayName = IdentifierReference subscripts =

PHP CODE	PHP	ATTRIBUTES	ASTM_PHP	ATTRIBUTES
array('title' =>'5 tips for a healthy diet	Array Type	Params = arrayEntry	Collection Expression	StringLiteral expressionList = DuplaArray
'title' =>'5 tips for a healthy diet'	ArrayEntry	Key = StringLiteral Value = StringLiteral	DuplaArray	Index = StringLiteral Value = StringLiteral
return \$items	ReturnStatement	return = Variable	Return Statement	returnValue = IdentifierReference

- Transformation from the function definition *customized\_menu\_websana\_menu()* into the element *FunctionDefinition*: The function *customized\_menu\_websana\_menu()* represented with the element *FunctionDefStatement* in the PHP metamodel and its attributes are transformed correctly into the elements *DeclarationOrDefinitionStatement* along with the *FunctionDefinition* in the ASTM\_PHP model. The attribute *body* of the *FunctionDefStatement* is composed of three *VariableDefStatements* and one *ReturnStatement*. The *VariableDefStatement* representing the array storing the menu items is mapped accurately to a *DeclarationOrDefinitionStatement* and the two *VariableDefStatements* that represent the two menu items are mapped to *ExpressionStatements*. Finally, the *ReturnStatement* is mapped to a *ReturnSatement* element. These elemenents are represented properly in the ASTM\_PHP model as it is shown in Figure 7-14 using the tree-like graphical editor implemented for the ASTM\_PHP DSL.

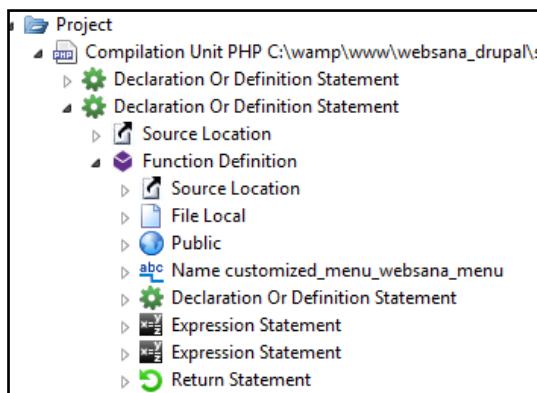


Figure 7-12. The *FunctionDefinition* element within the ASTM\_PHP model.

- **Transformation of the variable definition `$items = array()` into the element *VariableDefinition*:** the variable definition `$items = array()`, represented in the PHP metamodel as a *VariableDefStatement*, is transformed correctly into a *DeclarationOrDefinitionStatement* element and a *VariableDefinition* within the ASTM\_PHP model. The *ArrayType* from the attribute *right* is transformed properly into a *CollectionExpression* element. No array entries are defined for this array so that its attribute *expressionList* is defined as null. These elements are represented accurately in the ASTM\_PHP model as we can see in Figure 7-13.

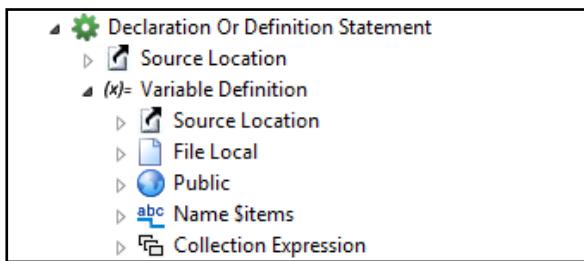
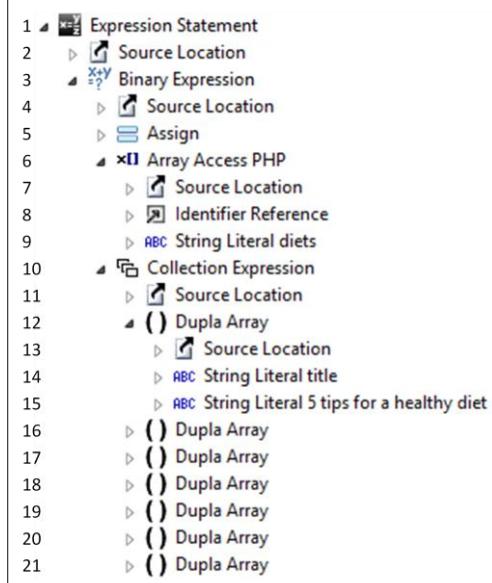
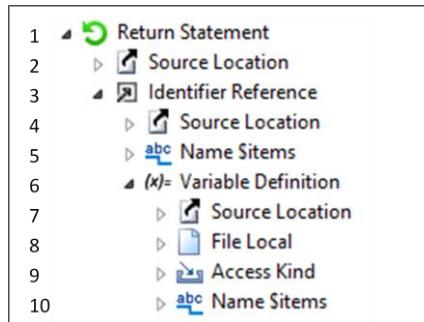


Figure 7-13. The *VariableDefinition* element within the ASTM\_PHP model.

- **Transformation of the variable definition `$items['diets'] = array('title' =>'Healthy diets' ...)` into the element *ExpressionStatement*:** The PHP statement `$items['diets'] = array('title' =>'Healthy diets' ...)`, represented with a *VariableDefStatement* in the PHP metamodel, is transformed correctly into an *ExpressionStatement* containing a *BinaryExpression* in the ASTM\_PHP model. It is because the attribute *left* of the *VariableDefStatement* is defined as an *ArrayAccess* (see Table 5-27). The *ArrayAccess* stored in the attribute *left* and the *ArrayType* from the attribute *right* of the *VariableDefStatement* are transformed properly into the *BinaryExpression* element, *ArrayAccessPHP* and *CollectionExpression*, respectively. The *Subscripts* defined for the *ArrayAccess* are transformed correctly into *StringLiterals* in the *ArrayAccessPHP*, whereas the array entries of the *ArrayType* are transformed in a correct way into *DuplaArrays* within the *CollectionExpression*. All these elements are accurately represented in the ASTM\_PHP model as we can see in Figure 7-14 with the use of the tree-like graphical editor defined for the ASTM\_PHP DSL.

Figure 7-14. The *ExpressionStatement* element within the ASTM\_PHP model.

- **Transformation of the *return \$items* into the element *ReturnStatement*:** The *return \$items*, represented in the PHP metamodel as the element *ReturnStatement*, is transformed properly into the element *ReturnStatement* of the ASTM\_PHP model. The *Variable* returned is correctly transformed into an *IdentifierReference*. Figure 7-15 presents this element in the ASTM\_PHP model.

Figure 7-15. The *ReturnStatement* element within the ASTM\_PHP model.

### 7.4.2 Generation of Code Model

From the previous ASTM\_PHP model, the Code model is generated. Figure 7-16 marks this task within the whole migration process defined for the case study of Websana.

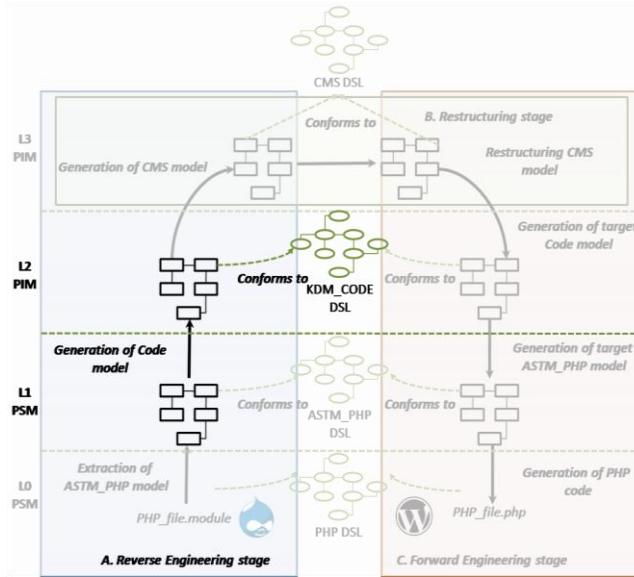


Figure 7-16. Generation of Code model.

This task allows us to validate the points presented below:

- The proper performance of the tree-like graphical editor for KDM\_CODE DSL.
- The correct execution of the L1-to-L2 M2M transformation implemented with ATL.

Table 7-3 shows the M2M transformation between the elements of the previous ASTM\_PHP model that represent the function *customized\_menu\_websana\_menu* into the elements in the Code model.

Table 7-3. Transformations from the ASTM\_PHP model to the Code model.

PHP CODE	ASTM_PHP	ATTRIBUTES	KDM	ATTRIBUTES
Function customized_ menu_websa	DeclarationOrD efinitionStateme nt	declOrDefn: FunctionDefinition	--	--

PHP CODE	ASTM_PHP	ATTRIBUTES	KDM	ATTRIBUTES
na_menu ()	Function Definition	accessKind = public identifierName=Name formalParameters = null body = DeclarationOrDefinitionStatement + ExpressionStatement + ExpressionStatement + ReturnStatement	Method Unit	export = public source = SourceRef name = customized_menu_web sana_install codeElement= 4 ActionElements + StorableUnit
				parameterUnit = null
\$items = array();	DeclarationOrDefinitionStatement	declOrDefn = VariableDefinition	Action Element	kind = newArray name= newArray actionRelation = Creates, Writes
		Accesskind = public identifierName = Name initialValue = collectionExpression		
Array ()		expressionList = null.		
\$items['diets'] = array('title' =>'5 tips for a healthy diet' ...)	Expression Statement	Expression = BinaryExpression	Action Element	kind = arrayReplace name= arrayReplace codeElement = ActionElement actionRelation = Addresses, Reads, Writes
		Operator = Assign leftOperand = ArrayAccessPHP rightOperand = CollectionExpression		
\$items['diets']		arrayName = IdentifierReference subscripts=StringLiteral		
Array ('title' =>'5 tips for a healthy diet'...)	Collection Expression	expressionList = DuplaArray	Action Element	kind = compound name= compound codeElement = ActionElement + ActionElement
			Action Element	kind = newArray name= newArray actionRelation = Creates, Writes Flow
'title' =>'5 tips for a healthy diet'	DuplaArray	Index = StringLiteral Value = StringLiteral	Action Element	kind = arrayReplace name = arrayReplace codeElement = Value + Value actionRelation = Addresses, Reads, Reads, Writes and Flow
return \$items	Return Statement	returnValue = IdentifierReference	Action Element	kind = return name = return actionRelation = Reads

- **Transformation of a *FunctionDefinition* into a *MethodUnit*:** The *FunctionDefinition* of the ASTM\_PHP model that represents the *function customized\_menu\_websana\_menu ()* is transformed correctly to a

*MethodUnit* and a *Signature* into the Code model. In this case *Signature* is empty because any parameter has not been defined to be passed to this function definition. The four statements defined in the attribute *body* of the element *FunctionDefinition* have been transformed in a correct manner to four *ActionElements* within the Code model. These elements have been represented without errors in the Code model using the tree-like graphical editor implemented as we can see in Figure 7-17.

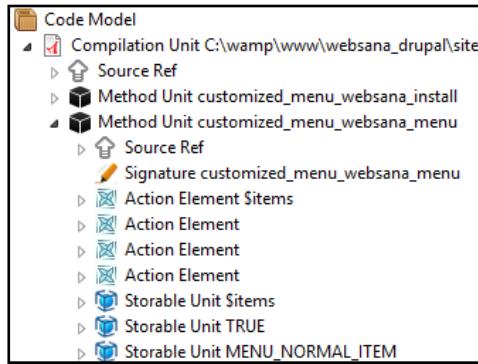


Figure 7-17. The *MethodUnit* element within the Code model.

- **Transformation of a *VariableDefinition* into an *ActionElement* defined as *NewArray*:** The *VariableDefinition* that represents `$items=array()` is transformed rightly to an *ActionElement* in the Code model. The attribute *initialValue* of the element *VariableDefinition* stores a *CollectionExpression* so that the attribute *kind* of the *ActionElement* is defined as a *newArray*. Within the attribute *actionRelation* of the *ActionElement*, the elements *Creates* and *Writes* have been created as it is indicated in the Micro KDM package. The *Creates* points to an *arrayType* and the *Writes* points to the variable *\$items*. Figure 7-18 shows correctly the *ActionElement* generated with this transformation in the Code model.

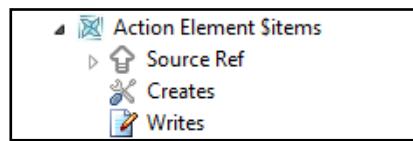


Figure 7-18. The *ActionElement* element (*NewArray*) within the Code model.

- **Transformation of a *ExpressionStatement* containing a *BinaryExpression* into an *ActionElement* defined as *NewArray*:** The *ExpressionStatement* whose attribute *expression* stores a *BinaryExpression* in the ASTM\_PHP

model is transformed without errors into an *ActionElement* in the Code model whose attribute *kind* is defined as an *arrayReplace* since the attribute *leftOperand* of the *BinaryExpression* is defined as an *ArrayAccessPHP*. Regarding to its attribute *actionRelation*, the elements *Addresses*, *Reads* and *Writes* have been created as it is specified in the Micro KDM package. The *CollectionExpression* from the *BinaryExpression* is transformed correctly into two *ActionElements*, one of them defined as a *compound* which contains another one defined as a *newArray*. The *DuplaArrays* from the *CollectionExpression* are transformed into *ActionElements* whose attribute *kind* is specified as *arrayReplace*. Figure 7-19 shows properly these elements in the Code model with the tree-like graphical editor.

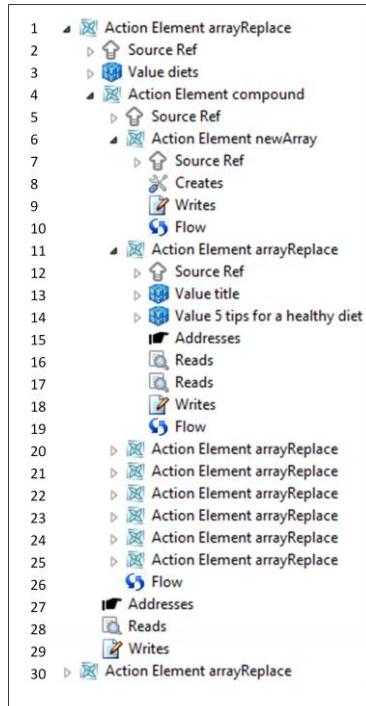


Figure 7-19. The *ActionElement* element (*NewArray*) within the Code model.

- **Transformation of the *ReturnStatement* element into an *ActionElement* defined as *Return*:** The *ReturnStatement* from the ASTM\_PHP model is transformed accurately into an *ActionElement* whose attribute *kind* is defined as *return*. Its attribute *actionRelation* stores a *Reads* which is related to the value returned. Figure 7-20 shows this element in the Code model.

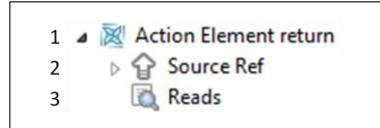


Figure 7-20. The *ActionElement* element (*Return*) within the Code model.

### 7.4.3 Generation of CMS Model

In this section, we explain the transformations between the previous Code model and the CMS model. Figure 7-21 shows this task in the migration process.

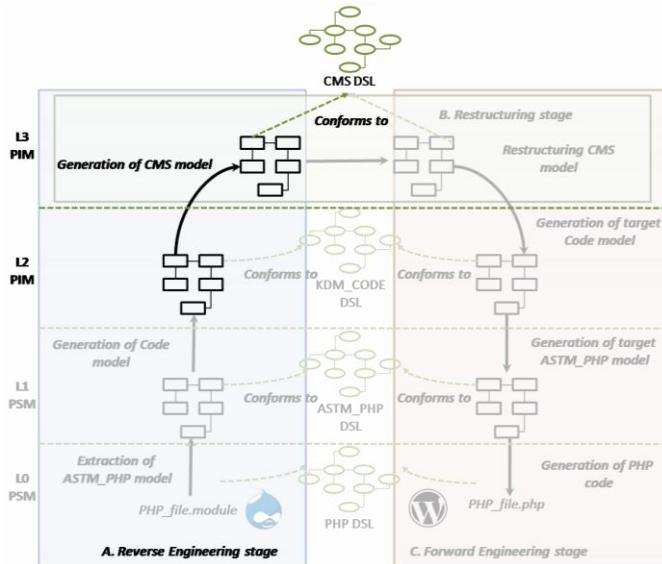


Figure 7-21. Generation of the CMS model.

This task allows us to validate the points presented below:

- The correct specification of the CMS Common Metamodel.
- The proper performance of the UML-like graphical editor for CMS DSL.
- The correct execution of the L2-to-L3 M2M transformation implemented with ATL.

In the following, we present the transformation required to generate automatically this CMS model from the previous Code model. Table 7-4 shows the transformations from the previous Code model into the elements *MenuItem* and *Page* of the CMS model.

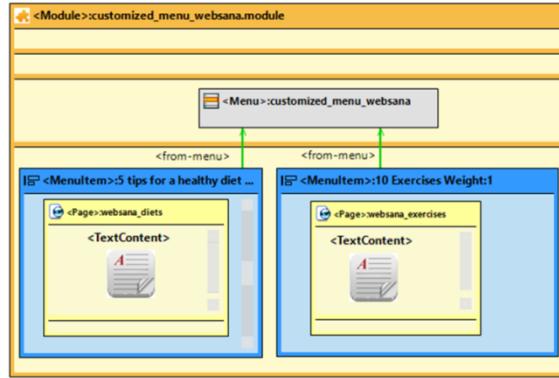
**Table 7-4.** Transformations from the Code model to the CMS model.

<i>PHP CODE</i>	<i>KDM</i>	<i>ATTRIBUTES</i>	<i>CMS</i>	<i>ATTRIBUTES</i>
Function customized_menu_websan a_menu ()	Mehod Unit	export = public source = SourceRef name = customized_menu_websan a_install codeElement= ActionElement (kind=newArray) + 2 ActionElements (kind=arrayReplace) + Actionelement (kind = return) + StorableUnit	--	--
	Signature	parameterUnit = null kind = newArray name= newArray actionRelation = Creates, Writes	--	--
\$items = array();	Action Element		--	--
\$items['diets'] = array('title' =>'5 tips for a healthy diet', 'description'=> 'Recommendations of balanced diets', 'weight' => 0, 'menu_name' =>'customized_menu_web sana)	Action Element	kind = arrayReplace name = arrayReplace codeElement = ActionElement (kind = compound) actionRelation = Addresses, Reads, Writes	Menu Item	
Array ('title' =>'5 tips for a healthy diet', 'description'=> 'Recommendations of balanced diets', 'weight' => 0, 'menu_name' =>'customized_menu_web sana')	Action Element	kind = compound name = compound codeElement = ActionElement (kind = newArray) + 4 ActionElement (kind = arrayReplace).	--	--
	Action Element	kind = newArray name= newArray actionRelation = Creates, Writes Flow	--	--
'title' =>'5 tips for a healthy diet'	Action Element	kind = arrayReplace name = arrayReplace codeElement = Value + Value actionRelation = Adresses, Reads, Reads, Writes and Flow	--	Title = 5 tip for a healthy diet (MenuItem)
'description'=> 'Recommendations of balanced diets'	Action Element	kind = arrayReplace name = arrayReplace codeElement = Value + Value actionRelation = Adresses, Reads, Reads, Writes and Flow		Description=R ecommendatio ns of balanced diets

PHP CODE	KDM	ATTRIBUTES	CMS	ATTRIBUTES
'weight' => 0	Action Element	kind = arrayReplace name = arrayReplace codeElement = Value + Value actionRelation = Adresses, Reads, Reads, Writes,Flow		Position = 0
'menu_name' =>'customized_menu_web sana'	Action Element	kind = arrayReplace name = arrayReplace codeElement = Value + Value actionRelation = Adresses, Reads, Reads, Writes,Flow		menu_name = customized_m enu_websana
'page_callback' =>'websana_diets'	Action Element	kind = arrayReplace name = arrayReplace codeElement = Value + Value actionRelation = Adresses, Reads, Reads, Writes,Flow	Page	title = websana_diets
			Text Context	
return \$items	Action Element	kind = return name = return actionRelation = Reads	--	--

- **Transformation of the *ActionElement* element into a *MenuItem* element:** The *MenuItems* in the CMS model are created without errors from *ActionElements* of the Code model defined as *arrayReplaces*. The main condition of these *ActionElements* is that it must be defined in a *MethodUnit* whose name ends with *\_menu*. The attributes of the *MenuItem* element are also transformed from other elements of the Code model. For instance, as we can see in Table 7-4, the attribute *title* is transformed from an *ActionElement* whose attribute *kind* is defined as an *arrayReplace* and represents the PHP code '*menu\_name' =>'customized\_menu\_websana*'.
- **Transformation of the *ActionElement* element into *Page* and *TextContent* elements:** The *Page* and *TextContent* elements of the CMS model is transformed rightly from an *ActionElement* defined as an *arrayReplace*. The PHP code that this *ActionElement* represents is '*page\_callback' =>'websana\_diets*'.

Figure 7-22 shows the CMS model generated. We can see the elements *MenuItem*, *Page* and *TextContent* along with the other element *Menu* extracted from the PHP code represented accurately with the UML-like graphical editor implemented for the CMS DSL.

Figure 7-22. *MenuItem*, *Page* and *TextContent* elements within the CMS model

#### 7.4.4 CMS Model Restructuring

In this section, we present the restructuring task where we redefine the CMS model generated in the previous task. Figure 7-23 frames this task in the whole migration process.

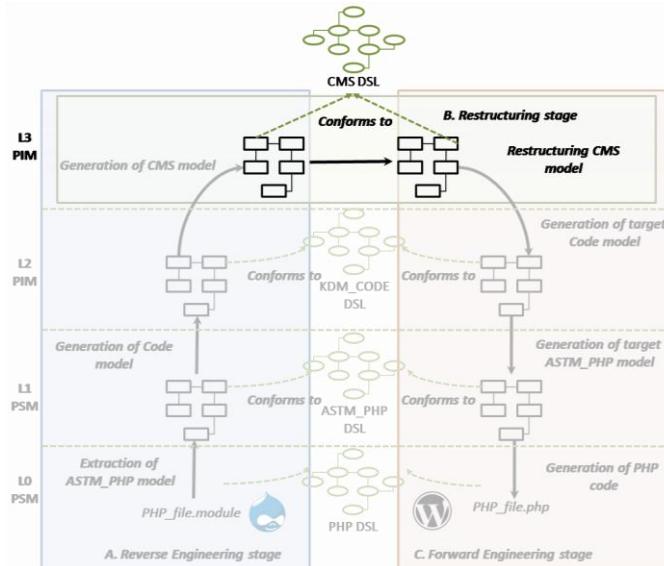


Figure 7-23. Restructuring CMS model.

This task allows us to validate the points presented below:

- The correct specification of the CMS Common Metamodel.

- The proper performance of the UML-like graphical editor for CMS DSL.

We have applied two changes to restructure the CMS model obtained in the previous task. These changes are presented below:

- **Redefinition of the names of the menu items** because we considered them too long and unintuitive. The menu item with the name *5 tips for a healthy diet* has been changed for the shorter name, *Healthy diet*. The menu item called *10 Exercises* has been modified for a more descriptive name like *To keep fit*.
- **Definition of a new menu item** called *Healing habits*. This menu item is linked to a page called *websana\_habits* which contains the text to be displayed in the target CMS-based Web application.

Figure 7-24 shows the restructured CMS model. As we can see in the figure, the new MenuItem is defined with the weight 2 (the third position with respect to the other menu items). Moreover, we can see how the three menu items belong to the menu *customized\_menu\_websana*.

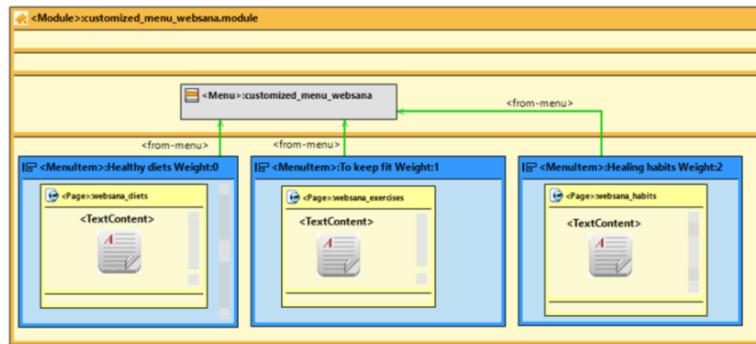


Figure 7-24. CMS model restructured.

#### 7.4.5 Generation of Target Code Model

We explain the task where we obtain the target Code model from the restructured CMS model from the previous section. This Code model represents at PIM level the code implementing the MenuItem, Pages and TextContents elements for the target CMS-based Web applications based on Wordpress. Figure 7-25 frames this task in the whole migration process.

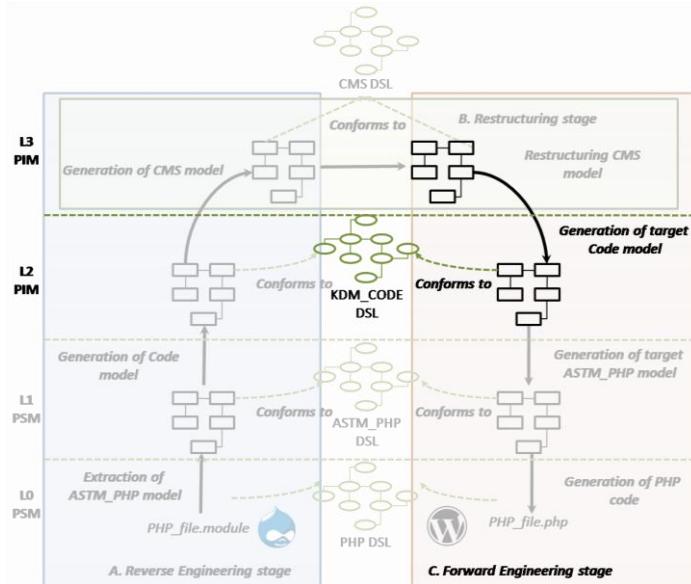


Figure 7-25. Generation of target Code model.

This task allows us to validate the : The correct execution of the L3-to-L2 M2M transformation implemented with ATL.

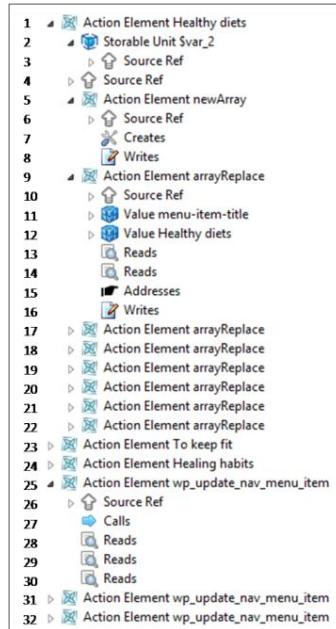
In Table 7-5, we present the transformations between the *MenuItem* element of the CMS model and the Code model. As it is presented in Figure 7-9, a *MenuItem* in Wordpress is created by the definition of an array that specifies the settings of the *MenuItem* and the call to the function *wp\_update\_nav\_menu\_item* which creates the *MenuItem*.

Table 7-5. Transformations from the restructured CMS model to the target Code model.

PHP CODE	CMS	ATTRIBUTES	KDM	ATTRIBUTES
\$var_2=array('me nu-item- title'=>'Healthy diets.);	MenuItem	title = Healthy diets description = Recommendations of balanced diets position = 0 menu_save = customized_menu_websan a	Action Element	kind = compound name = Healthy diets codeElement = ActionElement (kind = newArray) + 7 ActionElement (kind = arrayReplace).
array('menu-item- title'=>'Healthy diets'			Action Element	kind = newArray name= newArray actionRelation=Creates, Writes Flow
'menu-item- title'=>'Healthy diets'			Action Element	kind = arrayReplace name = arrayReplace codeElement = Value + Value actionRelation = Adresses, Reads, Writes and Flow

PHP CODE	CMS	ATTRIBUTES	KDM	ATTRIBUTES
wp_update_nav_menu_item(\$var_0,\$var_2);			Action Element	kind = call name= wp_update_nav_menu_item actionRelation = Calls, Reads, Reads

- **Transformation of the *MenuItem* element into elements of the Code model:** The *MenuItem* is transformed without errors to a set of elements within the Code model representing the PHP code in Wordpress. The definition of the array which specifies the settings is defined as an *ActionElement* determined as a *compound* since the different settings are specified with different positions in the array. This *ActionElement* called *Healthy diets* contains in its attribute *codeElement* one *ActionElement* defined as a *newArray* and seven *ActionElements* defined as *arrayReplace* which represent the different array entries determining the values of the settings. Finally, the function call to *wp\_update\_nav\_menu\_item* is represented as an *ActionElement* defined as a *call*. The action relationships defined within this *ActionElement* are, one *Calls* to the *MethodUnit* *wp\_update\_nav\_menu\_item*. Figure 7-26 shows the excerpt of the Code model resulting from these transformations using the tree-like graphical editor implemented for the KDM\_CODE DSL.

Figure 7-26. Target Code model fragment extracted from the *MenuItem* element.

In Table 7-6, we can see the transformations between the *Page* and *TextContent* elements of the restructured CMS model and the Code model. As we can see in Figure 7-9, a *Page* in Wordpress is created by the definition of an array that specifies the settings of the *Page* and the call to the function *wp\_insert\_post* which creates the page. The *TextContent* is represented as part of the settings of the page. It is stored in a position of the array whose index is defined as *post\_content*.

**Table 7-6. Transformations from the structured CMS to the target Code model.**

PHP CODE	CMS	ATTRIBUTES	KDM	ATTRIBUTES
\$var_9=array('post_title'=>'websana_diets', 'post_type'=>'post', 'post_status'=>'publish', 'post_content'=>'1. Eat with others' ...)	Page	title = websana_diets	Action Element	kind = compound name = websana_diets codeElement = ActionElement (kind = newArray) + 4 ActionElement (kind = arrayReplace).
array('post_title'=>'websana_diets')			Action Element	kind = newArray name = newArray actionRelation = Creates, Writes, Flow
'post_title'=>'websana_diets'			Action Element	kind = arrayReplace name = arrayReplace codeElement = Value + Value actionRelation = Addresses, Reads, Writes and Flow
\$var_9_page=wp_insert_post (\$var_9);			Action Element	kind = call name = wp_insert_post actionRelation = Calls, Reads
array('post_content'=>'1. Eat with others whenever possible...')	Text Content	Text = 1. Eat with others whenever possible	Action Element	kind = arrayReplace name = arrayReplace codeElement = Value + Value actionRelation = Addresses, Reads, Writes and Flow

- Transformation of the *Page* element into elements of the Code model:** As occurred with the *MenuItem* element, the *Page* is transformed correctly into a set of elements within the Code model representing the PHP code on Wordpress. The definition of the array defining the settings for the *Page* is represented with an *ActionElement* specified as a *compound*. It contains an *ActionElement* determined as a *newArray* and a set of *ActionElements* defined as *arrayReplaces* representing the different positions in this array defining the settings. The function call to *wp\_insert\_post* is represented with an *ActionElement* whose attribute *kind* is defined as a *call*.
- Transformation of the *TextContent* element into elements of the Code model:** Finally, the *TextContent* is transformed properly into an *ActionElement* whose attribute *kind* is defined as an *arrayReplace* since this

content is stored in a position of the array as another setting. Figure 7-27 shows the excerpt of the Code model resulting from these two transformations explained.

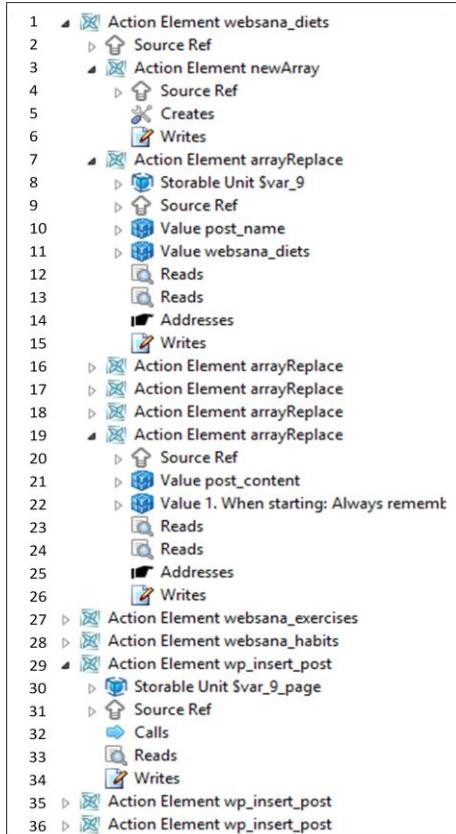


Figure 7-27. Target Code model fragment extracted from the *Page* and *TextContent* elements.

#### 7.4.6 Generation of Target ASTM\_PHP Model

In this section, we explain the task where we obtain the target ASTM\_PHP model from the Code model obtained in the previous section. This target ASTM\_PHP model represents at PSM level the PHP code implementing the *MenuItem*, *Pages* and *TextContents* based on Wordpress. Figure 7-27 frames this task in the whole migration process.

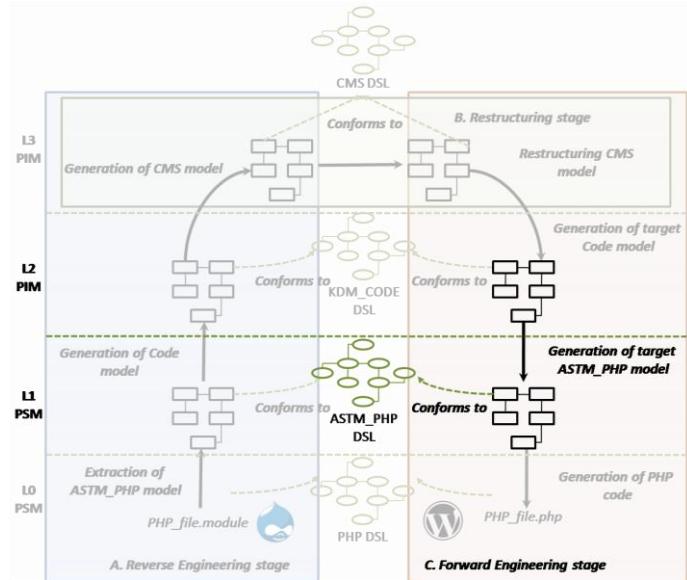


Figure 7-28. Generation of target ASTM\_PHP model.

This task allows us to validate the : The correct execution of the L2-to-L1 M2M transformation implemented with ATL.

In Table 7-7, we present the transformations from the elements of the target Code model to the elements of the target ASTM\_PHP model.

Table 7-7. Transformations from the target Code model to the target ASTM\_PHP model.

PHP CODE	KDM	ATTRIBUTES	ASTM_PHP	ATTRIBUTES
\$var_2=array('menu-item-title'=>'Healthy diets',);	Action Element	kind = newArray name = newArray actionRelation = Creates, Writes	DeclarationOrDefinition definitionStatement	declOrDefn = VariableDefinition  Accesskind = public identifierName = Name initialValue = collectionExpression
array('menu-item-title'=>'Healthy diets'	Action Element	kind = compound name = Healthy diets codeElement = ActionElement (kind = newArray) + 7 ActionElement (kind =	Collection Expression	expressionList = DuplaArray

<i>PHP CODE</i>	<i>KDM</i>	<i>ATTRIBUTES</i>	<i>ASTM_PHP</i>	<i>ATTRIBUTES</i>
		arrayReplace).		
'menu-item-title'=>'Healthy diets'	Action Element	kind = newArray name= newArray actionRelation = Creates, Writes Flow	DuplaArray	Index = StringLiteral Value = StringLiteral
wp_update_nav_menu_item (\$var_0,0,\$var_2)	Action Element	kind = call name= menu_save actionRelation = Calls, Reads	Expression Statement Function Call Expression	expression = FunctionCallExpression calledFunction = identifierReference actualParams = 3 ByValueParameterExpression

- Transformation of the *ActionElement* that represents `$var_2=array('menu-item-title'=>'Healthy diets'...)` into the **VariableDefinition element in the target ASTM\_PHP model:** The variable which defines the settings of the *MenuItems* is represented as a *DeclarationOrDefinitionStatement* whose attribute *declOrDefn* stores a *VariableDefinition*. This variable is called `$var_2` and its initial value is defined with a *CollectionExpression*. The entries of this *CollectionExpression* are defined with *DuplaArrays*. These elements are represented accurately in the target ASTM\_PHP model presented in Figure 7-29.
- Transformation of the *ActionElements* that represents `wp_update_nav_menu_item ($var_0,0,$var_2)` into the **FunctionCallExpression element int the target ASTM\_PHP model:** the function call to the function `wp_update_nav_menu_item` is represented as a *ExpressionStatement* whose attribute *expression* is defined as a *FunctionCallExpression*. The three parameters are transformed into *ByValueActualParameterExpression*. Figure 7-29 represents without errors

the excerpt of the target ASTM\_PHP model resulting from these transformations.

```

1   Declaration Or Definition Statement
2   ▷ Source Location
3   ▷ (x) Variable Definition
4   ▷ Source Location
5   ▷ File Local
6   ▷ Public
7   ▷ Name Svar_2
8   ▷ Collection Expression
9   ▷ Source Location
10  ▷ ( ) Dupla Array
11   ▷ Source Location
12   ▷ ABC String Literal menu-item-title
13   ▷ ABC String Literal Healthy diets
14   ▷ ( ) Dupla Array
15   ▷ ( ) Dupla Array
16   ▷ ( ) Dupla Array
17   ▷ ( ) Dupla Array
18   ▷ ( ) Dupla Array
19   ▷ ( ) Dupla Array
20 ▷ Declaration Or Definition Statement
21 ▷ Declaration Or Definition Statement
22 ▷ Expression Statement
23 ▷ Source Location
24 ▷ Function Call Expression
25 ▷ Source Location
26 ▷ Identifier Reference
27 ▷ Source Location
28 ▷ abc Name wp_update_nav_menu_item
29 ▷ Function Definition
30 ▷ ( ) By Value Actual Parameter Expression
31 ▷ Source Location
32 ▷ Identifier Reference
33 ▷ Source Location
34 ▷ abc Name Svar_0
35 ▷ (x) Variable Definition
36 ▷ ( ) By Value Actual Parameter Expression
37 ▷ Source Location
38 ▷ 12 Integer Literal 0
39 ▷ ( ) By Value Actual Parameter Expression
40 ▷ Expression Statement
41 ▷ Expression Statement

```

**Figure 7-29.** Excerpt of the target ASTM\_PHP model generated.

In Table 7-8, we present the mappings from the Code elements obtained from the *Page* and *TextContent* elements.

**Table 7-8. Transformation from the target Code model to target ASTM\_PHP model.**

PHP CODE	KDM	ATTRIBUTES	ASTM_PHP	ATTRIBUTES
\$var_9=array('post_t itle'=>'websana_diet s', 'post_type'=>'post',	Action Element	kind = newArray name= new Array actionRelation =	DeclarationOrD efinitionStatem ent	declOrDefn = VariableDefinitio n

<b>PHP CODE</b>	<b>KDM</b>	<b>ATTRIBUTES</b>	<b>ASTM_PHP</b>	<b>ATTRIBUTES</b>
'post_status'=>'publi sh','post_content'=> 1. Eat with others' ...		Creates, Writes	Variable Definition	Accesskind = public identifierName = Name initialValue = collectionExpress ion
array('post_title'=>' websana_diets'	Action Element	kind = compound name = websana_diets codeElement = ActionElement (kind = newArray) + 4 ActionElement (kind = arrayReplace).	Collection Expression	expressionList = DuplaArray
	Action Element	kind = newArray name= newArray actionRelation = Creates, Writes Flow		
'post_title'=>'websan a_diets'	Action Element	kind = arrayReplace name = arrayReplace codeElement = Value + Value actionRelation = Adresses, Reads, Reads, Writes and Flow	DuplaArray	Index = StringLiteral Value = StringLiteral
\$var_9_page=wp_in sert_post (\$var_9);	Action Element	kind = call name= menu_save actionRelation = Calls, Reads	DeclarationOr Definition Statement	declOrDefn = VariableDefinitio n
Variable Definition	Accesskind = public identifierName = Name initialValue = functionCallExpr ession			
FunctionCall Expression	calledFunction = identifierReferen ce actualParams = ByValueParamet erExpression			

- Transformation of the **ActionElement** that represents `$var_9 = array ('post_title' => 'websana_diets'...)` into the **VariableDefinition** element in the target **ASTM\_PHP** model: As occurred with the definition of the variable which stores the settings of the menu items, the variable storing the settings of the pages is mapped to **DeclarationOrDefinitionStatement** whose

attribute *declOrDefn* stores a *VariableDefinition*. This variable is called *\$var\_9* and its initial value is defined with a *CollectionExpression*. The entries of this *CollectionExpression* are defined with *DuplaArrays*.

- **Transformation of the ActionElement *\$var\_9\_page=wp\_insert\_post(\$var\_9)* into the *VariableDefinition* element in the target ASTM\_PHP model:** The function call to the function *wp\_insert\_post* is defined as a *VariableDefinition* since the value returned by this function call is stored in a variable called *\$var\_9\_page*. Therefore, it is mapped to *DeclarationOrDefinitionStatement* whose attribute *declOrDefn* stores a *VariableDefinition*. The *initialvalue* of this *VariableDefinition* is a *FunctionCallExpression*. Its parameter is mapped to *ByValueActualParameterExpression*. Figure 7-30 shows the excerpt of the ASTM\_PHP model resulting from these mappings.

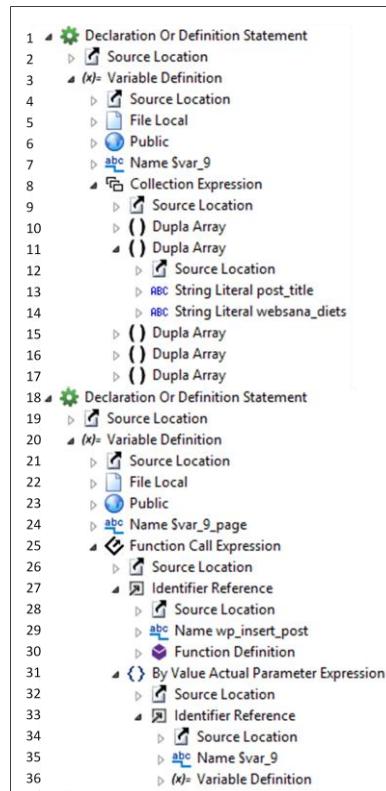


Figure 7-30. Excerpt of the target ASTM\_PHP model generated.

### 7.4.7 Generation of PHP Code

In this section, we present the validation of the L1-to-L0 M2T transformation required to generate the PHP code from the target ASTM\_PHP model generated in the previous section. This resulting PHP code runs correctly on the Wordpress platform. Figure 7-29 frames this task in the whole migration process.

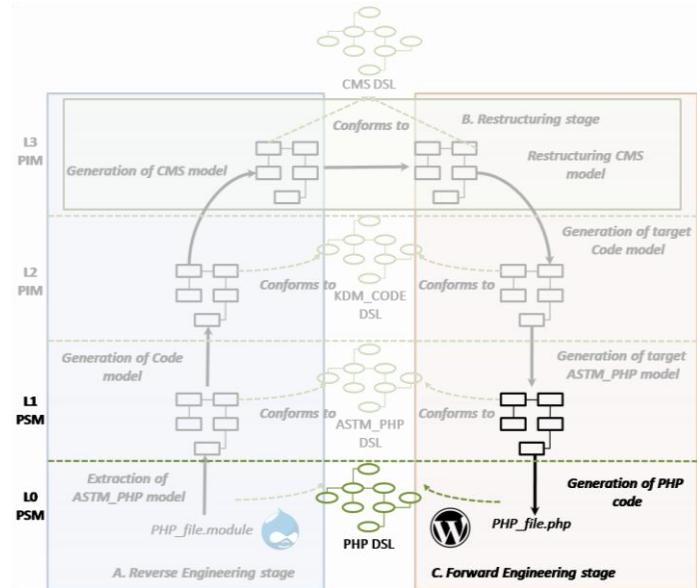


Figure 7-31. Generation of PHP code.

Table 7-9 shows the transformation rules in Acceleo which generate part of the PHP code to implement the *MenuItems*. This PHP code generated has been presented in Figure 7-9

Table 7-9. Generation of PHP code from the target ASTM\_PHP model.

ASTM_PHP	ATTRIBUTES	M2T TRANSFORMATION RULE	PHP CODE
DeclarationOrDefinitionStatement	declOrDefn = VariableDefinition	[template private generateVariableDefinition(aDeclaration : VariableDefinition)] [aDeclaration.identifierName.nameString] = [generateExpression(aDeclaration.initialValue)]/[template]	\$var_2=arry('menu-item-title'=>'Healty diets');
VariableDefinition	Accesskind = public identifierName = Name initialValue = collectionExpression		
Name	nameString = \$var_2		

ASTM PHP	ATTRIBUTES	M2T TRANSFORMATION RULE	PHP CODE
Collection Expression	expressionList = DuplaArray	[template private generateArray(aExpression : CollectionExpression)] <b>Array</b> ( [for(exp:Expression/aExpression.expr essionList) separator(',')] [generateExpression(exp)]/ [/for])/[template]	array('menu- item- title'=>'Healt hy diets'
DuplaArray	index = StringLiteral value = StringLiteral	[template private generateDuplaArray(aExpression : DuplaArray)] [generateExpression(aExpression.in dex)/] => [generateExpression(aExpression.v alue)]/[template]	'menu-item- title'=>'Healt hy diets'
FunctionCallEx pression	calledFunction = identifierReference actualParams = 3 ByValueParameterExp ression	[template private generateFunctionCallExp(functionCal l : FunctionCallExpression)] [functionCall.calledFunction.oclAs Type(IdentifierReference).name.na meString] ( [for (param: ActualParameter /functionCall.actualParams) separator(',')] [generateActualParam(param.oclAs Type(ByValueActualParameterExp ression))]/ [/for])/[template]	wp_update_ nav_menu_it em (\$var_0, \$v ar_2)

- **Transformation rule generateVariableDefinition:** This transformation rule prints the PHP code for the *VariableDefinition*. It prints the name of the variable as the left hand operand, the assign operator = and the expression as the right hand operand. To print the expression the rule *generateExpression* is launched.
- **Transformation rule generateFunctionCallExp:** The *initialValue* of the previous *VariableDefinition* is specified as a *FunctionCallExpression*. Thus, a transformation rule called *generateFunctionCallExp* is launched to process it. The template defining this transformation rule takes the value of the function stored in the attribute *calledFunction* of the *FunctionCallExpression*. If this name is defined as an *identifierReference* the transformation rule *generateIdRef* is launched. Then, the parameters passed to the function call are printed. These parameters are stored in the attribute *actualParams* of the *FunctionCallExpression*. These parameters are parsed by a *loop* (for). Each parameter is separated by commas and is processed by

the transformation rule *generateActualParam*. The parameters are bounded by two parentheses.

- **Transformation rule generateArray:** The rule *generateArray* prints the token **array** to start the definition of the new array. An array can be composed of different array entries. These array entries are bounded by two parentheses. To process them a loop is defined and the transformation rule *generationExpression* is launched.
- **Transformation rule generateDuplaArray:** The array entries are represented in the ASTM\_PHP model as *DuplaArrays*. These *DuplaArrays* are processed by the transformation rule *generateDuplaArray*. This rule takes the values from the attributes *index* and *value* and separate them by the operator  $=>$ .

## 7.5 Report and Conclusions

In this last section, we present the conclusions derived from the validation process carried out in this Chapter. For this, we answer the questions posed at the definition of the validation protocol (Section 7.1).

- **CSQ1 – Is the model extractor able to recognize all the PHP sentences and expressions from the legacy PHP code?**

Yes, the model extractor has parsed correctly the PHP code of the legacy CMS-based Web application. The parser has identified without errors the elements that specify the syntax of this legacy PHP code, including the precedence of operators in PHP. Therefore, we have validated the correct specification of the PHP metamodel.

This model extractor has extracted properly an ASTM\_PHP model. Thus, we have validated the implementation in Java of the L0-to-L1 T2M transformations defined in the *ADMigracMS* method.

- **CSQ2 – Is the ASTM\_PHP model able to represent at PSM level the syntax of the legacy PHP code? Is it possible to represent the ASTM\_PHP model with the tree-like graphical editor we have implemented?**

Yes, the ASTM\_PHP model generated with the model extractor has allowed us to represent accurately the syntax of the legacy PHP code at PSM

level. Therefore, we have validated the ASTM\_PHP metamodel as a correct extension and adaptation of the ASTM standard metamodel.

The tree-like graphical editor has allowed us to identify clearly and intuitively all the different elements of the ASTM\_PHP model as well as to edit the ASTM\_PHP model without errors. Thus, we have validated the proper performance of this tree-like graphical editor.

- **CSQ3 – Is the transformation from the ASTM\_PHP model into the Code model possible? Is it possible to represent the Code model with the tree-like graphical editor we have implemented?**

Yes, the generation of the Code model from the previous ASTM\_PHP model has been executed without errors. Thus, we have validated the correctness of the implementation in ATL of the L1-to-L2 M2M transformation.

The Code model obtained is a validate model conforming correctly to the KDM metamodel. The tree-like graphical editor has allowed us to create and edit properly the Code model identifying each element in a unique manner.

- **CSQ4 – Is the transformation from the Code model to the CMS model at possible? Is it possible to represent the generated CMS model with the UML-like graphical editor we have implemented?**

Yes, the CMS model has been correctly generated from the Code model. The elements migrated from the legacy CMS-based Web application has been represented properly in the CMS model. Furthermore, the attributes and relationships defined for each element within the CMS model captures the enough knowledge to be restructured for the target CMS-based Web application. Thus, we have validated the correct specification of the CMS Common Metamodel.

The UML-like graphical editor implemented to create and edit the CMS model has allowed us to restructure and represent without errors the knowledge being migrated. These elements have been represented intuitively and clearly in the graphical editor. So, we have validated the proper performance of this graphical editor.

Finally, we have validated the correct performance of the rules implemented in ATL that define the L2-to-L3 M2M transformation since we have obtained a validate CMS model. We have proved that the definition of the conditions depending on the CMS platform is crucial to generate the CMS model.

- **CSQ5 – Is it possible to generate the target Code model from the restructured CMS Model?**

Yes, the generation of the target Code model from the restructured CMS model has been executed without any problem. We have validated the correct execution of the transformation rules in ATL that implemet the L3-to-L2 M2M transformation.

This execution depends on the target CMS platform. In our case, it is the Wordpress platform, then the Code elements generated from the CMS elements depend on their implementation on this target CMS platform.

- **CSQ6 – Is it possible to generate the target ASTM\_PHP model from the target Code model?**

Yes, the generation of the target ASTM\_PHP model from the target Code model has been executed without any problem. We have validated the correct execution of the L2-to-L1 M2M transformation.

- **CSQ7 – Is the PHP code which implements the module interpreted correctly by the Wordpress platform?**

Yes, the generation of the PHP code from the target ASTM\_PHP model has been executed properly. We have validated the correct execution of the transformation rules in Accelo that implemet the L1-to-L0 M2T transformation.

The PHP code generated has been interpreted by the Wordpress platform without problems.

## 7.6 Concluding Remarks

In this chapter, we have presented the validation method followed to validate the *ADMigracMS* method and its toolkit. This method is composed of five tasks: the selection of the case studies, the definition of the sources and questions for the data retrieval, the design and execution of the case studies and reporting and conclusions extracted from the validation process.

For this validation, We have applied the migration of two **case studies**: a Web application for the management of a coaching centre (*Coaching Web*) and a Web application for a wellness and nutrition centre (*Websana*).

**Coaching Web** is a CMS-based Web application implemented in Wordpress and it has been migrated to Drupal. We have migrated a block that

displays a clock on the user interface. Otherwise, **Websana** is a CMS-based Web application based on Drupal and it has migrated to Worpdress. Concretely, we have migrated a menu with its menu items, pages and contents.

This validation has allowed us to **assess** the correct specification of the metamodels, the proper performance of the graphical editors and the correct execution of the automated transformations.

*Chapter 8: Conclusions and  
Future Works*

---



In this final chapter, we analyze the achievements reached according to the objectives set at the beginning of the dissertation (presented in Section 8.2); we explain the main contributions of this PhD Thesis (presented in Section 8.1), we present the publications of the scientific results (presented in Section 8.3) and finally, we propose a several open research lines to be carried out as future works (presented in Section 8.4).

## 8.1 Analysis of Achievements

At the beginning of this dissertation, Section 1.2 stated that the main objective of this PhD Thesis: *to specify an ADM-based method which automates the migration of legacy CMS-based Web applications to other CMS platforms.* To tackle this goal a set of partial objectives were established. In the following, the level of completion of these sub-objectives is analyzed in brief.

**Obj 1. Analysis and evaluation of previous research works and initiatives related to the topic of this PhD Thesis.**

To fulfil this objective, Chapter 2 provides two different literature reviews which allow us to identify the *body of knowledge* of this PhD Thesis. These two literature reviews have been based on the guidelines proposed by Kitchenham (B Kitchenham & Charters, 2007) and Biolchini (Biolchini et al., 2005) to perform *Systematic Literature Reviews* (SLRs)

In the first SLR, described in Section 3.2, we have analyzed the existing approaches for the development of CMS-based Web applications. In the second one, presented in Section 3.3, we have analyzed the current approaches based on ADM principles. In both cases, we have posed a set of research questions, presented in Sections 3.2.1 and 3.3.1. These questions have been answered according to a closed set of criteria, presented in Sections 3.2.2 and 3.3.2, defined to extract data from the approaches found.

From the first SLR, we conclude that the development of CMS-based Web applications entails a complex process with specific characteristics that must be addressed by dedicated methods. All the approaches found opt for a top-down development and none of them consider the migration of CMS-based Web applications. We also found that most of the approaches follow the MDE principles.

In the second SLR, we have been able to reassert that there is not any ADM-based method addressing the migration of CMS-based Web applications. Most of the approaches found are focused on the database scope. Moreover, we conclude that the use of the standard metamodels provided by ADM is pretty spread among these reengineering approaches. As for these metamodels, KDM is the most used one to represent at PIM level the data involved in the reengineering process; otherwise the ASTM is less considered by the approaches to represent models at PSM level.

The features of all the approaches found after performing the two SLRs were gathered in order to position the current PhD Thesis within its research scope (the use of ADM metamodels, the legacy code from extracting knowledge or the scope or domain of the approach). The main result of this study exposed the need for the specification of a framework different from those found in the literature. That novel *ADMigracMS* proposal would cope with the migration of CMS-based Web applications following a method based on ADM.

#### **Obj 2. Specification of a metamodel for the CMS domain.**

One of the main objectives of the migration method presented in this PhD Thesis is to represent in a model the data extracted from a legacy CMS-based Web application within the CMS domain. Therefore, we experienced the necessity of defining a metamodel defining the abstract syntax of this models.

This metamodel, namely CMS Common Metamodel, is presented in Chapter 4 where we explain the four-step process followed to define it. In the first step, presented in Section 4.2, we analyzed the CMS market to know which CMS platforms are the ones more used by organizations. We concluded that the three most used CMS platforms were: Wordpress, Joomla! and Drupal; all of them being open-source CMS platforms implemented in PHP. In the second step, explained in Section 4.3, we defined a metamodel for each of the three previous CMS platforms. These CMS metamodels capture the main elements and their relationships of each particular CMS platform. These metamodels are the basis for the definition of the CMS Common Metamodel. In the third step, presented in Section 4.4, we defined a set of *concerns* (*navigation, presentation, behaviour, content* and *user*) that allow us to handle the complexity of the CMS metamodels by organizing their elements in different views. Finally, in Section 4.5, we defined the CMS Common Metamodel considering the **union** of the elements gathered in the three previous CMS metamodels. Some of the elements captured in the CMS Common Metamodel are: *Theme, Menu, MenuItem, Role, Permission* among others.

### **Obj 3. Specification of the *ADMigraCMS* method.**

As we mentioned previously, the main objective of this PhD Thesis is the definition of the *ADMigraCMS* method, an ADM-based method that defines standard guidelines to migrate CMS-based Web applications to other CMS platforms. It is fully defined in Section 5.

As we explained in Section 5.1, the *ADMigraCMS* method is composed of three reengineering stages (Chikofsky & Cross, 1990) defining a “horseshoe” process: *reverse engineering stage*, *restructuring stage* and *forward engineering stage*. During the *reverse engineering stage*, the knowledge from the code that implements the legacy CMS-based Web application is extracted and represented in a set of models at different abstraction levels. One of these models (CMS model) represents that knowledge within the CMS domain. The CMS model conforms to the CMS Common Metamodel. In the *restructuring stage*, the CMS model is restructured to adapt the extracted knowledge to the features of the target CMS platform; finally, in the *forward engineering stage*, a set of models are generated from the CMS model at lower abstraction level. From these models, the code that implements the target CMS-based Web application is generated.

This method has been fundamentally based on ADM. Accordingly, it considers models at different abstraction levels as key artefacts to lead the migration process as well as a set of automated transformations to systemize it.

The ***ADMigraCMS* method** is structured in four different *modelling levels*: *Level 0* represents the code that implements the CMS-based Web application (PHP code); *Level 1* represents the code at platform-specific level using a model conforming to the ASTM metamodel (ASTM\_PHP model); *Level 2* specifies the code at platform-independent level with a model conforming to the KDM metamodel (Code model) and *Level 3* represents the code in the CMS domain with a the CMS model. For each modelling level, we have defined a *modelling language* in the form of *Domain Specific Languages* (DSL) such as, the PHP DSL (Level 0), the ASTM\_PHP DSL (Level 1), the KDM\_CODE DSL (Level 2) and the CMS DSL (Level 3). We have defined an abstract syntax with a metamodel and a concrete syntax with a customized graphical notation for each DSL.

The automated transformations between modelling levels have been specified in Section 5.1.2. It considers six automated transformations: **L0-to-L1** defines the T2M transformations between the L0 and L1 modelling levels. It extracts an ASTM\_PHP model from the PHP code ; **L1-to-L2** defines the M2M transformations between the L1 and L2 modelling levels. It generates a Code

model from the previous ASTM\_PHP model; **L2-to-L3** defines the M2M transformations between the L2 and L3 modelling levels. A CMS model is generated from the previous Code Model; **L3-to-L2** defines the M2M transformations between the L3 and L2 modelling levels. From the restructured CMS model, a target Code model is generated; **L2-to-L1** defines the M2M transformations between the L2 and L1 modelling levels. From the target Code model a target ASTM\_PHP model is obtained; finally, **L1-to-L0** defines the M2T transformations between the L1 and L0 modelling levels. The PHP code that implements the target CMS-based Web application is generated from the previous target ASTM\_PHP model.

**Obj 4. Specification of the modelling languages considered in the *ADMigraCMS* method.**

The models considered in each modelling level of the *ADMigraCMS* method have been defined according to a concrete modelling language. These modelling languages have been defined in the form of *Domain Specific Languages* (DSL) that is by specifying an abstract syntax and by providing a corresponding concrete syntax.

For *Level 0*, we have defined a PHP DSL, fully presented in Section 5.2.1. The abstract syntax of this DSL is depicted by defining the PHP metamodel that captures the syntax elements and their relationships of the PHP code.

For the definition of the models at *Level 1*, we have specified an ASTM\_PHP DSL that is presented in Section 5.2.2. The abstract syntax of this DSL is specified by the ASTM\_PHP metamodel that is presented in Section 5.2.2.1. This metamodel is the specialization of the ASTM standard metamodel (divided in GASTM that defines a common core for language modelling and SASTM that represents the specific elements of a concrete programming language). The elements captured in its GASTM domain were not enough to represent all the syntax elements of the PHP code, so that we needed to extend the ASTM by defining in its SASTM domain the specific PHP elements. The concrete syntax of this DSL is defined with a customized graphical notation presented in Section 5.2.2.2.

To define the models at *Level 2*, we have specified the KDM\_CODE DSL that presented in Section 5.2.3. The abstract syntax of this DSL is based on the code and action packages of the KDM standard metamodel proposed by ADM. With the elements of these two packages, explained in Section 5.2.3.1, we were able to define at platform-independent level the syntax elements of the PHP code

and its semantics. Finally, in Section 5.2.3.2 we customized the graphical notation of these elements defining their concrete syntax.

Finally, for models at *Level 3*, we defined the CMS DSL. Its abstract syntax has been defined with the CMS Common Metamodel explained in Chapter 4 and the customization of the graphical notation of these elements has been presented in Section 5.2.4, as its concrete syntax.

**Obj 5. Specification of the automated transformations involved in the *ADMigraCMS* method.**

The automated transformations between the different modelling levels are another key aspect of the *ADMigraCMS* method which allows systemizing the migration process. These automated transformations are defined as T2M, M2M and M2T transformations.

The first step to specify these automated transformations was to define the mappings between the elements of a source metamodel belonging to a concrete modelling level and the elements of a target metamodel defined in another modelling level.

The mappings defined for the L0-to-L1 T2M transformation and L1-to-L0 M2T transformation are defined, in turn, in Section 5.3.1. These mappings define the correspondence among the elements of the PHP metamodel with the elements of the ASTM\_PHP metamodel. Most of these mappings are defined as one-to-one mappings, where one element of the PHP metamodel is mapped to exclusively one element of the ASTM\_PHP metamodel and vice versa.

The mappings defined for the L1-to-L2 and L2-to-L1 M2M transformations are defined in Section 5.3.2. These mappings define the correspondence among the elements of the ASTM\_PHP metamodel with the elements of the KDM metamodel. The mappings contextualized within the L1-to-L2 M2M transformation are defined as many-to-one, where different elements of the ASTM\_PHP metamodel converge to the same element in the KDM metamodel. Otherwise, the mappings for the L2-to-L1 M2M transformation are defined as one-to-many transformations. For the execution of the one-to-many transformations the conditions defined for each element have a relevant role.

Finally, in Section 5.3.3, we have defined the mappings for the L2-to-L3 and L3-to-L2 M2M transformations. These mappings are defined between the elements of the KDM metamodel and the elements of the CMS Common Metamodel. The mappings in the context of the L2-to-L3 M2M transformation are

defined as one-to-many transformations. As occurred previously, the conditions to execute this kind of transformations are a key issue. These conditions are derived from CMS platform on which the legacy CMS-based Web application is based. In Section 5.3.3, we show the conditions for Drupal platform.

#### **Obj 6. Implementation of the *ADMigraCMS* toolkit.**

To support the migration process proposed by the *ADMigraCMS* method, we have implemented a toolkit that we have called *ADMigraCMS* toolkit. In its implementation, presented in Chapter 6, we have implemented the DSLs in the form of graphical editors and the automated transformations defined in our method.

For the PHP DSL, we have implemented successfully the PHP metamodel by using the Xtext framework (presented in Section 6.2.1). The Xtext framework provides a syntax language with which we have implemented all the syntax elements in the PHP metamodel. From the implementation of the PHP metamodel, the Xtext framework allowed us to generate automatically a parser implemented in Java to read a file written in PHP. We have used this parser to implement the L0-to-L1 T2M transformations by constructing a model extractor.

In reference to the ASTM\_PHP DSL, we have defined the ASTM\_PHP metamodel (explained in Section 5.2.2). Although we have not implemented it as an Ecore model since it can be download from the official website of ADM (<http://adm.omg.org/>), we have defined this metamodel by extending and adapting the ASTM standard metamodel with the specific elements of the PHP programming language.

Regarding the CMS DSL, we have implemented the CMS Common Metamodel as an Ecore model by using the EMF framework (explained in Section 6.2.2)

To create and edit models conforming to the ASTM\_PHP DSL and models according to the KDM\_CORE DSL, we have implemented two tree-like graphical editors (explained in Section 6.3). These graphical editors are based on the metamodels (abstract syntax) and the graphical notations (concrete syntax) defined for each DSL. To implement them without problem, we have used the EMF framework. Likewise, we have implemented a UML-like graphical editor to manage models conforming to the CMS DSL. For this implementation, we have used the tool EuGENia which is based on the GMF framework (explained in Section 6.3.2).

About the implementation of L0-to-L1 T2M transformations that extracts an ASTM\_PHP model from the PHP code, we have constructed a model extractor in Java (presented in Section 6.4). It reads the files written in PHP by using the parser obtained with the Xtext framework and generates the elements of the ASTM\_PHP model by using an API which implements in Java the ASTM\_PHP metamodel. This API has been obtained automatically using EMF from the implementation in Ecore of the ASTM\_PHP metamodel.

With regard to the implementation of the M2M transformations defined by the *ADMigraCMS* method, we have implemented a set of transformation rules in ATL. For example, in Section 6.4.2 we explain the implementation for the *L1-to-L2* M2M transformation.

Finally, for the implementation of the L1-to-L0 M2T transformation, we have implemented a code generator in Acceleo (explained in Section 6.4.6).

#### **Obj 7. Validation of the *ADMigraCMS* method.**

To validate the *ADMigraCMS* method and the toolkit, we have executed two case studies. The former is the migration from a CMS-based Web application implemented in *Wordpress to Drupal*. Concretely, we migrate a block that displays a clock in the user interface. This Web application is called *Coaching Web* and it manages a coaching centre. The latter is the migration of from a CMS-based Web application implemented in *Drupal to Wordpress*. Specifically, we migrate a menu with its menu items, pages and contents. The Web applications is called *Websana* and it manages a wellness and nutrition centre

In Chapter 7, we execute the validation method described in Section 2.3.2. We applied this validation method on *Websana* to validate the following points: the correct specification of the metamodels, the proper performance of the graphical editors and the correct execution of the automated transformations.

In Section 7.4.1, we have checked the correct performance of the model extractor which implements the L0-to-L1 T2M transformation as well as the correctness of the ASTM\_PHP DSL and the suitability of the tree-like graphical editor for editing ASTM\_PHP models. To do that, we have extracted from the PHP code that implements the *MenuItems* an ASTM\_PHP model that has been represented with the tree-like graphical editor.

In Section 7.4.2, the accuracy and appropriateness of the L1-to-L2 M2M transformation defined within the *ADMigraCMS* method have been validated. To do that, we have generated the Code model from the previous ASTM\_PHP model.

Furthermore, we have validated the accuracy of the KDM\_CODE DSL representing the generated Code model with the tree-like graphical editor.

In Section 7.4.3, we have validated the correct performance of the L2-to-L3 M2M transformation. To achieve this goal, we have generated the CMS model from the previous Code model. Besides, we have checked the correctness of the CMS DSL and the applicability of the UML-like graphical editor by editing this CMS model. The rest of the M2M transformations have been validated by generating the target Code model and the target ASTM\_PHP model.

Finally, we have validated the correct generation of PHP code from the target ASTM\_PHP model in Section 7.4.7. We have checked the appropriateness of the L1-to-L0 M2T transformation with the generation of the PHP code that implements in Wordpress the *MenuItems*.

Finally, with this validation we conclude that we have successfully met all the objectives and the hypothesis posed within this PhD Thesis.

## 8.2 Main Contributions

This PhD Thesis has resulted in a number of contributions, regarding not only the main asset of this research (the *ADMigracMS* method) but also related with other secondary aspects. Some of them were objectives fixed before addressing this work while others have emerged during its development. They are summarized in the following.

- **A complete study and analysis about the existing approaches addressing the development of CMS-based Web applications.**

One of the contributions of this PhD Thesis is the study about the existing approaches focused on the development of CMS-based Web applications. It is a consistent study since we have followed the guidelines proposed by Kitchenham (B Kitchenham & Charters, 2007) and Biolchini (Biolchini et al., 2005) to perform *Systematic Literature Reviews* (SLRs).

This study has been crucial to focus this PhD Thesis, since we did not find any approach covering the necessity of the organizations to migrate CMS-based Web applications to other CMS platforms. With the proposal of this PhD Thesis, we address this necessity. The results obtained in this study have been published and presented in the conference ICSOFT-2013 (Trias et al., 2013a).

- **A comprehensive study and analysis about the existing ADM-based reengineering approaches.**

Another contribution of this PhD Thesis is the study about the existing ADM-based reengineering approaches. With this study we reaffirmed the absence in the literature of reengineering approaches addressing the migration of CMS-based Web applications. Furthermore, we analyzed the characteristics of this kind of reengineering approaches, such as the use and acceptance of the standard metamodels proposed by ADM.

Also, this study has been performed following an SLR. Part of the results obtained in this SLR have been published in the conference ISD-2014 (Trias, de Castro, López-Sanz, & Marcos, 2014).

- **Specification of an ADM-based method to migrate CMS-based Web applications to other CMS platforms.**

The specification of the *ADMigraCMS* method, an ADM-based migration method to migrate CMS-based Web applications to other CMS platforms, is an important contribution. This method addresses the three reengineering stages defined by Chikofsky (Chikofsky & Cross, 1990) defining a horseshoe migration process.

To assure the robustness of this method we have based it on the standard metamodels proposed by ADM, mainly on ASTM and KDM. Moreover, the *ADMigraCMS* method allows representing the knowledge involved in the migration process in the CMS domain by using the CMS model that conforms to the CMS Common Metamodel, one of the cornerstones of this PhD Thesis.

This method does not only allow the migration of the elements from a legacy CMS-based Web application, but also to model and generate the code of new CMS elements for the target CMS-based Web application. It is possible during the restructuring stage by using the CMS model.

- **Development of a toolkit supporting the ADM-based migration method proposed within this PhD Thesis.**

The *ADMigraCMS* method is supported by a toolkit called *ADMigraCMS* toolkit. This toolkit implements the DSLs defined within the *ADMigraCMS* method by means of graphical editors. It also implements all the transformations (T2M, M2M, M2T) and allows their execution in order to automate the migration process proposed.

- **Definition and implementation of a PHP metamodel.**

We have defined and implemented a PHP metamodel which captures the syntax elements of the PHP code. In the study of the ADM-based reengineering approaches, we have found metamodels for Java , SQL-92 or JSP metamodels, but none for the PHP programming language.

In reference to the implementation of this metamodel, we have implemented it by using the syntax language provided by the Xtext framework. From this implementation, we have obtained automatically its implementation as an Ecore model by using the Xtext framework.

- **Use and extension of the ASTM standard metamodel.**

We have defined the ASTM\_PHP metamodel that is the extension of the ASTM standard metamodel. We have extended and redefined the definition of some elements of this standard metamodel with specific elements of the PHP programming language.

With this extension we have proved the usability, adaptability and scalability of the ASTM standard metamodel. Furthermore, we have identified some weaknesses such as its excessive object-orientation programming perspective or the ambiguity of some elements defined within the GASTM domain, that might be considered in next versions of this metamodel. We think that it is a relevant contribution for the research community.

- **Definition and implementation of a model extractor that extracts from PHP code models conforming to the ASTM metamodel.**

Another contribution of this PhD Thesis is the definition of a model extractor that allows to extract models from the PHP code. In the study of the ADM-based reengineering approaches, we have found initiatives extracting models from Java, JSP or SQL-92, but none of them focused on the extraction of models from PHP code.

- **The definition of a DSL for the ASTM and KDM standard metamodels as well as the implementation of a graphical editor supporting them.**

Within this PhD Thesis, we have used the implementation of the ASTM and the KDM standard metamodels from the official website of ADM (<http://adm.omg.org/>), to base the ASTM\_PHP and KDM\_CODE DSLs.

For each DSL, we have defined the graphical notation of the elements of both metamodels as well as the implementation of tree-like graphical editors which allow the creation and edition of models conforming to these DSLs.

These tree-like graphical editors allow developers to represent graphically the knowledge extracted from the legacy code being migrated in a clear and intuitive manner.

- **Definition and implementation of the CMS DSL along with the implementation of a graphical editor.**

One of the cornerstones of this PhD Thesis is the CMS Common Metamodel and the DSL based on this metamodel, called CMS DSL. To define this metamodel we have analyzed three of the most used open-source CMS platforms in the market: Wordpress, Joomla! and Drupal.

Based on the CMS DSL, we have implemented a UML-like graphical editor for creating and editing models conforming to this DSL. This graphical editor allows developers to represent graphically in a model the legacy code in the form of elements of the CMS domain. This graphical editor allows identifying clearly all the elements being migrated. Moreover, it allows restructuring the CMS model in an intuitive manner. Therefore, it allows developers to redefine elements, add new ones or delete those that are not interesting to be migrated in the target CMS-based Web application.

### 8.3 Scientific Results

Some of the results of this PhD Thesis have been published in different forums, both national and international. All these publications have allowed us to validate the proposal presented in this PhD Thesis by the scientific community. In the following, those publications are grouped according to the type of publication.

#### Articles in Iberoamerican Journals

- Feliu Trias, Iván Santiago Viñambres, Juan Manuel Vara, Valeria de Castro. M2DAT-HYMO: Una herramienta basada en MDA para la generación automática de aplicaciones Web a partir del modelo del hipertexto. *Revista (RISTI) Revista Ibérica de Sistemas y Tecnologías de Información*. Vol.: 6, pp: 31-44. ISSN: 1646-9895. Porto (Portugal). Dec, 2010.

### **Chapters in International Books**

- Feliu Trias, Valeria de Castro, Marcos López Sanz y Esperanza Marcos. Reverse Engineering Applied to CMS-Based Web Applications Coded in PHP: A Proposal of Migration. *Evaluation of Novel Approaches to Software Engineering - 8th International Conference, ENASE 2013*. Eds. Filipe, J., Maciaszedk, L.A. Vol. 417. pp: 241 - 256. ISBN: 978-3-642-54091-2. Berlin (Germany). Dic, 2013.

### **Articles in International Conferences**

- Feliu Trias, Valeria de Castro, Marcos López Sanz y Esperanza Marcos. A Toolkit for ADM-based Migration: Moving from PHP code to KDM Model in the Context of CMS-based Web Applications. *23rd International Conference on Information Systems Development (ISD2014)*. (Accepted). Varaždin (Croatia). Sep, 2014. (**CORE A**).
- Feliu Trias, Valeria de Castro, Marcos López Sanz y Esperanza Marcos. A Systematic Literature Review on CMS-based Web Applications. *8<sup>th</sup> International Conference on Software Engineering and Applications (ICSOFT-EA)*. pp: 132-140. ISBN: 978-989-8565-68-6. Reykjavik (Iceland). Jul, 2013. (**CORE B**).
- Feliu Trias, Valeria de Castro, Marcos López Sanz y Esperanza Marcos. An ADM-based Method for migrating CMS-based Web applications: extracting ASTM models from PHP code. *1st Workshop on Software Evolution and Modernization – SEM 2013, in conjunction with the 8th International Conference on Evaluation of Novel Software Approaches to Software Engineering – ENASE 2013*. pp: 85-92. ISBN: 978-989-85-66-2. Angers (France). Jul, 2013. (**CORE B**).
- Feliu Trias, Valeria de Castro, Marcos López Sanz y Esperanza Marcos. An ADM-based Method for migrating CMS-based Web applications. *25<sup>th</sup> International Conference on Software Engineering & Knowledge Engineering (SEKE 2013)*. pp: 256-261. ISBN: 978-1-891706-33-2. Boston (EEUU). Jun, 2013. (**CORE B**).
- Feliu Trias. Building CMS-based Web Applications Using a Model-driven Approach. *6th International Conference on Research Challenges in Information Science (RCIS 2012)*. pp: 1 - 6. ISBN: 978-1-4577-1938-7. València. Mar, 2012. (**CORE B**).

#### **Articles in Iberoamerican Conferences**

- Feliu Trias, Valeria de Castro, Marcos López Sanz y Esperanza Marcos. Aplicación de ADM para la migración de aplicaciones Web a plataformas CMS. *10th Conferencia Latinoamericana en Informática (CLEI 2014)*. (Accepted). Montevideo (Uruguay). Sep, 2014. (CORE C).

#### **Articles in National Conferences**

- Feliu Trias, Valeria de Castro, Marcos López Sanz y Esperanza Marcos. Estudio comparativo de diferentes propuestas dirigidas por modelos para la implementación de RIAs. *XVII Jornadas de Ingeniería del Software y Bases de Datos – JISBD 2012*. pp: 439-452. ISBN: 978-84-15487-28-9. Almería (Spain). Sep, 2012.
- Feliu Trias, Valeria de Castro, Marcos López Sanz y Esperanza Marcos. Definición del dominio de las aplicaciones Web basadas en CMS: un Metamodelo Común para CMS. *XVI Jornadas de Ingeniería del Software y Bases de Datos – JISBD 2011*. pp: 169-182. ISBN: 978-84-9749-486-1. A Coruña (Spain). Sep, 2011.

### **8.4 Future Works**

Despite the contributions made on this PhD Thesis, as the research tasks progressed, several directions to further work were detected. Some of them were just not considered as objectives of this PhD Thesis while others have emerged during the development of this work. Next subsections summarize some of them.

- **The refinement of the CMS Common Metamodel.**

As we have explained in Chapter 4, we have defined the CMS Common Metamodel which captures all the elements conforming a CMS-based Web application, thus we have defined the CMS domain.

As we have explained, it has been created with the union of the elements extracted from Drupal, Joomla! and Wordpress. As a future work, we propose the analysis of other CMS platforms from different scopes (e-commerce, web services or documents management). Also we are interested in analyzing a set of proprietary CMSs apart from the open-source ones as well as CMSs implemented in other programming languages such as Java, Perl, Python, ASP.NET or Ruby on the Rails.

With this analysis we intend to find new elements or redefined the existing ones to extend and improve the current version of the CMS Common Metamodel.

- **The improvement of the *ADMigraCMS* method.**

As a future work, we propose the continuous improvement of the *ADMigraCMS* method. The improvements can be sum up in the following aspects.

Up to now, the *ADMigraCMS* method is focused on the code of the legacy CMS-based Web application to extract the knowledge involved in the reengineering process, thus we carry out a static analysis. In the future, we propose to execute, in addition, a dynamic analysis of the CMS-based Web application. For instance, this dynamic analysis can be focused on the obtainment of event logs, as it occurs in (Pérez-Castillo et al., 2011) to make the *ADMigraCMS* method a more thorough migration method. It can suppose the inclusion of new models.

Currently, the *ADMigraCMS* method is focused on the migration of CMS-based Web applications implemented in PHP. One of the improvements is centred in considering CMS-based Web applications implemented in other programming languages such as Java, Perl or Ruby to be migrated with the *ADMigraCMS* method.

Finally, to improve this migration method we propose the extraction of information from the databases that contain all the data managed by the CMS-based Web application.

- **The improvement of the *ADMigraCMS* toolkit.**

So far, the automated transformations are executed sequentially by the *ADMigraCMS* toolkit. One of the future works is to encapsulate all these transformations in a unique and standalone plugin to allow developers to generate the target CMS-based Web application in one click. Concretely, this plugin would be defined in two steps. The first step would allow developers to generate the CMS model from the code of the legacy CMS-based Web application and the second step would permit them to generate the code implementing the target CMS-based Web application from the restructured CMS model.

Also, we propose the improvement of the graphical editors defined within the *ADMigraCMS* method. In this PhD Thesis, we propose a first release of the graphical editors which support the edition of models conforming the DSLs defined within the *ADMigraCMS* method. Furthermore, these graphical editors can be validated thoroughly with the application of more case studies.

- **To analyze the quality of the automated transformations considered within the *ADMigraCMS* method.**

It is evident that the quality of software constructed is a key differentiator, increasingly, considered, more and more, by organizations focused on the software development (Dorling, 1993), (Herbsleb, Zubrow, Goldenson, Hayes, & Paultk, 1997) and (Rothenberger, Kao, & Van Wassenhove, 2010).

For the model-driven reengineering, the quality of automated transformations directly affects the artefacts obtained (Sendall & Kozaczynski, 2010), since the models that define the system are the direct result of the execution of these transformations (Gerber et al., 2002), (Selic, 2003), (Bézivin, 2004b) and (Tratt, 2005). Therefore, it is required to ensure the quality of the transformations that guide the process of reengineering.

Overall, one of the methods used to evaluate software quality is the use of metrics. Since, in the field of model transformations, there already are some works in this direction (van Amstel & van den Brand, 2010) and (van Amstel, van den Brand, & Nguyen, 2010), we propose the analysis of such works in order to determine their applicability for assessing the quality of the transformations defined at different levels of abstraction in the *ADMigraCMS* method.

- **To improve the validation of the *ADMigraCMS* method and toolkit.**

For the validation of the *ADMigraCMS* method, we have used two simulated case studies which have allowed us to validate the proper performance and scalability of the *ADMigraCMS* method and toolkit. As a future work, we are interested in improving this validation taking into account real-world case studies that allow proving the *ADMigraCMS* method and toolkit in a business environment.

Applying this kind of validation allows increasing the reliability the *ADMigraCMS* method and toolkit and allows its market positioning in the future.

Moreover, it would be interesting to assess our approach by external users of different profiles (developers, analysts or project managers). These evaluations would consist in the use and experimentation of the *ADMigraCMS* toolkit by these users and then the evaluation through surveys so that they would allow us to know their level of satisfaction in relation to the usability.

- **To evaluate the DSLs defined in the *ADMigracMS* method.**

Within the *ADMigracMS* method we have defined three DSLs, ASTM\_PHP DSL, KDM\_CODE DSL and CMS DSL. Another future work would be the evaluation of these DSLs in a systemized manner.

One of the aspects to evaluate is the cognitive effectiveness of their graphical notation (concrete syntax) according to a set of solid principles based on the theoretical and empirical evidence proposed by as Moody's Physics of Notations (Moody, 2009).

- **To validate the models considered in the *ADMigracMS* method.**

The validation of the models involved in a process of model-driven development can be considered as a measure of quality, since it allows to detect modelling errors and establish constraints that can not be covered by the definition of metamodels (Molina et al., 2007) and (Rossini, Mughal, Wolter, Rutle, & Lamo, 2011). In general, the validation rules are set at metamodel level, specifying what conditions must meet each element and relationships. Therefore, the effort to implement the validation rules using some of the languages (EVL, OCL, etc...) is directly proportional to the number of elements and relationships in the metamodel.

Thus, if the model validation is considered a means to improve the quality of models, some method must be established to facilitate the implementation of the validation rules, especially if they are complex and numerous.

- **To combine the *ADMigracMS* method with other ADM-based reengineering approaches.**

Another future work would be to combine the *ADMigracMS* method with other ADM-based reengineering approaches which allow us to migrate other aspects of the CMS-based Web application do not tackled by the *ADMigracMS* method.

For instance, to combine the *ADMigracMS* with RationaWeb proposed by García-Rodríguez et al. to migrate the relational database of a legacy CMS-based Web application as well as to combine our method with the approach proposed by Rodríguez-Echevarría et al. with which we could migrate a legacy CMS-based Web application to a RIA.

*Appendix A: Resumen en  
Castellano*

---



Este capítulo ofrece un resumen extendido en castellano de la Tesis Doctoral que se presenta en esta memoria.

En primer lugar se ofrece una perspectiva general de las razones que han llevado a la realización de esta Tesis con el fin de justificar e identificar claramente los problemas de partida que se pretendían abordar en el momento de su realización. A continuación se especifican los objetivos concretos en los que se centra la investigación asociada a la Tesis junto con la metodología de trabajo en la que se ha materializado el trabajo de Tesis Doctoral. Por último, se presentan las conclusiones obtenidas de dicho trabajo.

### **Antecedentes**

Los Sistemas de Información (SI) que utilizan las organizaciones están en continua evolución. Con el tiempo, estos SI pueden quedar obsoletos para estas organizaciones (Lehman, 1998). Estos SI, llamados sistemas heredados, son activos valiosos para las organizaciones ya que integran mucho del conocimiento de negocio de éstas (Ian Sommerville, 2002). Por este motivo, se convierten en un problema importante a la hora de mantenerse debido a sus altos costes (altos costes en rehacer la documentación y en reestructuración de la arquitectura) (Polo et al., 2003). El IEEE (IEEE, 1983)define el mantenimiento de software como: "la modificación del producto de software después de la entrega para corregir los fallos, para mejorar el rendimiento u otros atributos, o para adaptar el producto a un entorno cambiante".

Aún así, el mantenimiento de los sistemas heredados suelen ser más barato que volver a desarrollarlos o sustituirlos por otros. El reemplazo con un nuevo IS es arriesgado, ya que tiene un gran impacto en los aspectos tecnológicos, económicos y humanos (Sneed, 2005). Desde el punto de vista tecnológico, las funcionalidades específicas se pueden perder debido a los cambios tecnológicos, por otra parte, el aspecto humano se ve afectado ya que la sustitución implica reentrenamiento de los usuarios con el fin de que entiendan el nuevo SI. Por último, el aspecto económico también se ve afectado ya que la sustitución de un sistema heredado puede exceder el presupuesto de una organización, así como puede implicar la pérdida del conocimiento de negocio valioso almacenado en el tiempo.

Por otro lado, el mantenimiento de software es la mejor solución ya que permite preservar el conocimiento de negocio embebido en el SI legado así como ayuda a controlar otros costes (Bennett & Rajlich, 2000), (Bianchi et al., 2003).

Durante los últimos años, la *Reingeniería* (Chikofsky & Cross, 1990) ha sido la principal herramienta para abordar el mantenimiento del software de los sistemas heredados (Bianchi et al., 2003), ya que facilita la reutilización de los artefactos software y preserva el conocimiento de negocio incorporado en el sistema a un coste tolerable (Bennett & Rajlich, 2000). Sin embargo, la reingeniería implica dos inconvenientes (Pérez-Castillo, Guzmán, et al., 2011); primero, la reingeniería para grandes sistemas heredados complejos conlleva altos costes de mantenimiento (Canfora & Penta, 2007); segundo, estos procesos se llevan a cabo de una manera ad-hoc no normalizada que dificulta su automatización (Kazman et al., 1998). Por estas dos razones, la industria del software está exigiendo procesos de reingeniería estandarizados y automatizados.

En los últimos años, *Arquitectura-Driven Modernization* (ADM) (Khusidman & Ulrich, 2007), (OMG, 2009), (Sadovsky, Vigier, Hoffmann, et al., 2009a), (Pérez-Castillo, Guzmán, et al., 2011) ha demostrado ser una solución importante para los inconvenientes mencionados. ADM propone llevar a cabo procesos de reingeniería teniendo en cuenta los principios del Desarrollo Dirigido por Modelos (*Model-Driven Development*, MDD) (Miller & Mukerji, 2003), (Bézivin, 2004a). En concreto, se basa en los principios de la Model-Driven Architecture (MDA) (Frankel, 2002), (Flore, 2003), (Kleppe & Warmer, 2003), (Miller & Mukerji, 2003), (Mellor et al., 2004), (Guttman & Parodi, 2006), (Selic, 2008), (Watson, 2008). ADM se centra en la formalización del proceso de reingeniería usando modelos que representan los artefactos software del sistema heredado a diferentes niveles de abstracción –Modelos Específicos de Plataforma (*Specific Platform Models*, PSM), Modelo Independiente de Plataforma (*Platform Independent Model*, PIM) y Modelo Independiente de Computación (*Computation Independent Model*, CIM)- y en la automatización del proceso mediante transformaciones automáticas. Entre los beneficios proporcionados por ADM destacamos los siguientes (OMG, 2007): 1) revitalización de los SI legados, 2) reducción de los costes de mantenimiento, 3) extensión de la vida útil de los SI legados, 4) mejora del ROI (Return Of Investment) de estos sistemas y 5) facilitación de la migración con otros sistemas y otros entornos.

Además de seguir los principios de MDA, ADM define un conjunto de metamodelos estándar (siete metamodelos) para representar de una manera formalizada la información involucrada en el proceso de reingeniería.

Actualmente, de los siete metamodelos, hay tres disponibles: ***Abstract Syntax Tree Metamodel*** (ASTM) (OMG, 2005a), ***Knowledge Discovery Metamodel*** (KDM) (ISO/IEC, 2012) y ***Structured Metrics Metamodel*** (SMM) (OMG, 2012b). Además de apoyar el proceso de reingeniería de software, estos metamodelos estándar permiten a los desarrolladores ahorrar tiempo y esfuerzo en la creación de sus propios metamodelos. Los modelos conforme a ASTM (modelos ASTM) permiten representar los artefactos software que componen un sistema heredado en forma de árboles sintácticos (*Abstract Syntax Tree*, AST) (Fischer et al., 2007) a un nivel específico de plataforma. Por otro lado, los modelos conforme a KDM (modelos KDM) permiten representar la semántica y la sintaxis de estos artefactos software a un nivel independiente de la plataforma. Por último, se utilizan los modelos conformes a la SMM para describir métricas y mediciones en modelos KDM.

La clave para la automatización de un proceso de reingeniería es la definición de transformaciones automáticas. Es posible definir tres tipos de transformaciones: transformaciones de texto a modelo (*text to model*, T2M), los cuales permiten obtener un modelo a partir de los artefactos de software del SI heredado; transformaciones de modelo a modelo (*model to model*, M2M), que permiten aumentar o disminuir el nivel de abstracción de los modelos; y transformaciones modelo a texto (*model to text*, M2T), que generan el código que implementa un SI a partir de un modelo (Gerber et al., 2002), (Selic, 2003), (Bézivin, 2004a), (Tratt, 2005). Estas transformaciones hacen posible unir todos los pasos del proceso de reingeniería y la definición de un llamado proceso de herradura compuesto de tres etapas como propone (Chikofsky & Cross, 1990): la **fase de ingeniería inversa**, es la etapa de la parte izquierda de la herradura, que analiza el SI legado y sus interrelaciones con los modelos en diferentes niveles de abstracción; la **fase de reestructuración**, es la parte superior de la curva de la herradura donde los modelos obtenidos en la fase de ingeniería inversa se reestructuran para adaptarlos a los SI destino; y la **fase de ingeniería directa**, es la parte derecha de la herradura que genera el código que implementa el SI destino

En los últimos años, la World Wide Web se ha convertido en una plataforma para aplicaciones empresariales sofisticadas que soportan complejos procesos de negocio (Souer et al., 2009). Las organizaciones usan Internet para apoyar sus procesos de negocio y obtener ventajas competitivas, para llevar a cabo una colaboración mundial y una integración con los socios externos (Turban et al., 1999) (Lee & Shirani, 2004). Como resultado, el número de aplicaciones Web

(Isakowitz et al., 1998) desarrolladas por la industria y usadas por las organizaciones ha aumentado en gran medida.

Al mismo tiempo, el volumen de contenido digital gestionado por estas organizaciones también ha aumentado exponencialmente en la última década. Por lo tanto, estas organizaciones han experimentado la necesidad de utilizar herramientas de gestión potentes para mantener sus aplicaciones web de forma constante (McKeever, 2003) (Souer, Weerd, et al., 2007). En ese sentido, los Sistemas de Gestión de Contenidos (*Content Management Systems*, CMS) se han convertido en la herramienta más eficaz que permite recoger, gestionar y mantener enormes cantidades de contenido digital de una manera robusta y fiable (Boiko, 2001). Por lo tanto, muchas de las aplicaciones Web han sido implementadas sobre estas plataformas CMS (Aplicaciones Web basadas en CMS). En comparación con las aplicaciones Web tradicionales, este tipo de aplicaciones proporcionan unos beneficios, que se presentan a continuación:

- **Creación dinámica de contenido:** el contenido se crea y se agrega de forma dinámica por los usuarios no técnicos de la Web, sin necesidad de la intervención del *webmaster*. Permite superar los cuellos de botella que se producen en el role del *webmaster*.
- **La separación entre el contenido y el diseño:** el diseño gráfico de la página Web se almacena en una plantilla y el contenido se almacena en una base de datos o un documento separado. Evita posibles inconsistencias en el *look-and-feel* de la página Web.
- **Diferentes niveles de autorización de acceso:** los CMS permiten la definición de roles con diferentes niveles de permisos y derechos de acceso a los contenidos gestionados por la aplicación Web. Permite proteger mejor el acceso al contenido.
- **Extensión de funciones:** CMS permite ampliar la funcionalidad de las aplicaciones Web con la inserción de nuevos módulos funcionales. De esta manera, se reduce el tiempo en el proceso de desarrollo.

El mercado de CMS está en constante evolución, por lo que podemos encontrar un gran número de plataformas CMS que ofrecen diferentes ventajas y adaptadas a diferentes dominios, como los blogs, comercio electrónico o *e-learning* (Shreves, 2011). Es posible encontrar plataformas CMS propietarias como SDL ACM (ACM, 2014), Verse contenido (Verse, 2014) o Cascade Server

(Hill, 2014), así como de código libre como Drupal (Drupal, 2014), Joomla! (Joomla!, 2014) o Wordpress (Wordpress, 2014).

Este hecho, junto con los posibles cambios en los objetivos de las organizaciones, puede hacer que éstas se vean con la necesidad de migrar sus aplicaciones Web basadas en CMS heredadas a otras plataformas CMS más modernas que satisfagan mejor sus necesidades. Este proceso de migración implica un proceso complejo, largo y propenso a errores. Por lo tanto, es necesario darle soporte mediante un método para mitigar todos estos inconvenientes. En la actualidad, la migración de aplicaciones Web basadas en CMS a otras plataformas CMS no se aborda de acuerdo con la revisión de la literatura publicada en (Trias et al., 2013a).

Por lo tanto, para resolver este vacío, definimos en esta tesis doctoral el método *ADMigraCMS* como un medio para apoyar este proceso de migración. Teniendo en cuenta las ventajas de ADM explicadas anteriormente, decidimos basar nuestro método sobre esta propuesta. Además, nuestro método se compone de tres fases que definen un proceso de reingeniería de herradura: fase de ingeniería inversa, fase de reestructuración y fase de ingeniería directa.

Hasta el momento, se centra en la migración de las aplicaciones Web basadas en CMS implementadas en código libre. En particular, consideramos plataformas como Drupal, Joomla! o Wordpress, debido a su uso extendido y la gran aceptación en la industria y los mercados a nivel mundial (Shreves, 2011). La mayoría de estas plataformas CMS de código abierto están implementadas en PHP, así que prestamos especial atención a este código.

### **Objetivos**

En esta sección se describen la hipótesis de partida de esta Tesis Doctoral junto con los objetivos que se derivan de ésta.

La **hipótesis formulada** en esta tesis doctoral es que - "En un mundo en evolución como el mundo de Ingeniería Web y en un mercado cambiante como el mercado de la CMS, es factible ofrecer a las organizaciones un método que les permite migrar sus aplicaciones Web basadas en CMS a otras plataformas CMS que cubran mejor sus necesidades."

El **objetivo principal** de la Tesis Doctoral presentada, que se deriva directamente de la hipótesis anterior, es, por lo tanto: "especificar un método basado en ADM que automatice la migración de las aplicaciones web basadas en CMS a otras plataformas CMS".

Este objetivo ha sido desglosado en los siguientes subobjetivos:

**Obj 1.** Análisis y evaluación de enfoques relacionados con el tema de esta Tesis Doctoral. Teniendo en cuenta que esta Tesis Doctoral se apoya en dos áreas claramente identificadas de interés dentro del campo de la reingeniería de software, este objetivo se puede dividir de la siguiente manera:

**Obj 1.2.** El estudio detallado de las iniciativas en el ámbito de las aplicaciones Web basadas en CMS centrándonos en las propuestas que aborden el desarrollo o la migración de este tipo de aplicaciones Web.

**Obj 1.3.** Estudio detallado de los enfoques actuales de reingeniería dirigido por modelos.

**Obj 2.** Especificación de un metamodelo para el dominio de los CMS. Este metamodelo se llama CMS Common Metamodel y representa los conceptos clave de las aplicaciones Web basadas en CMS.

**Obj 3.** Especificación del método *ADMigraCMS*, presentado en esta Tesis Doctoral para llevar a cabo la migración de las aplicaciones Web basadas en CMS. Este objetivo se puede dividir de la siguiente manera:

**Obj 3.1.** Definición del proceso del método *ADMigraCMS*.

**Obj 3.2.** Definición de los niveles de modelado (Nivel 0, Nivel 1, Nivel 2 and Nivel 3).

**Obj 3.3.** Definición de las transformaciones automatizadas entre niveles de modelado.

**Obj 4.** Especificación de los lenguajes de modelado considerados en el método *ADMigraCMS*. Estos lenguajes de modelado se definen como Lenguajes Específicos de Dominio (*Domain Specific Language, DSL*).

**Obj 4.1.** Definición del DSL PHP (Nivel 0).

**Obj 4.2.** Definición del DSL ASTM\_PHP (Nivel 1).

**Obj 4.3.** Definición del DSL KDM\_CODE (Nivel 2).

**Obj 4.4.** Definición del DSL CMS (Nivel 3).

**Obj 5.** Especificación de las transformaciones automatizadas del método *ADMigraCMS*.

**Obj 5.1.** Definición de los mapeos entre metamodelo PHP y el metamodelo ASTM\_PHP.

**Obj 5.2.** Definición de los mapeos entre el metamodelo ASTM\_PHP y metamodelo KDM.

**Obj 5.3.** Definición de los mapeos entre el metamodelo KDM y el CMS Common Metamodel.

**Obj 6.** Implementación de la herramienta que da soporte al método *ADMigracMS*. Este objetivo se puede dividir de la siguiente manera:

**Obj 6.1.** Implementación de los metamodelos que definen la sintaxis abstracta del DSL.

**Obj 6.2.** Implementación de un editor gráfico para cada DSL.

**Obj 6.3.** Implementación de las transformaciones automatizadas.

**Obj 7.** Validación del método de ADMigracMS así como de la herramienta que lo soporta. Aplicamos el método ADMigracMS a dos casos de estudio para validar el enfoque metodológico y el correcto funcionamiento de la herramienta.

## Metodología

El método de investigación utilizado en esta Tesis está adaptado del propuesto por Marcos & Marcos (Marcos & Marcos, 1998) para la investigación en el ámbito de la Ingeniería del Software. Este método se basa en el método hipotético-deductivo de Bunge (Bunge, 1979), que se compone de varios pasos, suficientemente generales, para ser aplicados a cualquier tipo de actividad investigadora. Las principales etapas del proceso de investigación seguido para completar la Tesis Doctoral actual se muestran en la Figure 9-1.

La primera etapa de este método de investigación es la **identificación del cuerpo de conocimiento** y un conjunto de problemas dentro de este contexto que necesita ser abordado. El resultado obtenido es el estado del arte de esta Tesis Doctoral que es la base teórica sobre la que basamos nuestra investigación.

La segunda etapa, como podemos ver en la Figure 9-1., es la **definición del problema**. A partir de la identificación del cuerpo de conocimiento y los problemas potenciales descubiertos, determinamos el problema a resolver.

La tercera etapa es la **definición de la hipótesis**. Después de la definición del problema, la hipótesis se formula como un conjunto de objetivos que cumplir para llegar a la solución. Esta solución debe verificar la hipótesis planteada anteriormente.

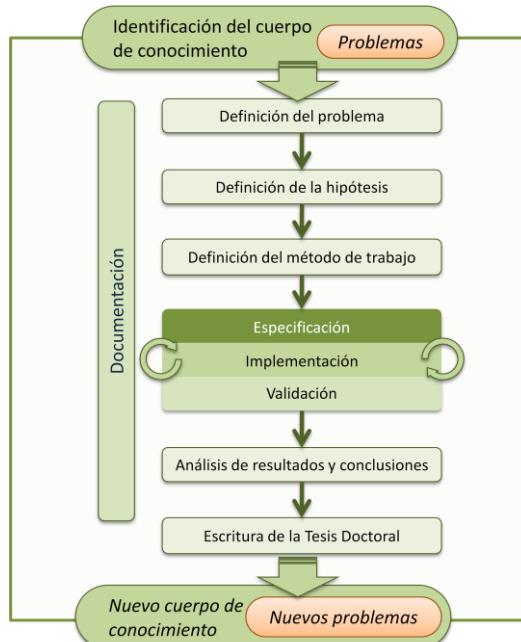


Figure 9-1. Método de investigación.

La cuarta etapa es la **definición del método de trabajo**, que permite realizar la especificación, implementación y validación de nuestra investigación. Este es un aspecto necesario debido a la capacidad de adaptación del método a diferentes contextos. Cada proyecto de investigación tiene sus propias características intrínsecas y por lo tanto no hay un método universal para ser aplicado a todo tipo de investigación.

La quinta etapa es la que corresponde a la etapa de **especificación, implementación y validación**. Esta etapa es de gran interés, ya que representa el núcleo de esta tesis doctoral. Se explica a continuación.

En la sexta etapa se lleva a cabo un **análisis de resultados y conclusiones** a partir de todo el trabajo realizado durante la investigación en esta Tesis Doctoral.

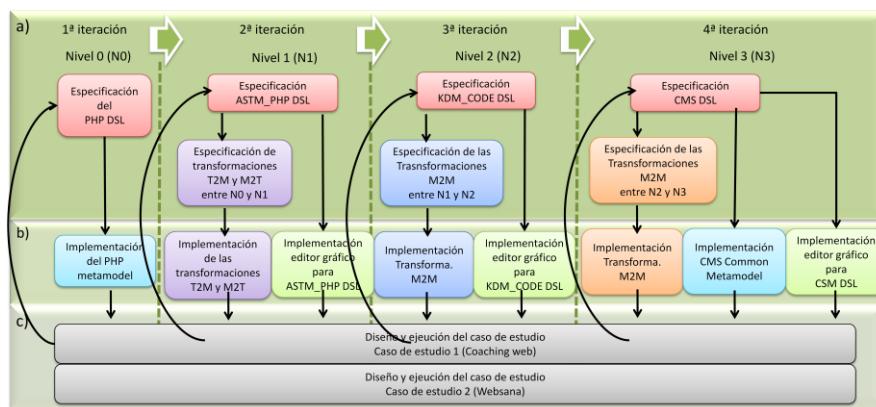
La séptima etapa es la etapa de la **escritura de la Tesis Doctoral** de reunir toda esta experiencia de investigación en una Tesis final.

Aunque el paso final del proceso de investigación puede ser realizada en la tarea de escribir la tesis doctoral, cualquier actividad de investigación establece un nuevo conjunto de conocimientos que se forma mediante la incorporación de todos los objetos de investigación como parte de ella. Esta situación se deriva de nueva

creación de nuevos problemas, nuevo cuerpo de conocimiento, que pueden ser objeto de tareas e iniciativas de investigación futuras.

### *Especificación, Implementación y Validación*

El método escogido en esta Tesis Doctoral para la fase de Especificación, Implementación y Validación es conforme a un modelo de proceso iterativo e incremental. Esta fase de investigación itera e incrementa la propuesta a través de la realización de las tareas mostradas y retroalimentándose entre una tarea y otra. Una visión genérica de este proceso se muestra en la Figure E-9-2



**Figure E-9-2. Definición de la etapa de a) especificación, b) implementación and c) validación.**

Elegimos de proceso iterativo e incremental debido a la naturaleza del problema a resolver en esta Tesis Doctoral. Es similar a los problemas que aparecen en un desarrollo de software real. La especificación, implementación y validación no representan un proceso sencillo, sino un proceso iterativo en el que los flujos de información transcurren hacia adelante y hacia atrás para mejorar cada una de las tareas de investigación. Como podemos observar, este proceso es ejecutado por cuatro iteraciones consecutivas. A continuación, le explicamos estas iteraciones.

**Primera iteración** centra su atención en la especificación, implementación y validación del metamodelo PHP. En la especificación del metamodelo PHP, capturamos los elementos que definen la sintaxis de este código. Para su implementación, se utiliza el lenguaje Xtext del framework Xtext. Por último, sus validación se lleva a cabo por pruebas compuestas de fragmentos de código PHP extraídos de los dos casos de estudio definidos para la validación de nuestra investigación.

**Segunda iteración** se centra en el DSL ASTM\_PHP. En su especificación definimos su sintaxis abstracta con el metamodelo ASTM\_PHP. En cuanto a su sintaxis concreta, definimos la notación gráfica para cada elemento de este metamodelo. A partir de esta sintaxis concreta, implementamos un editor gráfico en forma de árbol para crear modelos ASTM\_PHP. Por otro lado, se especifica las transformaciones bidireccionales entre el metamodelo PHP y el metamodelo ASTM\_PHP. Para su implementación, implementamos un extractor de modelos que genera un modelo ASTM\_PHP a partir del código PHP. Este extractor de modelos está implementado en Java. Por otro lado, las transformaciones M2T que generan el código PHP a partir de un modelo ASTM\_PHP se implementan con un conjunto de reglas de transformación en *Acceleo*. Finalmente, la validación del metamodelo y las transformaciones se llevan a cabo mediante los dos casos de estudio propuestos.

**Tercera iteración** se centra en el DSL KDM\_CODE. Para su especificación, no definimos un nuevo metamodelo para definir su sintaxis concreta, sino que utilizamos el metamodelo estándar de KDM. Del conjunto de paquetes de este metamodelo consideramos dos: los paquetes *code* y *action*. En cuanto a la sintaxis concreta, se especifica una notación gráfica para los elementos de estos dos paquetes. A partir de esta sintaxis concreta, implementamos un editor gráfico en forma de árbol para crear modelos Code. Además, especificamos las transformaciones M2M bidireccionales entre el metamodelo ASTM\_PHP y el metamodelo KDM. La implementación de estas transformaciones M2M se hacen con un conjunto de reglas de transformación implantadas en ATL. Al igual que ocurrió anteriormente, la validación del editor gráfico en forma de árbol y las transformaciones M2M se basan en los dos casos de estudio.

**Cuarta iteración** se centra en el DSL CMS. Para su especificación se define la sintaxis abstracta con el CMS Common Metamodel. En cuanto a su sintaxis concreta, se define la notación gráfica de los elementos de este metamodelo. Este metamodelo se ha implementado como un modelo Ecore utilizando EMF. A partir de la sintaxis concreta se implementa un editor gráfico UML utilizando GMF. Por otro lado, se especifica las transformaciones M2M bidireccionales entre el metamodelo KDM y el CMS Common Metamodel. Para llevar a cabo la implementación de las transformaciones M2M se define un conjunto de reglas de transformación implantados en ATL. Al igual que ocurrió anteriormente, la validación del editor gráfico-UML similar y las transformaciones M2M se basan en los dos casos de estudio.

## Conclusiones

Esta Tesis Doctoral proporciona una serie de contribuciones, no sólo en el ámbito de la investigación planteada como punto de partida (la especificación de *ADMigraCMS* como método ADM para migrar aplicaciones Web basadas en CMS a otras plataformas CMS), sino también en relación con otros aspectos complementarios. Estas contribuciones se resumen a continuación:

- **Un estudio y análisis completo acerca de los enfoques existentes que abordan el desarrollo de aplicaciones Web basadas en CMS.**

Una de las aportaciones de esta Tesis Doctoral es el estudio acerca de los enfoques existentes centrados en el desarrollo de aplicaciones Web basadas en CMS. Se trata de un estudio consistente, ya que hemos seguido las directrices propuestas por Kitchenham (B Kitchenham & Charters, 2007) y Biolchini (Biolchini et al., 2005) para llevar a cabo una revisión sistemática de la literatura (*Systematic Literature Review, SLR*).

Este estudio ha sido crucial para enfocar esta Tesis Doctoral, ya que no se encontró ninguna aproximación que abordase la necesidad de las organizaciones para migrar sus aplicaciones Web basadas en CMS a otras plataformas CMS que cubrieran major sus necesidades. Con la propuesta de esta Tesis Doctoral, cubrimos esta necesidad encontrada. Los resultados obtenidos en este estudio han sido publicados y presentados en la conferencia ICSOFT-2013 (Trias et al., 2013a).

- **Un estudio y análisis amplio acerca de los enfoques existentes de reingeniería basados en ADM.**

Otra aportación de esta Tesis doctoral es el estudio acerca de los enfoques existentes de reingeniería basados en ADM. Con este estudio hemos reafirmado la ausencia en la literatura de los enfoques de reingeniería que abordan la migración de las aplicaciones Web basadas en CMS. En este estudio, se analizaron las características de este tipo de enfoques de reingeniería, como el uso y la aceptación de los metamodelos estándar propuestos por ADM.

Como hicimos con el primer estudio, este estudio se ha realizado mediante la ejecución de una SLR. Parte de los resultados obtenidos en esta SLR se han publicado en la conferencia de ISD-2014 (Trias et al., 2014).

- **Especificación de un método basado en la ADM para migrar las aplicaciones Web basadas en CMS a otras plataformas CMS.**

La principal aportación de esta Tesis Doctoral es la especificación del método *ADMigracMS*, un método de migración basada en ADM para migrar aplicaciones Web basadas en CMS a otras plataformas CMS.

Este método está definido como un proceso en hendidura compuesto de las tres etapas de reingeniería propuestas por Chikofsky (Chikofsky & Cross, 1990): fase de ingeniería inversa, fase de reestructuración y fase de ingeniería directa.

Para asegurar la robustez de este método nos hemos basado en los metamodelos estándar propuesto por ADM, principalmente en ASTM y KDM. Por otra parte, el método *ADMigracMS* permite representar el conocimiento involucrado en el proceso de migración en el dominio CMS mediante el modelo CMS que está conforme al CMS Common Metamodel, una de las principales contribuciones de esta Tesis Doctoral.

Este método no sólo permite la migración de los elementos de una aplicación Web basada en CMS heredada, sino también permite modelar y generar el código de nuevos elementos insertados en la fase de reestructuración mediante el modelo CMS.

- **Desarrollo de una herramienta que dé apoyo al método *ADMigracMS*.**

El método *ADMigracMS* está apoyado por una herramienta llamada *ADMigracMS toolkit*. Hasta el momento, esta herramienta permite la migración de algunos de los elementos que componen las aplicaciones Web basadas en CMS, tales como menús, elementos de menú, páginas y bloques.

La herramienta *ADMigracMS* implementa los DSLs definidos dentro del método *ADMigracMS* por medio de editores gráficos, así como también implementa todas las transformaciones automáticas (T2M, M2M, M2T) que sistematizan el proceso de migración propuesto por nuestro método.

- **Definición e implementación de un metamodelo PHP.**

Otra contribución resultante de esta Tesis Doctoral es la definición e implementación de un metamodelo PHP que captura los elementos de la sintaxis del código PHP. En el segundo estudio acerca de propuestas de reingeniería basadas en ADM, hemos encontrado metamodelos como metamodelo Java, metamodelo SQL-92 o metamodelos JSP, pero no hemos encontrado ninguno

metamodelo PHP que podamos utilizar para basar la implementación de nuestro extractor de modelos.

Hemos implementado en metamodelo PHP con el lenguaje que proporciona el framework Xtext. A partir de esta implementación, el framework Xtext nos ha permitido obtener de forma automática su implementación también en Ecore.

- **Uso y extensión del metamodelo estándar ASTM.**

Otra de las aportaciones es la extensión del metamodelo estándar ASTM con elementos específicos del código PHP. Hemos ampliado y redefinido la definición de algunos elementos de este metamodelo creando el llamado metamodelo ASTM\_PHP.

Con el uso de este metamodelo y su extensión, hemos validado su facilidad de uso y su capacidad de extensión por medio de su dominio SASTM. Por otra parte, hemos identificado algunas debilidades, como su excesivo enfoque en la programación orientada a objetos o la ambigüedad de algunos de los elementos definidos en el dominio GASTM, que podrían ser considerados en próximas versiones de este metamodelo. Creemos que la adaptación de este metamodelo estándar (metamodelo ASTM\_PHP) es una contribución relevante para la comunidad investigadora que puede ser utilizado por otras propuestas de reingeniería.

- **Definición e implementación de un extractor de modelos que extrae modelos ASTM\_PHP a partir de código PHP.**

Otra aportación de esta Tesis Doctoral es la definición de un extractor de modelos que permite extraer modelos ASTM\_PHP a partir del código PHP. En el segundo estudio sobre propuestas de reingeniería basadas en ADM, hemos encontrado algunas de ellas que extraen modelos a partir de código Java, JSP o SQL-92, pero ninguno de ellos se centró en la extracción de modelos a partir del código PHP.

- **La definición de un DSL para los modelos standar ASTM y KDM, así como la implementación de un editor gráfico que soporta estos DSLs.**

En el contexto de esta Tesis Doctoral, hemos definido dos DSLs basados en los metamodelos estándar de ASTM y KDM. Estos dos metamodelos se pueden descargar en la página Web de ADM (<http://adm.omg.org/>).

Por un lado, hemos definido el DSL ASTM\_PHP que se basa en el metamodelo ASTM\_PHP (la extensión del metamodelo ASTM con los elementos

de PHP); por otro lado, hemos especificado el DSL KDM\_CODE definido sobre los paquetes *code* y *action* del metamodelo KDM.

Para cada DSL, hemos definido la notación gráfica de los elementos que conforman los dos metamodelos, así como la implementación de editores gráficos en forma de árbol, que permiten la creación de los modelos que se ajusten a estos DSLs.

- **Definición e implementación del DSL CMS junto con la implementación de un editor gráfico.**

Una de las piedras angulares de esta Tesis Doctoral es la definición del CMS Common Metamodel y el DSL basado en este metamodelo, llamado CMS DSL. No hemos encontrado en la literatura un metamodelo que capture de manera completa e integral los elementos que implementan las aplicaciones Web basadas en CMS, por lo que hemos definido e implementado este nuevo metamodelo

Para definir el CMS Common Metamodel hemos analizado tres de las plataformas CMS de código libre más utilizadas en el mercado: Wordpress, Joomla! y Drupal.

Finalmente, se ha especificado el DSL CMS que define la notación gráfica de los elementos capturados por el CMS Common Metamodel. En base a este DSL, hemos implementado un editor gráfico UML para crear modelos conforme a este DSL.

*Appendix B: Systematic  
Literature Reviews*

---



In Section 2.2 of this dissertation, we have presented the identification of the body of knowledge based on a systematic literature review.

Then, in Section 3.2, we perform a SLR to identify the existing approaches for the development or migration of CMS-based Web applications. Then in Section 3.3, we carry out a SLR to identify the current model-driven reengineering methods. In those sections, we focused on the three of the tasks involved in the SLR process, the *research question definition*, the *data extraction definition* from the *Planning* phase and the *data extraction* from the *Result Analysis* phase.

In this Appendix, we show the whole set of tasks considered during the performance of the two SLRs. In Section 0, we present the tasks performed during the *planning* phase, otherwise in Section 0, we show the tasks involved the *execution* phase.

### **Planning Phase**

The *planning* phase aims for defining the objective of the SLR and the protocol conducting it. This protocol should specify the research questions, the search strategies, the inclusion and exclusion criteria and the criteria for the data extraction to obtain relevant information for our research. In next subsections, we present the tasks to define all these aspects.

#### ***Research Questions Definition***

The first task in this SLR process is to define the objective of the research. To reach this objective, we define a set of research questions (RQ). In the following, we present the objective and the RQ's defined for each SLR performed in this PhD Thesis.

##### *SLR 1 - Approaches centred in CMS-based Web applications*

This SLR is focused on **analyzing those existing approaches which tackle the top-down development of CMS-based Web applications, as well as those ones addressing the migration of this type of Web applications**. For this SLR we have defined the following RQs:

- **RQ1. Which approaches do exist in the literature for addressing a top-down development of CMS-based Web applications?**

We are interested in finding Web Engineering methodological approaches following a typical top-down process for the development of CMS-based Web applications.

- **RQ2. Which approaches do exist in the literature for addressing a reengineering process for CMS-based Web applications?**

We want to know if there are reengineering methods focused on the automation and systematization of the reengineering process for CMS-based Web applications.

- **RQ3. Which of the existing Web Engineering methods have been adapted/extended to cover the specific issues of the development of CMS-based Web applications?**

To date, the research field of Web Engineering has derived in many proposals supporting the complex tasks of designing and creating Web applications. Otherwise, the development of CMS-based Web applications entails complex and time consuming tasks which cannot be addressed by these current Web Engineering methods. Therefore, with this research question we are interested in identifying those Web Engineering methods that, undergoing some adaptations, have been capable to cope with the particularities of a CMS-based Web application.

- **RQ4. Which approaches can be framed within the context of Model-Driven Engineering?**

Nowadays, there is an increasing tendency of traditional Web Engineering methods towards the use of a model-driven paradigm because of its benefits (Schmidt, 2006). Because of that, we are interested in testing if the approaches found in this SLR follow this paradigm. Our intention is to analyze aspects such as the processes proposed, the models defined, the transformations considered, the tools provided, the automatic generation of code, etc.

- **RQ5. Which is the interest of the Web Engineering community in researching about CMS-based Web applications?**

Concretely, we want to know which research groups or organizations are leading the research in this area and which forums publish studies focused on this issue.

*SLR 2 – Model-driven Reengineering Methods*

The main objective of this SLR is to **identify and analyze the existing model-driven methods addressing the reengineering process of Information Systems**. To address this objective we propose the following RQs.

- **RQ1. Which model-driven approaches do exist in the literature for addressing a reengineering process?**

We are interested in finding the existing reengineering methodological approaches based on the model-driven principles to know their features and operative steps.

- **RQ2. In which domain does the Software Reengineering Community dedicate more efforts to define model-driven approaches to automate a reengineering process?**

We are interested in knowing the domain where the approach is executed. This domain could be the databases, the legacy systems in general or Web applications, among others. We know that the CMS-based Web applications domain is not addressed by the reengineering approaches currently, but we want to know which is the domain most tackled by these approaches.

- **RQ3. Which kind of process is defined by the approaches? Which reengineering stages (reverse engineering, restructuring and forward engineering) do they consider in their process?**

We are interested in knowing which engineering stages are addressed by the approach. It is possible to find approaches addressing the three stages, defining a horseshoe process or just addressing just one of them. This issue allows us to know how to address comprehensively a reengineering process.

- **RQ4. Which is the level of automation of the approaches found in the literature? Are they supported by any toolkit?**

We want to know the level of automation of the approach to check the level of maturity of the approach as to avoid errors and to reduce time in the migration process. We analyze which type of transformations (T2M, M2M or M2T) is considered by the approach and whether it is supported by any toolkit allowing to automate the reengineering process.

- **RQ5. Which type of techniques are used by the approaches to extract and model the software artefacts from the legacy system? In which code**

**are the software artefacts implemented? Which is the technique to generate code.?**

We are interested in knowing which techniques are used by the model-driven reengineering approaches to extract the models from the legacy code. Also, we want to know whether any of these approaches use any model extractor from PHP code. If so, we could assess the possibility of use it.

- **RQ6. Which type of techniques are used by the approaches to implement the M2M transformations? Which are the MDA level of the models transformed by these M2M transformations?**

We want to know from the approaches that implement the M2M transformations as part of their process, what transformation language they use to implement them. It can help us to choose the transformation language to implement the M2M transformation of our approach.

- **RQ7: Which is the use of the standard metamodels proposed by ADM among the reengineering approaches?**

We want to know the level of implementation of the ADM initiative. We are interested in knowing if the approaches use the standard metamodels proposed by ADM. Thereby, we can analyze the level of use and acceptance of this initiative.

- **RQ8. Which is the interest of the Web Engineering community in researching about CMS-based Web applications?**

We want to know which research groups or organizations are leading the research in this area and which forums publish studies focused on this issue. It allows to analyze the impact and interest of this topic in the research community.

***Search Strategy Definition***

The *Planning* stage also involves the definition of the search strategy for the studies that contribute to our SLR. This strategy is based on enumerating the data sources (digital libraries) where we will search the studies and on defining the keywords that will be used in this search.

The following digital libraries are selected to carry out the search process for this SLR (Name [Acronym]: website):

- ACM Digital Library [ACM]: <http://portal.acm.org/>
- IEEEXplore [IEEEX]: <http://ieeexplore.ieee.org/>

- ISI Web of Knowledge [ISI]: <http://www.webofknowledge.com/>
- Science Direct [SD]: <http://www.sciencedirect.com/>
- SpringerLink [SL]: <http://www.springerlink.com/>
- Scopus [SCP]: <http://www.scopus.com/home.url>
- Google Scholar [GS]: <http://scholar.google.es/>

All the previously digital libraries are based on search engines that works with the launching of keywords. Therefore, we select some keywords related to the main goal of these SLRs to perform this search. These keywords are combined with logical operators to conform a query string (QS).

It worth mentioning that this QS must be adapted to the particular syntax of the digital library to be interpreted. Below, we present the keywords and the QSs defined for our SLRs.

#### *SLR 1 - Approaches Centred in CMS-based Web Applications*

For the SLR related to the approaches that are centred in the development and migration of CMS-based Web applications, we have selected the following keywords: *Web Content Management, Content Management System, CMS, Web Engineering, Web application, method, methodology, frameworks, development, migration, modernization, generation.*

As we can see below, the QS used is combined with a set of OR and AND operators.

- a) "Web Content Management" OR "Content Management System" OR CMS.
- b) "Web Engineering" OR "Web application".
- c) method OR methodology OR frameworks.
- d) development OR migration OR modernization OR generation.

$$\text{QS} = (\text{a}) \text{ AND } (\text{b}) \text{ AND } (\text{c}) \text{ AND } (\text{d})$$

#### *SLR 2 – Model-driven Reengineering Methods*

The keywords selected for the SLR focused on the model-driven reengineering methods are: *modernization, migration, reverse engineering, reengineering, architecture driven, model driven, mda-based, mde-base and mdd-based.*

The resulting QS used is also combined with a set of OR and AND operators.

- a) modernization OR migration OR “reverse engineering” OR reengineering
- b) “architecture driven” OR “model driven” OR “mda-based” OR “mde-based” OR “mdd-based”

$$QS = (\mathbf{a}) \text{ AND } (\mathbf{b})$$

#### ***Inclusion and Exclusion Criteria Definition***

Although the QS has been defined accurately, we consider the possibility that a certain number of studies obtained from the searches do not provide any evidence related to the research questions posed. In order to filter those studies, it is necessary to define some inclusion and exclusion criteria based on these research questions. In the following we present the inclusion and exclusion criteria defined for each SLR.

##### *SLR 1 - Approaches Centred in CMS-based Web Applications*

In this section we present the inclusion criteria defined for the first SLR are. Firstly we apply the inclusion criteria and then the exclusion criteria. The inclusion criteria are that:

- One of the following words: “Web content management”, “content management system” or CMS must appear at least in the title, abstract or keywords of the study.
- The abstract must let us to conclude that the main purpose of the study is a methodological proposal focused on the engineering of CMS-based Web applications.

Studies that fulfill the inclusion criteria exist but they maybe do not provide relevant information for our research. Therefore, we have to read in detail each study keeping in mind the following exclusion criteria:

- Studies using CMS-based Web applications for supporting or implementing a concrete case study (not proposing a methodological approach).
- Studies focused on the management of content not designed to be available through the Web.
- Duplicate reports of the same study. Only the most completed version of the study will be included.

- It is worth noting that studies have not been excluded on the basis of their publication date.

#### *SLR 2 – Model-driven Reengineering Methods*

As for the second SLR, we have defined a set of inclusion criteria and exclusion criteria that are presented below. The inclusion criteria are that:

- The keywords defined for the QS must appear at least in the title, abstract or keywords of the study.
- The abstract must let us to conclude that the main purpose of the study is a methodological proposal focused on a model-driven reengineering method.
- The studies must be written in English, Spanish and Portuguese.

On the other hand, the exclusion criteria defined for this SLR are listed in the following:

- Duplicate reports of the same study. Only the most completed version of the study will be included.
- Studies focused on the reengineering without applying the model-driven principles.

#### *Data Extraction Definition*

As the last task in the *Planning* phase, we define a set of criteria as a mechanism for the data extraction from the selected studies. This data extraction allows us to compare the different approaches found in our SLR and to reach to final conclusions. In next subsections, we present the criteria for the data extraction for each SLR.

#### *SLR 1 - Approaches Centred in CMS-based Web Applications*

For the data extraction definition, we define a set of seven criteria to extract data from the studies found in this SLR. These criteria are presented below.

- The **type of process** allows us to identify if the process defined by the approach is a development process or a reengineering process. The development process is characterized by starting with the definition of the IS at a high abstraction level and finishing with the generation of code at a low level. The reengineering process is defined as a horseshoe process which starts extracting knowledge from a legacy IS and finishes generating code for a target IS.

- The **Web Engineering views** allows us to know how complete is the method considering the coverage of the Web Engineering views. These views according to (Kappel, Pröll, et al., 2006) (Ginige & Murugesan, 2001a) (Ginige & Murugesan, 2001b) (Murugesan et al., 2001) (Ginige, 2002) (Deshpande & Hansen, 2001) (Deshpande et al., 2002) are: 1) **content view**, is the view used to represent the business and data objects; 2) **navigation view**, is the view which expresses the composition of the interface in terms of container and the navigation map of the Web application; 3) **process view**, is the view that defines how the application reacts to the events raised by the user's navigation and 4) **presentation view**, is the view for specifying the layout and the look & feel of the interface. Furthermore, to complete this analysis we have considered two views belonging to the traditional Software Engineering (Pressman, 1996), (Ian Sommerville, 2002): the **requirements view** which is the view that reflects the functionality that Web applications offer to users and the **implementation view** which expresses Web applications in terms of artefacts implemented by code.
- The **models considered** allows us to know whether the approach uses models as a key artefact to carry out the development or reengineering process. With this criterion we can know if it meets with the MDE principles.
- The **MDA abstraction levels** allows us to know if the models used in the development/reengineering process are defined at some of the abstraction levels proposed by related by MDA (CIM, PIM, PSM). Thus, we can know if the method meets with one of the MDA principles.
- The **modelling languages** is related to the modelling language used to define the models. We consider interesting to know if the approach is supported by a standard modelling language such as UML or, otherwise, the approach uses its own DSL.
- The **automation level** refers to the level of automation of the approach. It allows us to know which type of automated transformation (T2M, M2M, M2T) are considered by the approach. If the approach automates all the development/reengineering process, it has a high level of automation. If the approach just automates some of the steps of the process, we consider it with a medium level of automation. Finally, if the approach does not consider any type of automated transformation it is considered to have a low level of automation.

- The **supporting toolkit** this criterion refers to the toolkit supporting the methodological approach. This toolkit can automate the T2M, M2M and M2T transformations as well as to assist the definition of the models considered within the approach.

#### *SLR 2 – Model-driven Reengineering Methods*

For the data extraction definition of this second SLR, we define seven criteria to extract data from the studies found in this SLR. These criteria are presented below.

- The **reengineering stage** analyzes which of the established reengineering stages are covered by the proposal: 1) *reverse engineering stage*, 2) *restructuring stage* or 3) *forward engineering stage*.
- The **scope of the approach** analyzes the aspect in which the approach is focused, so that it is possible to find approaches focused on: 1) the reengineering of data bases, 2) reengineering of Web applications, 3) the reengineering of Web services or 4) legacy systems (general purpose).
- The **ADM abstraction levels** checks the abstraction level at which the models are defined by the approach: 1) CIM level, 2) PIM level or 3) PSM level.
- The **metamodels considered** allows to know which metamodels the models defined by the approach conforms to. Accordingly, the approaches can: 1) define their own metamodel to represent the models at any abstraction level, 2) use existing standard metamodels and 4) use the ADM standard metamodels (ASTM, KDM or SMM).
- The **automation level** allows to analyze whether the transformations considered by the approach to systemize the reengineering process are implemented. If the approach implements all its automated transformations, we consider it with a high level of automation. If the approach implements some of its transformations we consider it with a medium level of automation. Finally, if the approach does not implement any type of transformation we consider it with a low level of automation.
- The **implementation of transformations** allows to know the technologies and techniques used to implement the automated transformations considered in the approach. In the case of the T2M transformations the techniques might be: 1) parser, 2) existing tool in the market or 3) implementation of

transformation rules. As for the M2M and M2T transformations, the technologies are related to the transformation languages used to implement the transformation rules

- The **kind of code to migrate** is related to the source code implementing the legacy system from which the approach extracts the models and starts the reengineering process.
- The **supporting toolkit** is about the availability of tools supporting the tasks defined in each reengineering approach, such as graphical editors for creating models or frameworks allowing to run the automated transformations.

## Execution Phase

The *Execution* phase of the SLR is basically the execution of the search process and selection of primary studies that help address the main goal of this research. Therefore, in this section the results obtained in the execution of searches (section 0) and the list of selected primary studies (section 0) are presented.

### *Search Process*

In this task, we adapt the QS to the syntax of the digital library and we launch them to obtain the first set of studies. With the objective of maximizing the results, we decided to run the searches in the widest scope allowed by the digital library. In the following, we present the execution of this task in both SLRs.

#### *SLR 1 - Approaches Centred in CMS-based Web Applications*

Table 10-1 shows the QSs for the SLR1 adapted to the digital libraries selected. The first column is the acronym to identify the digital library, the second column is the QS adapted, and the third column is the widest scope allowed by the digital library to launch the QS.

**Table 10-1. QS adapted (for the SLR1) to the syntax of the digital library.**

DIGITAL LIBRARY	ADAPTED QUERY STRING	SCOPE
ACM	("web Content Management" or "Content Management System" or CMS) and ("Web Engineering" or "Web application") and (method or methodology or frameworks) and (development or migration or modernization or generation)	All
CSX	((("web Content Management" OR "Content Management System" OR CMS) AND ("Web Engineering" OR "Web application")) AND (method OR methodology OR frameworks) AND (development OR migration OR modernization OR generation))	Text
IEEEEX	((("web Content Management" OR "Content Management	All

DIGITAL LIBRARY	ADAPTED QUERY STRING	SCOPE
	System" OR CMS) AND ("Web Engineering" OR "Web application") AND (method OR methodology OR frameworks) AND (development OR migration OR modernization OR generation))	
ISI	TS = ("web Content Management" OR "Content Management System" OR CMS) AND TS = ("Web Engineering" OR "Web application") AND TS = (method OR methodology OR frameworks) AND TS = (development OR migration OR modernization OR generation)	Topic
SD	ALL (("web Content Management" OR "Content Management System" OR CMS) AND ("Web Engineering" OR "Web application") AND (method OR methodology OR frameworks) AND (development OR migration OR modernization OR generation))	All
SL	(("web Content Management" OR "Content Management System" OR CMS) AND ("Web Engineering" OR "Web application") AND (method OR methodology OR frameworks) AND (development OR migration OR modernization OR generation))	All
GS	(("web Content Management" OR "Content Management System" OR CMS) AND ("Web Engineering" OR "Web application") AND (method OR methodology OR frameworks) AND (development OR migration OR modernization OR generation))	All

In the launch of the QSSs, we found 2,484 studies. To these studies, we applied the inclusion criteria defined in Section 0 to filter them and obtain the relevant studies. We obtained 89 relevant studies out of the 2,484 total studies (3.58%). We determined that most of the studies belonged to other research areas different from Software Engineering or Web Engineering. Analyzing the results obtained from the digital libraries, we can say that the two most precise digital libraries are SCP and IEEEX. On the one hand, the 1.21% of the relevant studies were found with SCP. On the other hand, the 0.85% of the relevant studies were found with IEEEX.

Afterwards, to accurate the SLR we applied the exclusion criteria to obtain the primary studies of our research. Therefore, we read entirely each study and we rejected those which only used CMS-based Web applications for supporting or implementing a concrete case study (not proposing a methodological approach) as well as those which were focused on the management of content not designed to be available through the Web. As a result, we obtained the list of 15 primary studies (17%). The two digital libraries from which we obtained more primary studies are SCP with the 4.49% of the relevant studies and GS with the 3.37%. Table 10-2 shows the numerical information obtained from this research process.

**Table 10-2. Result of the search process of the SLR1**

DIGITAL LIBRARY (DL)	SEARCH RESULTS	RELEVANT STUDIES	PRIMARY STUDIES	% OF RELEVANT STUDIES	% OF PRIMARY STUDIES
ACM	187	10	2	0,40 %	2,25 %
IEEEEX	386	21	2	0,85 %	2,25 %
ISI	10	6	2	0,24 %	2,25 %
SD	259	1	1	0,04 %	1,12 %
SL	443	6	1	0,24 %	1,12 %
SCP	199	30	4	1,21 %	4,49 %
GS	1000	15	3	0,60 %	3,37 %
<b>ALL LIBRARIES</b>	<b>2,484</b>	<b>89</b>	<b>15</b>	<b>3,58 %</b>	<b>17 %</b>

*SLR 2 – Model-driven Reengineering Methods*

In this subsection, we show the search process perform in the SLR2. Table 10-3 shows the QSs for the SLR2 adapted to the digital libraries selected.

**Table 10-3. QS adapted (for the SLR2) to the syntax of the digital library**

DIGITAL LIBRARY	ADAPTED QUERY STRING	SCOPE
ACM	((modernization or migration or “reverse engineering” or reengineering) and (“architecture driven” or “model driven” or “mda-based” or “mde-based” or “mdd-based”))	All
CSX	((modernization OR migration OR “reverse engineering” OR reengineering) AND (“architecture driven” OR “model driven” OR “mda-based” OR “mde-based” OR “mdd-based”))	Text
IEEEEX	((modernization OR migration OR “reverse engineering” OR reengineering) AND (“architecture driven” OR “model driven” OR “mda-based” OR “mde-based” OR “mdd-based”))	All
ISI	TS = (modernization OR migration OR “reverse engineering” OR reengineering) AND TS = (“architecture driven” OR “model driven” OR “mda-based” OR “mde-based” OR “mdd-based”))	Topic
SD	ALL ((modernization OR migration OR “reverse engineering” OR reengineering) AND (“architecture driven” OR “model driven” OR “mda-based” OR “mde-based” OR “mdd-based”))	All
SL	((modernization OR migration OR “reverse engineering” OR reengineering) AND (“architecture driven” OR “model driven” OR “mda-based” OR “mde-based” OR “mdd-based”))	All
GS	((modernization OR migration OR “reverse engineering” OR reengineering) AND (“architecture driven” OR “model driven” OR “mda-based” OR “mde-based” OR “mdd-based”))	All

Launching the previous QSs, we found 6,130 studies. To these studies, we applied the inclusion criteria defined in Section 0. We obtained 175 relevant studies out of the 6,130 total studies (2.85 %). Analyzing the results obtained from the digital libraries, we can say that the two most precise digital libraries are SCP and GS. The 0.86% of the relevant studies were found with SCP and the 0.65% with GS.

Then, to accurate the SLR we applied the exclusion criteria to obtain the primary studies of our research. As a result, we obtained 27 primary studies (15%). The two digital libraries from which we obtained more primary studies are GS with the 5.14% and SCP (along with SL) with the 2.86%. Table 10-4 shows the numerical information obtained from this research process.

**Table 10-4. Result of the search process of the SLR2**

DIGITAL LIBRARY (DL)	SEARCH RESULTS	RELEVANT STUDIES	PRIMARY STUDIES	% OF RELEVANT STUDIES	% OF PRIMARY STUDIES
ACM	396	10	1	0,16 %	0,57 %
IEEEEX	736	28	3	0,46 %	1,71 %
ISI	3	3	2	0,05 %	1,14 %
SD	730	16	2	0,26 %	1,14 %
SL	364	25	5	0,41 %	2,86 %
SCP	471	53	5	0,86 %	2,86 %
GS	3430	40	9	0,65 %	5,14 %
<b>ALL LIBRARIES</b>	<b>6,130</b>	<b>175</b>	<b>27</b>	<b>2,85 %</b>	<b>15 %</b>

#### *Primary Studies Selection*

Once executed the research process, we have selected the primary studies that contribute to our research. We present these primary studies considering the two SLRs performed.

#### *SLR 1 - Approaches Centred in CMS-based Web Applications*

Table 10-5 shows the primary studies found in the SLR1. These studies have been grouped in approaches according into their commonalities in terms of authors and proposed ideas. Accordingly, we define four different approaches identified with an ID (APPRNum), as it is shown in the first column. It is worth noting that APPR1 and APPR2, in spite of belonging to the same research group, have been considered as different approaches because of their different nature.

**Table 10-5. Primary studies obtained in SLR1**

ID	AUTHORS	TITLE
APPR1	J. Souer, T. Urlings, R. Helms, S. Brinkkemper	Engineering Web information systems: A content management system-based approach (Jurriaan Souer et al. 2011).
	L. Luinenburg, S. Jansen and J. Souer, I. Van De Weerd and S. Brinkkemper	Designing Web Content Management Systems Using the Method Association Approach (Luinenburg et al. 2008).
	J. Souer, L. Luinenburg, J. M. Versendaal, I. van de Weerd and S. Brinkkemper	Engineering a design method for web content management implementations (Jurriaan Souer, Luinenburg, et al. 2008).
	J.Souer, P. Honders, J.M. Versendaal and S.Brinkkemper	A Framework for Web Content Management System Operations and Maintenance (Jurriaan Souer,

<b>ID</b>	<b>AUTHORS</b>	<b>TITLE</b>
APPR2	J.Souer, P. Honders, J.M. Versendaal and S. Brinkkemper	Honders, et al. 2008). Defining operations and maintenance in web engineering: A framework for CMS-based Web applications (Jurriaan Souer, Honders, et al. 2007).
	I. van de Weerd, S. Brinkkemper, J. Souer, J. M. Versendaal	A situational implementation method for web-based content management system-applications: method engineering and validation in practice (I. V. D. Weerd & Sjaak Brinkkemper 2006).
	J. Souer, I. van de Weerd, J. M. Versendaal and S. Brinkkemper	Situational requirements engineering for the development of content management system-based Web Application (J Souer, I. van de Weerd, et al. 2007).
	J. Souer, T. Kupers, R. Helms and S. Brinkkemper	A model-driven Web Engineering for the automated configuration of Web content management systems (J Souer et al. 2009).
	J. Souer and T. Kupers	Towards a Pragmatic Model Driven Engineering Approach for the Development of CMS-based Web Applications (Jurriaan Souer & Thijs Kupers 2009).
	J.D.S. Saraiva and A.R. da Silva	Web-Application Modelling With the CMS-ML Language (de Sousa & Rodrigues 2010).
APPR3	J.D.S. Saraiva and A. R. da Silva	Development of CMS-Based Web-Applications Using a Model-Driven Approach (J. D. S. Saraiva & A. R. D. Silva 2009).
	J.D.S. Saraiva and A.R. da Silva	CMS-Based Web-Application Development Using Model-Driven Languages (de Sousa & Rodrigues 2009).
	J.D.S. Saraiva and A.R. da Silva	The WebComfort Framework: An Extensible Platform for the Development of Web Applications (J. d. S. Saraiva & A. R. . Silva 2008)
APPR4	K. Vlaanderen, F. Valverde and O. Pastor	Model-driven Web Engineering in the CMS domain: A preliminary research applying SME (K Vlaanderen et al. 2009).
	K. Vlaanderen, F. Valverde and O. Pastor	Improvement of a Web Engineering method applying situational method engineering (Kevin Vlaanderen 2008).

#### *SLR 2 – Model-driven Reengineering Methods*

Table 10-6 presents the primary studies found in the SLR2. These studies have been grouped in 9 approaches considering their communalities, as we did in the SLR1.

**Table 10-6. Primary studies obtained in SLR2**

<b>ID</b>	<b>AUTHORS</b>	<b>TITLE</b>
APPR1	Garcia-Rodriguez de Guzman, I.Polo, M., Piattini, M.	An ADM Approach to Reengineer Relational Databases towards Web Services (Garcia-Rodriguez de Guzman et al., 2007)
	Garcia-Rodriguez de Guzman, I.	PRESSWEB : A Process to Reengineer Legacy Systems towards Web Services (García-Rodríguez de Guzmán, 2007)
	García-Rodríguez de Guzmán, I.	A methodology for database reengineering to Web Services (García-Rodríguez de Guzmán, Polo, &

<b>ID</b>	<b>AUTHORS</b>	<b>TITLE</b>
	Polo, Macario, Piattini, Mario	Piattini, 2006)
	Perez-Castillo, Ricardo, García-Rodríguez de Guzmán, I.	Obtaining web services from relational database (Garcia-Rodriguez de Guzman et al., 2006b)
APPR2	Blanco, CarlosPérez-castillo, RicardoHernández, Arnulfo Fernández-Medina, Eduardo	Towards a Modernization Process for Secure Data Warehouses (Blanco et al., 2009)
APPR3	Pérez-Castillo, RicardoGarcía-Rodríguez de Guzman, Ignacio, Avila-Garcia, Orlando, Piattini, Mario	On the use of ADM to Contextualize Data on Legacy Source Code for Software Modernization (Perez-Castillo, Garcia-Rodriguez de Guzman, Avila-Garcia, et al., 2009)
	Pérez-Castillo, RicardoGarcía-Rodríguez de Guzmán, Ignacio, Piattini, Mario	Business process archeology using MARBLE (Pérez-Castillo, García-Rodríguez de Guzmán, & Piattini, 2011)
	Pérez-castillo, RicardoWeber, Barbara, García-Rodríguez de Guzmán, Ignacio, Piattini, Mario	Modernizing Legacy Systems through Runtime Models (Pérez-castillo, Weber, García-Rodríguez de Guzmán, & Piattini, 2010)
	Pérez-Castillo, RicardoGarcía-Rodríguez de Guzmán, Ignacio, Piattini, Mario, Weber, Barbara	Integrating event logs into KDM repositories (Pérez-Castillo, García-Rodríguez de Guzmán, Piattini, et al., 2012)
	Pérez-Castillo, Ricardo,García-Rodríguez de Guzmán, Ignacio, Piattini, Mario, Weber, Barbara., Places, Ángeles S.	An empirical comparison of static and dynamic business process mining (Pérez-Castillo, García-Rodríguez de Guzmán, Piattini, et al., 2011)
	Pérez-Castillo, RicardoCruz-Lemus, José a., Garcia-Rodriguez de Guzman, Ignacio, Piattini, Mario	A family of case studies on business process mining using MARBLE (Pérez-Castillo, Cruz-Lemus, Garcia-Rodriguez de Guzman, & Piattini, 2012)
	Pérez-castillo, Ricardo, Fernández-ropero, María, Garcia-Rodriguez de Guzman, Ignacio, Piattini, Mario	MARBLE. A business process archeology tool (Pérez-castillo, Fernández-ropero, Garcia-Rodriguez de Guzman, & Piattini, 2011)
	Pérez-castillo, Ricardo, García-Rodriguez de Guzman, Ignacio, Piattini, Mario	Implementing Business Process Recovery patterns through QVT transformations (Pérez-castillo, García-Rodriguez de Guzman, et al., 2010)
	Pérez-Castillo, R. Weber, B., García-Rodriguez de Guzman, Ignacio, Piattini, M.	Process mining through dynamic analysis for modernizing legacy systems (Pérez-Castillo et al., 2011)
APPR4	Perez-Castillo, Ricardo, García-Rodríguez de Guzmán, Ignacio	PRECISO: A Reengineering Process and a Tool for Database Modernisation through Web Services (Perez-Castillo & García-Rodríguez de Guzmán, 2009)
	Perez-Castillo, Ricardo, García-Rodriguez de Guzman, Ignacio, Caballero, Ismael, Polo, Macario, Piattini, Mario	PRECISO: A Reverse Engineering Tool to Discover Web Services from Relational Databases (Perez-Castillo, Garcia-Rodriguez de Guzman, Caballero, et al., 2009)
APPR5	Pérez-Castillo, Ricardo, García-Rodríguez de Guzmán, Ignacio, Caivano, Danilo, Piattini, Mario.	Database Schema Elicitation to Modernize Relational Databases (Pérez-Castillo, García-Rodríguez de Guzmán, Caivano, et al., 2012)
APPR6	Rodríguez-Echeverría, Roberto, Conejero, M, Clemente, Pedro J, Villalobos, D., Fernando, S.	Generation of WebML Hypertext Models from Legacy Web Applications (Rodríguez-Echeverría, Conejero, Clemente, et al., 2012)
	Rodríguez-Echeverría, Roberto, Clemente, Pedro J, Preciado, Juan C, Fernando, S	Modernization of Legacy Web Applications into Rich Internet Applications (Rodríguez-Echeverría et al., 2011)

<i>ID</i>	<i>AUTHORS</i>	<i>TITLE</i>
	Rodríguez-Echeverría, Roberto, Conejero, José María, Linaje, Marino, Preciado, Juan Carlos, Sánchez-Figuerola, Fernando	Re-engineering Legacy Web Applications into Rich Internet Applications (Rodríguez-echeverría et al., 2010)
	Roberto Rodríguez-Echeverría, Jose M. Conejero, Pedro J. Clemente, Víctor, M. Pavón, and Fernando Sanchez-Figuerola	Model Driven Extraction of the Navigational Concern of Legacy Web Applications (Rodríguez-Echeverría, Conejero, Pedro J. Clemente, et al., 2012)
	Roberto Rodríguez-Echeverría, Clemente, Pedro J, Villalobos, Dolores, Fernando, S.	Extracting Navigational Models from Struts-Based Web Applications (Rodríguez-Echevarría et al., 2012)
APPR7	Van Hoorn, Andre, Sören, Frey Goerigk, Wolfgang Hasselbring, Wilhelm Knoche, Holger Köster, Sönke Krause, Harald Porembski, Marcus Stahl, Thomas Steinkamp, Marcus Wittmüss, Norman	DynaMod Project: Dynamic Analysis for Model-Driven Software Modernization (Van Hoorn et al., 2011)
APPR8	Sadovykh, AndreyVigier, Lionel Hoffmann, Andreas Grossmann, Juergen Ritter, Tom Gomez, Eduardo Estekhin, Oleg	Architecture Driven Modernization in Practice-Study Results (Sadovykh, Vigier, Hoffmann, et al., 2009b)
	Sadovykh, Andrey, Vigier, Lionel, Gomez, Eduardo, Hoffmann, Andreas	On Study Results : Round Trip Engineering of Space Systems (Sadovykh, Vigier, Gomez, et al., 2009)
APPR9	Normantas, Kestutis, Sosunovas, Sergejus, Vasilecas, Olegas	An Overview of the Knowledge Discovery Meta-Model (Normantas, Sergejus, & Vasilecas, 2012)
	Vasilecas, Olegas, Normantas, Kestutis	Deriving Business Rules from the Models of Existing Information Systems (Vasilecas & Normantas, 2011b)

*Appendix C : Specification of  
the PHP Metamodel*

---

---



In this Appendix, we present the rest of the specification of the PHP metamodel explained in Chapter 5. To not overextend this Appendix, we explain just one element of each table presented.

### PHP Expressions

In this section, we present the rest of **PHP Expressions** defined and represented in the PHP metamodel.

#### Object Access Expression

In this section, we present the representation of the *object/class access expressions* into the PHP metamodel. This mapping is shown in Table 11-1.

**Table 11-1. Mappings from other PHP expressions to PHP elements.**

<i>PHP CODE</i>		<i>PHP METAMODEL</i>		<i>CONDITION</i>
<i>Element</i>	<i>Element</i>	<i>Properties</i>		
Object/Class access expression	ObjectAccess	Left, obj, right		No conditions

- *Mapping from the element Object/Class Access Expression to the element ObjectAccess:* The *Object Access Expression* is mapped to the PHP element *ObjectAccess*. It has three attributes *left*, *obj* and *right*. The attribute *left* stores the object or class being accessed. The attribute *obj* stores the access operator. This access operator can be defined as an object access ( $\rightarrow$ ) or as a class access ( $::$ ). Finally, the attribute *right* defines the variable, the function or another class or object to be accessed.

#### Unary Expressions

In Table 11-2, we present the mappings for the rest of *unary expressions* to the PHP metamodel.

**Table 11-2. Mappings from Unary Expressions to PHP elements.**

<i>PHP CODE</i>		<i>PHP METAMODEL</i>		<i>CONDITION</i>
<i>Element</i>	<i>Element</i>	<i>Properties</i>		
Unary expression	NewOrClone	tag right		If the unary operator is Class New, clone
	Increment	operator right left		If the unary operator is Increment/Decrement++, --

- **Mapping from the element Unary Expression to the element NewOrClone:** If the *Unary Expression* is defined with a unary operator of the type *class* it is mapped to the PHP element *NewOrClone*. This element is composed of two attributes, *tag* and *right*. The attribute *tag* stores one of the operators *new* or *clone*, whereas the attribute *right* stores the expression assessed by these operators.

### Binary Expressions

In Table 11-3, we present the mappings for the rest of *binary expressions* to the PHP metamodel. The rest of the mappings are very similar, so that we have explained one of them (*Multiplication*). Most of the PHP elements generated are composed of the attributes *left*, and *right*, some of them define the attribute *operator* also. In the following, Table 11-3 presents the rest of mappings from *binary expressions*.

**Table 11-3. Mappings from binary expressions to PHP elements.**

<i>PHP CODE</i>		<i>PHP METAMODEL</i>		<i>CONDITION</i>
<i>Element</i>	<i>Element</i>	<i>Properties</i>		
<b>Binary expression</b>	<b>Multiplication</b>	left operator right		If the binary operator is one of these three arithmetic operators(*, /, %)
	<b>ComparisonCheck</b>	left operator right		If the binary operator is one of these comparison operators (<, <=, >, >=)
	<b>WeakLogicalAnd</b>	left right		If the binary operator is one of these two logical operators (and)
	<b>WeakLogicalOr</b>	left right		If the binary operator is one of these two logical operators (or)
	<b>LogicalOr</b>	left right		If the binary operator is one of these two logical operators (  )
	<b>WeakLogicalXor</b>	left right		If the binary operator is a xor operator
	<b>BitWiseAnd</b>	left right		If the binary operator is this bitwise operator &
	<b>BitWiseOr</b>	left right		If the binary operator is this bitwise operator
	<b>BitWiseXor</b>	left right		If the binary operator is this bitwise operator ^

- **Mapping from the element Binary Expression to the element Multiplication:** As we can see in Table 11-3, the *Binary Expression* defined with a binary operator of type *multiplication* is mapped to the PHP element *Multiplication*. This element is composed of three attributes, *operator*, *right* and *left*. The attribute *operator* stores the operator of the binary expression,

the attribute *left* stores the left operand and the attribute *right* stores the right operand of the binary expression.

## PHP Statements

In this section, we present the rest of **PHP Statements** defined and represented in the PHP metamodel.

### Definition Statements

In Table 11-4, we present the mapping from the class definition to the PHP metamodel.

**Table 11-4. Mappings from definition statements to PHP elements.**

<b>PHP ELEMENT</b>	<b>PHP METAMODEL</b>		<b>CONDITION</b>
	<i>Element</i>	<i>Properties</i>	
Class definition	ClassDefStatement	className extName members	No conditions

- *Mapping from the element Class Definition to the element ClassDefStatement:* The *Class Definition* is mapped to the element *ClassDefStatement* in the PHP metamodel. It is composed of three attributes *className*, *extName* and *members*. The attribute *className* defines the name of the class, whereas the *extName* stores the name of a parent class in case it is an extension. Finally, the attribute *members* stores all the definitions (*variable definitions* and *function definitions*) specified within the class.

### Expression Statements

In Table 11-5, we present the mapping from the *object access statement* to the PHP metamodel.

**Table 11-5. Mappings from Expression Statements to PHP elements.**

<b>PHP ELEMENT</b>	<b>PHP METAMODEL</b>		<b>CONDITION</b>
	<i>Element</i>	<i>Properties</i>	
Object Access Statement	ObjectAccessStatement	obj right left	No conditions
	Member	obj right left	No conditions

- *Mapping from the element Object Access Statement to the elements ObjectAccessStatement and Member:* The *Object Access Statement* is mapped to two elements in the PHP metamodel, the *ObjectAccessStatement*

and *Member* elements. The former is composed of three attributes *left*, *right* and *obj* as occurred with the *ObjectAccess* element. The attribute *left* stores the object or class accessed. The attribute *obj* stores the access operator. This operator can be defined as an object access ( $\rightarrow$ ) or as a class access (::). Finally, the attribute *right* defines the member accessed. This member is mapped to the element *Member*, composed also by the three attributes *left*, *right* and *obj*.

### Control Statements

In Table 11-6, we present the mappings from the *control statements* to the PHP metamodel.

**Table 11-6. Mappings from control statements to PHP elements.**

<i>PHP ELEMENT (SOURCE)</i>		<i>PHP METAMODEL (TARGET)</i>	
<i>Element</i>	<i>Element</i>	<i>Properties</i>	<i>CONDITION</i>
<b>Switch Statement</b>	<b>SwitchStatement</b>	condition case defaultStatement	No conditions
	<b>Case</b>	label statements	No conditions
<b>Foreach Statement</b>	<b>ForeachStatement</b>	expressio_array value bodyStatement	No conditions
<b>While Statement</b>	<b>WhileStatement</b>	condition statements	No conditions

- **Mapping from the element Switch Statement to the elements SwitchStatement and Case:** The *Switch Statements* are mapped to two PHP elements, the *SwitchStatement* element and the *Case* element. The *SwitchStatement* element is defined with three attributes, *condition*, *case* and *defaultStatement*. The condition of the switch statement is stored in the attribute *condition*, whereas the different entries (alternatives) of the switch statement within the attribute *case*. Finally, the default entry is defined in the attribute *defaultStatement*. The entries are mapped to the element *Case* which is defined by two attributes, *label* and *statements*. The former represents the value represented by a concrete entry and the latter the body composed of statements.

*Appendix D: Concrete Syntax  
of the CMS DSL*

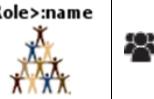
---

---



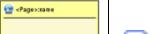
In Table 12-1, we present the GMF annotations, the EOL definition and the graphical notation of the elements of the CMS Common Metamodel belonging to the user concern.

**Table 12-1. Graphical notation of the elements of the user concern**

CMS	GMF ANNOTATION	EOL DEFINITION	GRAPHICAL
Role	@gmf.node(figure="figures.RoleFigure", label="name", label.pattern=" <role>:{0}")</role>	Label formatting var diagramLabel = GmfGraph!DiagramLabel.all.selectOne(l l.name='RoleLabel'); diagramLabel.elementIcon=false;	<Role>:name 
permission	@gmf.link(source="permissionsRole", target="permissionsFun", label="setting")	Colour var colour = new GmfGraph!RGBColour; colour.red = 102; colour.green = 0; colour.blue = 51; permissions.foregroundColour = colour;	ENABLED/DISABLED/INHERITED 
User	@gmf.node(figure="figures.UserFigure", label="name", label.pattern=" <user>:{0}")</user>	Label formatting var userLabel = GmfGraph!Label.all.selectOne(l l.name = 'UserLabelFigure'); userLabel.font = new GmfGraph!BasicFont; userLabel.font.style = GmfGraph!FontStyle#BOLD; userLabel.font.height=9;;	<User>:name 
User Role	@gmf.link (label="<role>")	Colour var colour = new GmfGraph!RGBColour; colour.red = 0; colour.green = 51; colour.blue = 51; userRole.foregroundColour = colour;	<role> 
Role Group	@gmf.node(figure=rounded, label="name", label.pattern=" <rolegroup>:{0}")</rolegroup>	Background colour var colour = new GmfGraph!RGBColour; colour.red =153; colour.green = 255; colour.blue = 153; roleGroup.backgroundColour = colour;	<RoleGroup>:name 

In Table 12-2, we present the GMF annotations, the EOL definition and the graphical notation of the elements of the CMS Common Metamodel belonging to the navigation concern.

**Table 12-2. Graphical notation of the elements of the navigation concern**

CMS	GMF ANNOTATION		EOL DEFINITION	GRAPHICAL
Menu	@gmf.node(figure=rectangle, label="name", label.pattern="<Menu>:{0}")	Background colour	var colour = new GmfGraph!RGBColour; colour.red = 224; colour.green = 224; colour.blue = 224; menuFigure.backgroundColour = colour;	 
menuItem	@gmf.node(figure=rectangle, label="Title,position", label.pattern="<MenuItem>:{0}Weight:{1}")	Background colour	var colour = new GmfGraph!RGBColour; colour.red = 51; colour.green = 153; colour.blue = 255; menuItemFigure.backgroundColour = colour;	 
menu_name	@gmf.link (label="<from-menu>")	Colour	var colour = new GmfGraph!RGBColour; colour.red = 0; colour.green = 204; colour.blue = 0; menu_name.foregroundColour = colour;	 
parent MenuItem	@gmf.link (label="<menuItem-parent>")	Colour	var colour = new GmfGraph!RGBColour; colour.red = 100; colour.green = 149; colour.blue = 237; parentMenuItem.foregroundColour = colour;	 
Page	@gmf.node(figure=rectangle, label="title", label.pattern="<Page>:{0}")	Background colour	var colour = new GmfGraph!RGBColour; colour.red = 255; colour.green = 255; colour.blue = 153; pageFigure.backgroundColour = colour;	 
Page Category	@gmf.link (label="<category>")	Colour	var colour = new GmfGraph!RGBColour; colour.red = 171; colour.green = 130; colour.blue = 255; pageCategory.foregroundColour = colour;	 

In Table 12-3, we present the GMF annotations, the EOL definition and the graphical notation of the elements of the CMS Common Metamodel belonging to the content concern.

**Table 12-3. Graphical notation of the elements of the content concern**

CMS	GMF ANNOTATION		EOL DEFINITION	GRAPHICAL
MediaGallery Content	@gmf.node(figure="figures.MediaGalleryContentFigure", label="name", label.pattern="<MediaGalleryContent>")	Label formatting	contentLabel4.font = new GmfGraph!BasicFont; contentLabel4.font.style = GmfGraph!FontStyle#BOLD; contentLabel4.font.height=9;	 

CMS	GMF ANNOTATION	EOL DEFINITION	GRAPHICAL
ListContent	@gmf.node(figure="figures.ListContentFigure", label="name", label.pattern="<ListContent>")	Label formatting var contentLabel3 = GmfGraph!Label.all.selectOne(l l.name = 'ListContentLabelFigure'); contentLabel3.font = new GmfGraph!BasicFont; contentLabel3.font.style = GmfGraph!FontStyle#BOLD; contentLabel3.font.height=9;	<ListContent> 
MediaContent	@gmf.node(figure="figures.MediaContentFigure", label="name", label.pattern="<MediaContent>")	Label formatting var contentLabel2 = GmfGraph!Label.all.selectOne(l l.name = 'MediaContentLabelFigure'); contentLabel2.font = new GmfGraph!BasicFont; contentLabel2.font.style = GmfGraph!FontStyle#BOLD; contentLabel2.font.height=9;	<MediaContent> 
TextContent	@gmf.node(figure="figures.TextContentFigure", label="name", label.pattern="<TextContent>")	Label formatting var contentLabel = GmfGraph!Label.all.selectOne(l l.name = 'TextContentLabelFigure'); contentLabel.font = new GmfGraph!BasicFont; contentLabel.font.style = GmfGraph!FontStyle#BOLD; contentLabel.font.height=9;	<TextContent> 
Category	@gmf.node(figure="figures.CategoryFigure", label="name", label.pattern="<Category>:{0}")	Label formatting var categoryLabel = GmfGraph!Label.all.selectOne(l l.name = 'CategoryLabelFigure'); categoryLabel.font = new GmfGraph!BasicFont; categoryLabel.font.style = GmfGraph!FontStyle#BOLD; categoryLabel.font.height=9;	<Category>:name 
Language	@gmf.node(figure="figures.LanguageFigure", label="name", label.pattern="<Language>:{0}")	Label formatting var languageLabel = GmfGraph!Label.all.selectOne(l l.name = 'LanguageLabelFigure'); languageLabel.font = new GmfGraph!BasicFont; languageLabel.font.style = GmfGraph!FontStyle#BOLD; languageLabel.font.height=9;	<Language>:name 
Comment	@gmf.node(figure="figures.CommentFigure", label="text", label.pattern="<Comment>")	Label formatting var commentLabel = GmfGraph!Label.all.selectOne(l l.name = 'CommentLabelFigure'); commentLabel.font = new GmfGraph!BasicFont; commentLabel.font.style = GmfGraph!FontStyle#BOLD; commentLabel.font.height=9;	<Comment> 
Vocabulary	@gmf.node(figure=rectangle, label="name", label.pattern="<Vocabulary>:{0}")	Background colour var colour = new GmfGraph!RGBColour; colour.red = 255; colour.green = 187; colour.blue = 255; vocabularyFigure.backgroundColour = colour;	<Vocabulary>:name 

CMS	GMF ANNOTATION	EOL DEFINITION	GRAPHICAL
Term	@gmf.node(figure="figures.TermFigure", label="name", label.pattern="<Term>:{0}")	Label formatting var termLabel = GmfGraph!Label.all.selectOne(l l.name = 'TermLabelFigure'); termLabel.font = new GmfGraph!BasicFont; termLabel.font.style = GmfGraph!FontStyle#BOLD; termLabel.font.height=9;	<Term>:name 
ParentTerm	@gmf.link (label="<term-parent>")	Colour var colour = new GmfGraph!RGBColour; colour.red = 255; colour.green = 187; colour.blue = 255; parentTerm.foregroundColour = colour;	
TermContent	@gmf.link (label="<tagged>")	Colour var colour = new GmfGraph!RGBColour; colour.red = 0; colour.green = 0; colour.blue = 0; termContent.foregroundColour= colour;	

In Table 12-4, we present the GMF annotations, the EOL definition and the graphical notation of the elements of the CMS Common Metamodel belonging to the behaviour concern.

Table 12-4. Graphical notation of the elements of the behaviour concern

CMS	GMF ANNOTATION	EOL DEFINITION	GRAPHICAL
Function	@gmf.node(figure="figures.FunctionFigure", label="name", label.pattern="<Function>:{0}")	Label formatting var functionLabel = GmfGraph!Label.all.selectOne(l l.name = 'FunctionLabelFigure'); functionLabel.font = new GmfGraph!BasicFont; functionLabel.font.style = GmfGraph!FontStyle#BOLD; functionLabel.font.height=9;	<function>:name_function 
BlockDecorator	@gmf.node (figure="figures.BlockDecoratorFigure", label="name", label.pattern="<BlockDecorator>:{0}")	Label formatting var blockDecoratorLabel = GmfGraph!Label.all.selectOne(l l.name = 'BlockDecoratorLabelFigure'); blockDecoratorLabel.font = new GmfGraph!BasicFont; blockDecoratorLabel.font.style = GmfGraph!FontStyle#BOLD; blockDecoratorLabel.font.height =9	<Block Decorator>:name_block 
BlockList	@gmf.node(figure="figures.BlockListFigure", label="name", label.pattern="<BlockList>:{0}")	Label formatting var blockListLabelFigure = GmfGraph!Label.all.selectOne(l l.name = 'BlockListLabelFigure'); blockListLabelFigure.font = new GmfGraph!BasicFont; blockListLabelFigure.font.style = GmfGraph!FontStyle#BOLD; blockListLabelFigure.font.height=9;	<Block List>:block_name 

CMS	GMF ANNOTATION	EOL DEFINITION	GRAPHICAL
module	@gmf.node( figure=rectangle, label="name", label.pattern="<Module>:{0}"")	Background colour <pre>var colour = new GmfGraph!RGBColour; colour.red = 247; colour.green = 188; colour.blue = 71; moduleFigure.backgroundColor</pre>	 

In Table 12-5, we present the GMF annotations, the EOL definition and the graphical notation of the elements of the CMS Common Metamodel belonging to the presentation concern.

**Table 12-5. Graphical notation of the elements of the presentation concern**

CMS	GMF ANNOTATION	EOL DEFINITION		GRAPHICAL
Theme	@gmf.node( figure=rectangle, label="name", label.pattern="<Theme>:{0}"")	Background colour	<pre>var colour = new GmfGraph!RGBColour; colour.red = 153; colour.green = 153; colour.blue = 255; themeFigure.backgroundColour = colour;</pre>	 
Region	@gmf.node( figure=rectangle, label="name", label.pattern="<Region>:{0}"")	Style line	<pre>var regionFigure = GmfGraph!Rectangle.all. selectOne(r r.name = 'RegionFigure'); regionFigure.lineKind=GmfGraph!LineKind#LINE_DOT; regionFigure.lineWidth=3;</pre>	 



*Appendix E: Definition of  
Automated Transformations*

---

---



In this Appendix, we extend the specification of the automated transformations of the *ADMigraCMS* method that is explained in Chapter 5. To not overextend this Appendix, we explain just one element of each table presented.

### Transformations Between L0 and L1

#### PHP Expressions

##### *Object Access Expression*

Table 13-1 shows the mapping between the *object acces* element from the PHP metamodel to the ASTM\_PHP elements.

**Table 13-1. Mappings from PHP expressions to ASTM\_PHP elements.**

<i>PHP METAMODEL</i>		<i>ASTM_PHP METAMODEL</i>		<i>CONDITION</i>
<i>Element</i>	<i>Properties</i>	<i>Element</i>	<i>Properties &amp; value</i>	
<b>Object Access</b>	left obj right	<b>Object Access</b> <b>ClassAccess</b>	objectName = left members = right	The <i>Expression</i> mapped depends on the value in the <i>obj</i> attribute.

- **Mapping from the element *ObjectAccess* to the elements *ObjectAccess* or *ClassAccess*:** The element *ObjectAccess* can be mapped to the element *ObjectAccess* or to the element *ClassAccess* depending on the attribute *obj*. If *obj* stores the string →, it is mapped to the element *ObjectAccess*, otherwise if it stores the string ::, it is mapped to the element *ClassAccess*. Both, *ObjectAccess* and *ClassAccess* have two attributes *objectName* and *members*. The former stores the name of the object or the class and the latter contains the member accessed. This member can be a variable or function call. The *objectName* takes the value from the attribute *left* and the attribute *members* from the attribute *right*.

#### *Unary Expressions*

Table 13-2 presents the rest of the mappings defined between the PHP elements that represent *unary expressions* and the ASTM\_PHP elements.

**Table 13-2.** Mappings from unary expressions to ASTM\_PHP elements.

PHP METAMODEL		ASTM_PHP METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
NewOrClone	tag right	Unary Expression	operand = right operator = tag	If the value of <i>tag</i> is <i>clone</i> it is mapped to the <i>Clone</i> element. If the value of <i>tag</i> is <i>new</i> it is mapped to <i>New</i> element.
Increment	operator right left	Unary Expression	operand = right operator = operator	If the value of <i>operator</i> is <i>++</i> it is mapped to <i>Increment</i> element. If the value of <i>operator</i> is <i>--</i> it is mapped to <i>Decrement</i> element.
			operand = left operator = operator	If the value of <i>operator</i> is <i>++</i> it is mapped to <i>PostIncrement</i> element. If the value of <i>operator</i> is <i>--</i> it is mapped to <i>PostDecrement</i> element

- **Mapping from the element NewOrClone to the element UnaryExpression:** The element *NewOrClone* is also mapped to the *UnaryExpression* element. The attribute *operand* of this element takes the value from the attribute *right* of the element *NewOrClone* and the attribute *operator* takes the value of the attribute *tag*. This attribute can be mapped to two different elements depending on its value. If its value is ‘new’ it is mapped to the element *New*, otherwise if it is ‘clone’ is mapped to the element *Clone*.

### Binary Expressions

In the following, we explain the mappings between the PHP elements representing *binary expressions* and the ASTM\_PHP elements. In Table 13-3, we present a set of these mappings.

**Table 13-3.** Mappings from binary expressions to ASTM\_PHP elements.

PHP METAMODEL		ASTM_PHP METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Multiplication	left operator right	Binary Expression	left = left operator = operator right = right	If the value of <i>operator</i> is <i>/</i> is mapped to a <i>Divide</i> element. If the value of <i>operator</i> is <i>*</i> is mapped to <i>Multiply</i> element. If the value of <i>operator</i> is <i>%</i> is mapped to <i>Modulus</i> element.
Comparison Check	left operator right	Binary Expression	left = left operator = operator right = right	If the value of <i>operator</i> is <i>&lt;</i> is mapped to <i>Less</i> element. If the value of <i>operator</i> is

PHP METAMODEL		ASTM_PHP METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
				<= is mapped to <i>NotGreater</i> If the value of <i>operator</i> is > is mapped to <i>Greater</i> element. If the value of <i>operator</i> is >= is mapped to <i>NotLess</i> .
<b>WeakLogical And</b>	left right	<b>Binary Expression</b>	left = left operator = <i>And</i> (element) right = right	No condition
<b>WeakLogical Or</b>	left right	<b>Binary Expression</b>	left = left operator = <i>Or</i> element right = right	No condition
<b>LogicalOr</b>				
<b>WeakLogical Xor</b>	left right	<b>Binary Expression</b>	left = left operator = <i>Xor</i> element right = right	No condition
<b>BiteWiseAnd</b>	left right	<b>Binary Expression</b>	left = left operator = <i>BitAnd</i> element right = right	No condition
<b>BiteWiseOr</b>	left right	<b>Binary Expression</b>	left = left operator = <i>BitOr</i> element right = right	No condition
<b>BiteWiseXor</b>	left right	<b>Binary Expression</b>	left = left operator = <i>BitXor</i> element right = right	No condition

- **Mapping from the element Multiplication to the element BinaryExpression:** The element *Multiplication* is mapped to the element *BinaryExpression*. Its definition is very similar to the *Addition* element. The attribute *operator* of the *BinaryExpression* can be mapped to three different elements depending on the type of the operator. If it is represented by a ‘/’ it is mapped to the element *Divide*, if it is a ‘\*’ it is mapped to the element *Multiply*, and finally if it is a ‘%’ it is mapped to the element *Modulus*.

## PHP Statements

### Definition Statements

Table 13-4 shows the mapping established between the element *ClassDefStatement* of the PHP metamodel to the elements of the ASTM\_PHP metamodel.

**Table 13-4.** Mappings from classDefStatement to ASTM\_PHP element.

PHP METAMODEL		ASTM_PHP METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
ClassDef Statement	className extName members	DefinitionOrDeclarationStatement	declOrDefn = AggregateType Definition	If the <i>extName</i> and <i>className</i> attributes are defined are mapped to a <i>ClassType</i> .
		Aggregate Type Definition	typeName = className aggregateType = extName and members	

- *Mapping from the element ClassDefStatement to the elements DefinitionOrDeclarationStatement and AggregateTypeDefintion:* The *ClassDefStatement* element is mapped to two elements; a *DefinitionOrDeclarationStatement* element whose attribute *declOrDefn* stores an *AggregateTypeDefintion* element. This element has two attributes: the *typeName* and the *aggregateType*. The former represents the name of the element and takes the value from the attribute *className*. The later stores an instance of the element *ClassType*.

### Expression Statements

Table 13-5 shows the mappings from the PHP elements that represent *expression statements* to elements of the ASTM\_PHP metamodel.

**Table 13-5.** Mappings from object access and member to ASTM\_PHP elements.

PHP METAMODEL		ASTM_PHP METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
ObjectAccess Statement	obj right left	Expression Statement	expression = ObjectAccess or ClassAccess (element)	The expression mapped depends on the value in the <i>obj</i>
Member	Left Obj right	Expression	--	The <i>Expression</i> mapped depends on the value of the attribute <i>right</i>

- *Mapping from the element ObjectAccessStatement to the element ExpressionStatement:* The element *ObjectAccessStatement* is also mapped to the *ExpressionStatement*. In this case, within the attribute *expression* we store the element *ObjectAccess* or *ClassAccess*.

### Control Statements

Finally, Table 13-6 presents the mappings established between the PHP elements representing the *control statements* and the ASTM\_PHP elements.

**Table 13-6. Mappings from control statements to ASTM\_PHP elements.**

PHP METAMODEL		ASTM_PHP METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
<b>Switch Statement</b>	condition case defaultStatement	<b>Switch Statement PHP</b>	switchExpression = condition cases = case / defaultStatement	If the <i>case</i> and <i>defaultStatement</i> attributes are defined are mapped to <i>Case</i> element.
<b>Case</b>	label statements	<b>CaseBlock</b>	caseExpressions = label body = statements	No condition
<b>Foreach Statement</b>	expression_array value bodyStatement	<b>Foreach Statement</b>	expressionArray = expression_array value = value body = bodyStatement	No condition
<b>While Statement</b>	condition statements	<b>While Statement</b>	condition = condition body = statements	No condition

- **Mapping from the element SwitchStatement to the element SwitchStatementPHP:** This element is mapped to the *SwitchStatementPHP* element. This element has two attributes: *switchExpression* and *cases*. The attribute *switchExpression* takes the value from the attribute *condition*. Otherwise, the attribute *cases* takes the value from the attribute *case* which stores a set of instances of the *Case* element. As we will see below, this element is mapped to the *SwitchCase* element.

## Transformations Between L1 and L2

### PHP Expressions

#### Object Access Expression

Table 13-7 presents the mappings from the ASTM\_PHP elements defining object or class accesses to KDM elements.

**Table 13-7. Mappings from expressions to KDM elements.**

ASTM_PHP METAMODEL		KDM METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
<b>Object Access or ClassAccess</b>	objectName members	<b>Action Element</b>	name = objectName kind = memberSelect / memberReplace codeElement = members actionRelation = Addresses, Reads, Writes	If the attribute <i>members</i> refers to a member unit
		<b>Action Element</b>	name = objectName kind = memberSelect / methodCall codeElement = members actionRelation = Addresses, Reads, Writes, Calls, Flow	If the attribute <i>members</i> refers to a function call

- **Mapping from the elements ObjectAccess or ClassAccess to the element ActionElement:** These two elements are mapped to an *ActionElement* element whose attribute *kind* depends on the attribute *members*. If the attribute *members* refers to a member unit, the attribute *kind* can be defined as a *memberSelect* or a *memberReplace*. Otherwise, if the attribute *members* points to a function call, the attribute *kind* is defined as a *methodCall* or *memberSelect*. In the case that the *ActionElement* is defined as a *memberSelect* or *memberReplace*, the *inputs* of this micro action are an *Invokes* relationship to the *DataElement* that represents the element or the object and a *Reads* relationship to the member being accessed (*MemberUnit* element) or the new value. The *output* is defined by a *Writes* relationship that represents the *DataElement* to which the value of the member is assigned. Finally, if the *ActionElement* is defined as a *methodCall*, the *inputs* are the *Invokes* relationship to the *DataElement* that represents the class or object, an optional ordered set of *Reads* relationship to *DataElements* that represent the input actual parameters. The *output* is a *Calls* relationship to the *MethodUnit* and an optional single *Flow* relationship to the next micro action. The attribute *name* of the *ActionElement* element takes the value from the attribute *objectName* of the *ObjectAccess/ClassAccess* element. Moreover, its attribute *codeElement* can store the objects from the attribute *members* of the *ObjectAccess/ClassAccess* element.

## PHP Statements

### Definition Statements

Table 13-8 shows the mappings defined between the ASTM\_PHP elements which represent the *definition statements* and the KDM elements.

**Table 13-8. Mappings from definition statements to KDM elements.**

<i>ASTM_PHP METAMODEL</i>		<i>KDM METAMODEL</i>		<i>CONDITION</i>
<i>Element</i>	<i>Properties</i>	<i>Element</i>	<i>Properties &amp; value</i>	
<b>DefinitionOrDeclarationStatement</b>	declOrDefn	--	Name = typeName codeElement = aggregateType codeRelation = aggregateType.derivesFrom	If aggregateType defines a value in its attribute derivesFrom, it is mapped to the attribute codeRelation
<b>Aggregate Type Definition</b>	typeName aggregateType	<b>ClassUnit</b>		

- **Mapping from the element DefinitionOrDeclarationStatement with AggregateTypeDefintion to the element ClassUnit:** The element *DefinitionOrDeclarationStatement* containing an *AggregateTypeDefintion*

element is mapped to the element *ClassUnit*. This element is a subelement of the *DataType* element which specifies three attributes: *name*, *isAbstract* and *codeElement*, which we explain in the following. The attribute *name* receives the value of the attribute *typeName* of the *AggregateTypeDefinition* element. Otherwise, the attribute *isAbstract* is an optional Boolean attribute that indicates if the element is considered as abstract or not. Finally, the attribute *codeElement* is a multivalued attribute representing the body of the *ClassUnit*. It stores the variable and function definitions defined within the attribute *aggregateType* of the *AggregateTypeDefinition* element.

In the following, we explain the mappings which refer to the ASTM\_PHP elements representing *control statements*. Table 13-9 shows these mapping between these ASTM\_PHP elements and KDM elements.

**Table 13-9. Mappings from control statements to KDM elements.**

<i>ASTM_PHP METAMODEL</i>		<i>KDM METAMODEL</i>		<i>CONDITION</i>
<i>Element</i>	<i>Properties</i>	<i>Element</i>	<i>Properties &amp; value</i>	
<b>Switch Statement PHP</b>	switchExpression cases	<b>Action Element</b>	kind = <i>Switch</i> codeElement = switchExpression, cases actionRelation = <i>Reads</i> , <i>GuardedFlow</i> , <i>FalseFlow</i>	No condition
<b>CaseBlock</b>	caseExpressions body	<b>Action Element</b>	kind = <i>Guard</i> codeElement = caseExpressions, body actionRelation = <i>Reads</i> , <i>Flow</i>	No condition
<b>Foreach Statement</b>	expressionArray value body	<b>Action Element</b>	Kind = <i>Foreach</i> , codeElement = expressionArray, value, body actionRelation = <i>Reads</i> , <i>Reads</i> , <i>TrueFlow</i> , <i>FalseFlow</i> , <i>Flow</i>	No condition
<b>While Statement</b>	condition body	<b>Action Element</b>	Kind = <i>Loop</i> , codeElement = condition, body	No condition

- **Mapping from the element *SwitchStatementPHP* to the element *ActionElement*:** The *SwitchStatementPHP* element is mapped to an *ActionElement* element whose attribute *kind* is *switch*, which represents a branching based on the value of a *StorableElement*. This *StorableUnit* can be stored within the attribute *codeElement* of the *ActionElement* element and it is mapped from the attribute *switchExpression* of the *SwitchStatementPHP* element. The *input* of this micro action is a *Reads* relationship to a *DataElement* that represents the selector value. This micro action does not

define any *output* either. Otherwise, it defines a control part composed of a set of *GuardedFlow* relationships to a second micro action of kind *guard* that represents the value of the branch of a complex condition, and *FalseFlow* relationship representing the default branch. The *Guard* micro actions are mapped from the attribute *cases* of the *SwitchStatementPHP* element and stored within the attribute *codeElement* of the *ActionElement* element. The specification of the *Guard* micro action is presented in next subsection explaining the mapping of the *CaseBlock* element. The relationships defined within the *input* and *control* parts are stored in the attribute *actionRelation* of the *ActionElement* element.

### Transformations Between L2 and L3

In this section, we present the mappings between the KDM elements and the CMS elements considering the Wordpress platform. We have determined the conditions required to define these mappings.

For the explanation of these mappings we grouped them considering the concerns defined to group the elements of the CMS Common Metamodel (see Chapter 4).

#### Navigation concern

We start explaining the mappings to CMS elements belonging to the navigation concern. These mappings are listed in Table 13-10.

**Table 13-10. Mappings from KDM elements to CMS elements of the navigation concern.**

KDM METAMODEL		CMS COMMON METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Action Element	source codeElement	Menu	path = source name = code Element	<b>Source</b> = wp-content\plugins\plugin_directory\menu.php <b>Kind</b> = call <b>Calls flow</b> → wp_create_nav_menu <b>Reads flow</b> → menu_name
Action Element	source	Menu Item	path = source name, description, position, type, status = code Element	<b>Source</b> = wp-content\plugins\plugin_directory\file.php <b>Kind</b> = call <b>Calls flow</b> → wp_update_nav_menu_item
Action Element	codeElement	MenuItem		<b>Source</b> = wp-content\plugins\plugin_directory\menuItem.php <b>Kind</b> = arrayReplace <b>Reads flows</b> → indexes and values
Action Element	source	Page	path = source id, type, title, status, date, comment, author, content,	<b>Source</b> = wp-content\plugins\block_page\page.php <b>Kind</b> = call <b>Calls flow</b> → wp_insert_post

KDM METAMODEL		CMS COMMON METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Action Element	codeElement		description, name, password = code Element	Source = wp-content\plugins\block_page\page.php Kind = arrayReplace Reads flows → indexes and values

- **Mapping from the elements ActionElement to the element Menu:** If any *ActionElement* matches with the following conditions, it is mapped to the *element Menu*. These conditions are: 1) the *ActionElement* has to represent a piece of code located in a PHP file stored in *wp-content\plugins\plugin\_directory*, 2) its attribute *kind* must be defined as a *call* and 3) its *Calls* flow has to point to a *MethodUnit* called *wp\_create\_nav\_menu*. After generating the *Menu* element, it is necessary to assign values to its attributes. Its attribute *name* takes the value from the attribute *codeElement* of the *ActionElement* element, whereas its attribute *path* takes the value from the object *SourceFile* which is stored within the attribute *source* of the *ActionElement*.

#### Presentation concern

In the following, we explain the mappings between the elements of the KDM model and the CMS elements considered within the presentation concern. Table 13-11 sums up the information explained below.

Table 13-11. Mappings from KDM elements to CMS elements of the presentation concern.

KDM METAMODEL		CMS COMMON METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Compilation Unit	source			Source = wp-content\themes\theme_directory\index.php
Comment Unit	text	Theme	path = source name = text	Source = wp-content\themes\theme_directory\style.css
Action Element	source actionRelation	Region	path = source name = action Relation	Source = Located in wp-content\themes\theme_directory\index.php Kind = call Calls flows → get_header   get_sidebar   get_footer.

- **Mapping from the elements CompilationUnit and CommentUnit to the element Theme:** We generate the *Theme* element within the CMS model from two elements of the KDM model. These elements are a *CompilationUnit* and a *Comment* element. The *CompilationUnit* maps to a

*Theme* element whether it matches with one condition: 1) it must be linked, by its attribute *source*, to a file called index.php located in wp-content\themes\theme\_directory. The object *SourceFile* which is stored within the attribute *source* of the *CompilationUnit* element is used to assign the value to the attribute *path* of the *Theme* element. On the other hand, from the *Comment* element it is possible to map the values of other attributes of the *Theme* element since this comment contains theme metadata. The condition of this *Comment* element is that it must be linked to a file called style.css located in the same directory as the *CompilationUnit*.

### Behaviour concern

Below, we explain the mappings between the elements of the KDM model and the CMS elements considered within the behaviour concern. Table 13-12 reflects data of the mappings defined to generate these CMS elements.

Table 13-12. Mappings from KDM elements to CMS elements of the behaviour concern.

KDM METAMODEL		CMS COMMON METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Action Element	source	Block	path = source name = codeElement <sup>1</sup> content = code Element	Source = wp-content\plugins\Plugin_directory\block.php Kind = call <b>Calls flow → register_widget</b>
Action Element	codeElement <sup>1</sup>			Source = wp-content\plugins\Plugin_directory\block.php Kind = assign <b>Reads flow → block_name</b> <b>Writes flow → variable \$name</b> It is the <b>parameter</b> of the function call WP_Widget
Action Element	codeElement			Source = wp-content\plugins\Plugin_directory\block.php Kind = assign <b>Reads flow → block_code</b> <b>Writes flow → variable \$message</b> It is the <b>defined</b> within the function called widget.
Compilation Unit	source	Module	path = source name = text	Source = wp-content\plugins\Plugin_directory\block.php
Comment Unit	text			Source = wp-content\plugins\Plugin_directory\block.php

KDM METAMODEL		CMS COMMON METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
MethodUnit	source name	Function	Path = source name = name	Source = wp-content\ includes\ functions.php

- **Mapping from the elements ActionElement to the element Block:** We generate the *Block* element from three *ActionElements* of the KDM model. The common condition of these three *ActionElements* is that they must represent a piece of code from PHP file located in *wp-content\plugins\plugin\_directory*. One of these *ActionElements* map to the *Block* element. This *ActionElement* must meet the following conditions: 1) its attribute *kind* is defined as a *call* and 2) its *Calls* flow must point to a *MethodUnit* called *register\_widget*. The other two *ActionElements* allow assigning a value to each attribute of the *Block* element. On the one hand, there is an *ActionElement* which allows defining the name of the *Block*. This *ActionElement* meets the following conditions it 1) has the attribute *kind* defined as an *assign* micro action, 2) has its *Writes* flow linked to a variable called \$name and 3) it represents a parameter of the function call called *WP\_Widget*. Its *Reads* flow points to the value which represents the name of the block. On the other hand, the remaining *ActionElement* allows defining the content of the *Block*. This *ActionElement* meets the following conditions it 1) its attribute *kind* is defined as an *assign* micro action, 2) has its *Writes* flow linked to a variable called \$message and 3) it is defined in the body of a *MethodUnit* called *Widget*. Its *Reads* flow points to the value representing the code implementing the block.

### Content Concern

After presenting the mappings related to the CMS elements of the behaviour concern, we explain the mappings between the elements of the KDM model and the CMS elements considered within the content concern. It is worth noting that the elements *Language* and *Vocabulary* of the CMS Common Metamodel have not been included in the mappings because their implementations in PHP are not considered by Wordpress.

Table 13-13 presents the complete list of the mappings.

**Table 13-13.** Mappings from KDM elements to CMS elements of the content concern.

KDM METAMODEL		CMS COMMON METAMODEL		CONDITION
Element	Properties	Element	Properties & value	
Action Element	codeElement	Content	text = code Element	Source = wp-content\plugins\plugin_directory\page.php Kind = arrayReplace Reads flow → index = 'post_content'
Action Element	source	Comment	path = source id, text, date, status = codeElement	Source = wp-content\plugins\plugin_directory\comment.php Kind = call Calls flow → wp_insert_comment
Action Element	codeElement	Comment	path = source id, text, date, status = codeElement	Source = wp-content\plugins\plugin_directory\comment.php Kind = arrayReplace Reads flows → indexes and values
Action Element	source codeElement	Term	path = source name = code Element id, description = code Element	Source = wp-content\plugins\plugin_directory\term.php Kind = call Calls flow → wp_insert_term
Action Element	codeElement	Term	path = source id, name, description = code Element	Source = wp-content\plugins\plugin_directory\term.php Kind = arrayReplace Reads flows → indexes and values
Action Element	source	Category	path = source id, name, description = code Element	Source = wp-content\plugins\plugin_directory\category.php Kind = call Calls flow → wp_insert_category
Action Element	codeElement	Category	path = source id, name, description = code Element	Source = wp-content\plugins\plugin_directory\term.php Kind = arrayReplace Reads flows → indexes/values

- **Mapping from the element ActionElement to the element Content:** This is a 1 to 1 transformation generating the *Content* element from an *ActionElement* of the KDM model. This *ActionElement* must meet the following conditions: 1) it has to represent a piece of located in the same file used to define the *Page* elements, 2) its attribute *kind* is defined as an *arrayReplace micro action* and 2) the identifier of this position is *post\_content*. The value is assigned to the attribute *text* of the *Content* element.

#### Permission concern

Finally, we explain the mappings between the elements of the KDM model and the CMS elements considered within the user concern. It is worth noting that the elements RoleGroup and Permission of the CMS Common Metamodel have not been included in the mappings because their implementation in PHP is not considered by Wordpress. Table 13-14 presents the complete list of the mappings.

**Table 13-14. Mappings from KDM elements to CMS elements of the user concern.**

<b>KDM METAMODEL</b>		<b>CMS COMMON METAMODEL</b>		<b>CONDITION</b>
<b>Element</b>	<b>Properties</b>	<b>Element</b>	<b>Properties &amp; value</b>	
Action Element	source codeElement	Role	path = source name = code Element <sup>1</sup> rolePermission= code Element <sup>2</sup>	Source = wp-content\plugins\plugin_directory\role.php Kind = call <b>Calls flow</b> → add_role
Action Element	codeElement			Source = wp-content\plugins\plugin_directory\role.php Kind = arrayReplace <b>Reads flows</b> → indexes and values
Action Element	source	User	path = source id ,name, user_login, password, email, regDate = codeElement	Source = wp-content\plugins\plugin_directory\user.php Kind = call <b>Calls flow</b> → wp_insert_user
Action Element	codeElement			Source = wp-content\plugins\plugin_directory\user.php Kind = arrayReplace <b>Reads flows</b> → indexes and values

- **Mapping from the element ActionElement to the element Role:** From two *ActionElements* of the KDM model we generate the *Role* element within the CMS model. The first condition, which is common to both *ActionElements* is that they have to represent a piece of code in a PHP file located in wp-content\plugins\plugin\_directory. One of these *ActionElements* map to the *Role* element. This *ActionElement* must match with the following conditions: 1) its attribute *kind* is defined as a *call* micro action since it represents a function call and 2) its *Calls flow* must point to a *MethodUnit* called *add\_role*. The other *ActionElement* allows assigning value to each attribute of the *User* element. The conditions of this *ActionElement* are: 1) the attribute *kind* must be defined as an *arrayReplace* micro action since it is the assignment of a Boolean value to a position of an array and 2) the identifier of this position have to match with one of the indexes that represent the name of the default permissions defined by Wordpress (*read*, *edit\_posts*,

*delete\_posts* and *upload\_files among others*). If the value is *true* we establish a relationship between the *Role* element and the *Permission* element.

*Appendix F: Transformations  
of the Case Study of Websana*

---

---



In this Appendix, we explain the rest of transformations executed for our case study of *Websana*. To not overextend this Appendix, we explain just one element of each table presented.

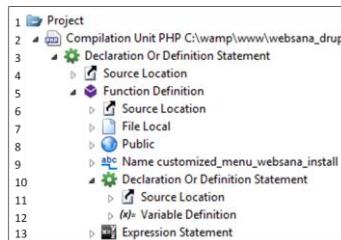
### Generation of ASTM\_PHP Model

In Table 14-1, we present the T2M transformation of the PHP code that implements the element *Menu* (see Figure 7-6) to the elements of the ASTM\_PHP model.

**Table 14-1. Transformation from PHP elements to ASTM\_PHP model.**

PHP CODE	PHP	ATTRIBUTES	ASTM_PHP	ATTRIBUTES
Function customized_menu_websana_install()	Function Def Statement	Visibility = public functionIdentifier = Name params = null body = VariableDefStatement + ReturnStatement	DeclarationOrDefinitionStatement	declOrDefn: FunctionDefinition
\$menu= array('title' =>'customized_menu_websana'...);	Variable Def Statement	Visibility = public Left = Name Right = ArrayType	DeclarationOrDefinitionStatement	declOrDefn: variableDefinition
Array (‘menu_name’ =>‘customized_menu_websana’)	Array Type	Params = arrayEntry	Collection Expression	expressionList = DuplaArray
‘menu_name’ = ‘customized_menu_websana’	Array Entry	Key = StringLiteral Value = StringLiteral	DuplaArray	Index = StringLiteral Value = StringLiteral
menu_save (\$menu)	FunctionCallStatement Exp	fun: FunctionCallExpression	Expression Statement	expression = FunctionCallExpression
	FunctionCallExpression	functionIdentifier = Variable params = Variable	FunctionCall Expression	calledFunction = identifierReference actualParams = ByValueActualParameterExpression
\$menu	Variable	Value = \$menu	ByValueActual ParameterExpression	value=IdentifierReference

- **Transformation from the function definition *customized\_menu\_websana\_install* to the element *FunctionDefinition*:** As we can see in Table 14-1, the PHP code implementing the function definition *customized\_menu\_websana\_install* is represented with the PHP element *FunctionDefStatement*. The attributes of this element are, *visibility* which takes the value of *public*, *functionIdentifier* which stores an element *Name* which defines the name of the function definition, *params* which in this case does not define any parameter (null) and *body* which stores a *VariableDefStatement* and a *ReturnStatement* as sentences conforming the body of the function definition. The element *FunctionDefStatement* is transformed into two ASTM\_PHP elements the *DeclarationOrDefinitionStatement* and a *FunctionDefinition*. The attribute *declOrDefn* of the *DeclarationOrDefinitionStatement* contains the *FunctionDefinition*. The *FunctionDefinition* is composed of the attributes *accesskind* which takes the value of *public* from the attribute *visibility*, the *identifierName* which stores a *Name* taken from the attribute *identifierName*, the *formalParameters* which in this case does not take any value from the *params* attribute, and the *body* which stores another *DeclarationOrDefinitionStatement* and a *ExpressionStatement* mapped from the attribute *body*. Figure 14-1 shows these ASTM\_PHP elements in the ASTM\_PHP model using the tree-like graphical editor implemented for the ASTM\_PHP DSL.



**Figure 14-1. AST\_M\_PHP model representing the function *customized\_menu\_websana\_install*.**

Table 14-2 shows the transformations of the PHP elements that implement the function definition *websana\_diets()* which defines the content displayed in the Web page.

**Table 14-2. Transformations from the PHP elements to the ASTM\_PHP model.**

PHP CODE	PHP	ATTRIBUTES	ASTM_PHP	ATTRIBUTES
Function websana_diets()	FunctionDefStatement	Visibility = public	DeclarationOrDefinitionStatement	declOrDefn: FunctionDefinition

PHP CODE	PHP	ATTRIBUTES	ASTM_PHP	ATTRIBUTES
		functionIdentifier = Name params = null body = ReturnStatement	nt Function Definition	accessKind = public identifierName=Name formalParameters = null body = ReturnStatement
return "I Eat with others whenever possible..."	Return Statement	return = StringLiteral Value = \$items	Return Statement	returnValue = Literal

All the transformations between the PHP elements and ASTM\_PHP elements of the Table 14-2 have been explained previously with the explanation of the transformations defined in Table 14-1. In Figure 14-2, we present the fragment of the ASTM\_PHP model where this function definition is represented.

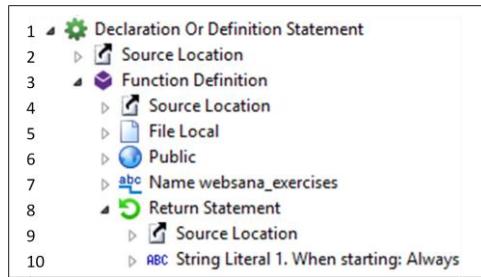


Figure 14-2. ASTM\_PHP model for the function call *websana\_exercises*.

### Generation of Code Model

Concretely, Table 14-3 is focused on the mappings from the ASTM\_PHP elements that define the function definition *customized\_menu\_websana\_install*.

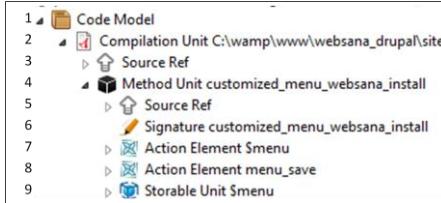
Table 14-3. Transformations from the ASTM\_PHP model to Code model.

PHP CODE	ASTM_PHP	ATTRIBUTES	KDM	ATTRIBUTES
Function customized_menu _websana_install() )	DeclarationOrDefinitionStatement	declOrDefn: FunctionDefinition	--	--
	Function Definition	accessKind = public locationInfo = path identifierName= Name formalParameters = null body = DeclarationOrDefinitionStatement + ReturnStatement	Method Unit	export = public source = SourceRef name = customized_menu_web sana_install codeElement= ActionElement + ActionElement + StorableUnit
			Signature	parameterUnit = null

PHP CODE	ASTM_PHP	ATTRIBUTES	KDM	ATTRIBUTES
\$menu= array('menu_name' =>'customized_menu_websana');	DeclarationOrDefinitionStatement	declOrDefn: variableDefinition	Action Element	kind = compound name = compound codeElement = ActionElement (kind=newArray) + 3 ActionElements (kind= arrayReplace)
		Accesskind = public identifierName = Name initialValue = CollectionExpression.		
Array ('menu_name' =>'customized_menu_websana')	Collection Expression	expressionList = DuplaArray	Action Element	kind=newArray name = newArray actionRelation = Creates, Writes and Flow
'menu_name' = 'customized_menu_websana'	DuplaArray	Index = StringLiteral Value = StringLiteral	Action Element	kind = arrayReplace name = arrayReplace codeElement = 2 Values actionRelation = Addresses, 2 Reads, Writes, Flow.
menu_save (\$menu)	ExpressionStatement Function Call Expression	expression = FunctionCallExpression calledFunction = identifierReference actualParams = ByValueParameterExpression	Action Element	kind = call name= menu_save actionRelation = Calls, Reads
\$menu	ByValueActual ParameterExpression	value=IdentifierReference		

- **Transformation of the FunctionDefinition element:** As we can see in Table 14-3, the *FunctionDefinition* element extracted from the PHP code defining the function definition *customized\_menu\_websana\_install*, is transformed into two KDM elements, the *MethodUnit* element and the *Signature* element. The *MethodUnit* element is composed of four attributes: *export*, *source*, *name* and *codeElement*. The *export* attribute which defines the visibility of the method takes the value *public* from the attribute *accesskind* of the *FunctionDefinition* element. The attribute *source* takes the value from the attribute *locationInfo*. The attribute *name* takes the value from the attribute *identifierName*. Finally, the attribute *codeElement* stores the KDM elements representing the sentences which compose the body of the function definition. In this case, there are two *ActionElements*. On the other hand, the *Signature* element stores the parameters passed to the function definition. In this case, any parameter is passed. Figure 14-3 shows

these KDM elements in the Code model represented with the tree-like graphical editor implemented for the KDM\_CODE DSL.



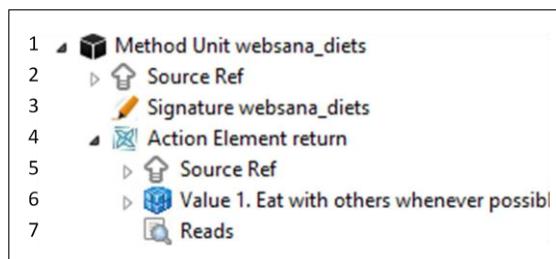
**Figure 14-3. Code model representing the function *customized\_menu\_websana\_install*.**

Table 14-4 shows the transformations of the ASTM\_PHP elements that implement the function definition *websana\_diets* to the KDM elements.

**Table 14-4. Transformations from the ASTM\_PHP model to Code model.**

PHP CODE	ASTM_PHP	ATTRIBUTES	KDM	ATTRIBUTES
Function websana_diets()	DeclarationOrDefinitionStatement	declOrDefn = FunctionDefinition	Method Unit	export = public source = SourceRef name = websana_diets codeElement= ActionElement
	Function Definition	accessKind = public identifierName= Name formalParameters = null body = ReturnStatement		parameterUnit = null
	Return Statement	returnValue = Literal	Action Element	kind = return name = return actionRelation = Reads
return "1. Eat with others whenever possible..."				

All the transformations between these elements, have been explained previously so that we centre our attention in presenting the fragment of the resulting Code model which is presented in Figure 14-4.



**Figure 14-4. Code model model for the function call *websana\_diets*.**

## Generation of CMS Model

Table 14-5 shows the transformations from the Code elements representing the *customized\_menu\_websana\_install* which defines the *Menu*. The transformations explained in this section are the implementation of the mappings defined in Section 5.3.3.

**Table 14-5. Transformation from the Code model to the element Menu.**

PHP CODE	KDM	ATTRIBUTES	CMS	ATTRIBUTES
Function customized_menu_websana_install	Mehod Unit	export = public source = SourceRef name = customized_menu_websana_install codeElement = ActionElement (kind = compound) + ActionElement (kind = call) + StorableUnit	--	--
	Signature	parameterUnit = null	--	--
\$menu= array('menu_name' =>'customized_menu_websana');	Action Element	kind = compound name = compound codeElement = ActionElement (kind=newArray) + 3 ActionElements (kind= arrayReplace)	--	--
Array ('menu_name' =>'customized_menu_websana', 'description' => 'Further information about diets and excercises')	Action Element	kind=newArray name = newArray actionRelation = Creates, Writes and Flow	--	--
'menu_name' => 'customized_menu_websana'	Action Element	kind = arrayReplace name = arrayReplace codeElement = 2 Values actionRelation = Addresses, 2 Reads, Writes, Flow.	--	name = customized_menu_websana
'description' => 'Further information about diets and excercises')	Action Element	kind = arrayReplace name = arrayReplace codeElement = 2 Values actionRelation = Addresses, 2 Reads, Writes, Flow.	--	description = Further information about diets and excercises
menu_save (\$menu)	Action Element	kind = call name= menu_save actionRelation = Calls, Reads	Men u	

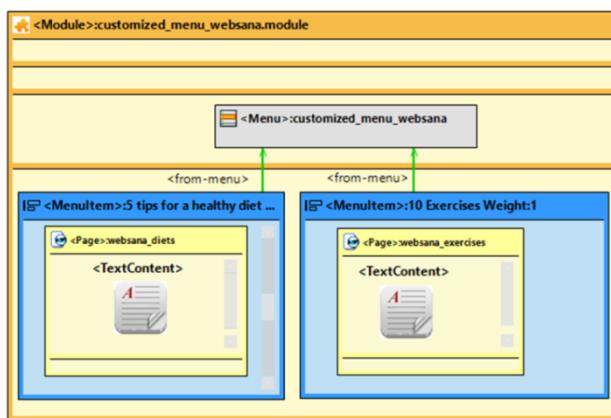
- Transformation of the ActionElement element into Menu element:** As it is shown in Table 14-5, an *ActionElement* whose attribute *kind* is defined as *call* and its name is *menu\_save* is transformed into the *Menu* element in the CMS model. The mappings with the conditions of this transformation are

presented in Table 13-10. As we can see in Table 14-5, the attributes of the element *Menu* are mapped from the KDM elements of the Code model. The *ActionElement* whose *kind* attribute is defined as *ArrayReplace* and represents the PHP code '*menu\_name*' => '*customized\_menu\_websana*' is transformed into the attribute *name* of the *Menu* element. Otherwise, the *ActionElement* defined as an *ArrayReplace* representing the PHP code '*description*' => 'Further information about diets and excercises') is transformed into the attribute *description*.

**Table 14-6.** Transformation from the Code model to the element *TextContent*.

PHP CODE	KDM	ATTRIBUTES	CMS	ATTRIBUTES
Function websana_diets()	Method Unit	export = public source = SourceRef name = websana_diets codeElement= ActionElement	Text Content	
	Signature	parameterUnit = null	--	--
return "1. Eat with others whenever possible..."	Action Element	kind = return name = return actionRelation = Reads	--	Text = 1. Eat with others whenever possible

- **Transformation of the *ActionElement* element into *Page* element:** As we can see in Table 14-6, the *TextContent* element is transformed from the *MethodUnit* element called *websana\_diets*. To transform this *MethodUnit* into the *TextContent* element is necessary that its name must be the same as the name of the *Page* we explained previously. The attribute *text* of the *TextContent* is defined from an *ActionElement* defined as a *return*.



**Figure 14-5.** CMS model representing the *customized\_menu\_websana*.

### Generation of Target Code Model

In Table 14-7 we can see the transformations between the *Menu* element of the CMS model and the KDM elements of the Code model. A *Menu* on Wordpress is created by a call to the function *wp\_create\_nav\_menu* defined in the function definition *create\_menu\_customized\_menu\_websana*, as we see in Figure 7-9.

**Table 14-7. Transformation from the restructured CMS model to Target Code model.**

PHP CODE	CMS	ATTRIBUTES	KDM	ATTRIBUTES
function create_menu_customized_menu_websana ()	Menu	title = customized_menu_websana description = Further information about diets and excercises	Method Unit	export = public source = SourceRef name = create_menu_customized_menu_websana codeElement= 13 ActionElements, Value, StorableUnit, Signature
			Signature	parameterUnit = null
\$var_0 = wp_create_nav_menu ('customized_menu_websana');		--	Action Element	kind = call name= menu_save actionRelation = Calls, Reads, Writes
customized_menu_websana		--	Value	name = customized_menu_websana
\$var_0		--	Storable Unit	name = \$var_0
register_activation_hook (__FILE__, 'create_menu_customized_menu_websana');		--	Action Element	kind = call name= menu_save actionRelation = Calls, Reads, Writes

- **Transformation of the Menu element:** As we can see in Table 14-7, from the *Menu* element the KDM elements required to implement it are created, such as *MethodUnit* with a *Signature* element and *ActionElements*. The function definition *create\_menu\_customized\_menu\_websana* is defined by the *MethodUnit* and *Signature*. The visibility of this *MethodUnit* is defined as *public* specified with the attribute *export*. Its attribute *codeElement* stores all the sentences defined with the function definition. Otherwise, the *Signature* is empty since any parameter is not passed to the function, so that its attribute *parameterUnit* is initialized as null. On the other hand, the call to the function *wp\_create\_nav\_menu* is defined in the target Code model as an *ActionElement* determined as a *call*. Finally, the function call *register\_activation\_hook* is represented also with an *ActionElement* defined as a call. Figure 14-6 shows the Code model with the KDM elements generated with this transformation.

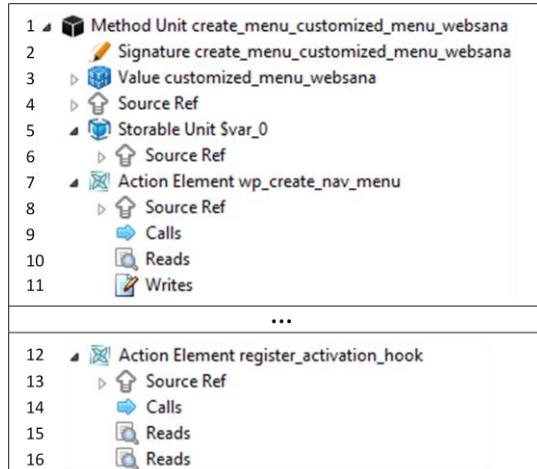


Figure 14-6. Target Code model representing the function `create_menu_customized_menu_websana`.

### Generation of Target ASTM\_PHP Model

In Table 14-8, we can see the transformations between the KDM elements that represent the *Menu* and the ASTM\_PHP elements of the ASTM\_PHP model.

Table 14-8. Transformation from the Target Code model to Target ASTM\_PHP model.

PHP CODE	KDM	ATTRIBUTES	ASTM_PHP	ATTRIBUTES
function create_menu_c ustomized_me nu_websana ()	Method Unit	export = public source = SourceRef name = create_menu_customized _menu_websana codeElement= 13 ActionElements, Value, StorableUnit, Signature	DeclarationOrD efinitionStateme nt	declOrDefn=Function Definition
			Function Definition	accessKind = public identifierName= Name formalParameters = null body = ReturnStatement (actualizar)
	Signature	parameterUnit = null	Name	nameString= create_menu_custo med_menu_websana
\$var_0 = wp_create_nav _menu('custo ')	Action Element	kind = call name= menu_save	DeclarationOrD efinitionStateme nt	declOrDefn = VariableDefinition

<b>PHP CODE</b>	<b>KDM</b>	<b>ATTRIBUTES</b>	<b>ASTM_PHP</b>	<b>ATTRIBUTES</b>
mized_menu_websana');		actionRelation = Calls, Reads	Variable Definition	Accesskind = public identifierName = Name initialValue = functionCallExpression
			FunctionCall Expression	calledFunction = identifierReference actualParams = ByValueParameterExpression
			Identifier Reference	name = Name refersTo = FunctionDefinition
			FunctionDefinition	identifierName = Name
			Name	name = wp_create_nav_menu
customized_menu_websana	Value	name = customized_menu_websana	ByValue Actual Parameter Expression	value = StringLiteral
		kind = call name= menu_save actionRelation = Calls, Reads, Reads	StringLiteral	Value = customized_menu_websana
register_activation_hook (__FILE__,'create_menu_customize_d_menu_websana');			Expression Statement	expression = FunctionCallExpression
			FunctionCall Expression	calledFunction = identifierReference actualParams = 2 ByValueParameterExpression
			Identifier Reference	name = Name refersTo = FunctionDefinition
			Function Definition	identifierName = Name
create_menu_customize_d_menu_websana	Value	name = create_menu_customize_d_menu	Name	name = register_activation_hook
			ByValue Actual ParameterExpression	value = StringLiteral

- **Transformation of the MethodUnit element:** As we can see in Table 14-8, the function *create\_menu\_customize\_d\_menu\_websana* is represented with a *MethodUnit* and a *Signature* elements within the target Code model are transformed into a *DeclarationOrDefinitionStatement* and a *FunctionDefinition* element within the ASTM\_PHP model.

```

1   ▲ Declaration Or Definition Statement
2     ▷ Source Location
3   ▲ Function Definition
4     ▷ Source Location
5     ▷ File Local
6     ▷ Public
7     ▷ abc Name create_menu_customized_menu_websana
8   ▲ Declaration Or Definition Statement
9     ▷ Source Location
10    ▷ (x)= Variable Definition
11      ▷ Source Location
12      ▷ File Local
13      ▷ Public
14      ▷ abc Name $var_0
15      ▲ Function Call Expression
16        ▷ Source Location
17        ▷ Identifier Reference
18          ▷ Source Location
19          ▷ abc Name wp_create_nav_menu
20          ▷ Function Definition
21        ▲ {} By Value Actual Parameter Expression
22          ▷ Source Location
23          ▷ ABC String Literal customized_menu_websana
24
25  ...
26
27
28
29
30
31
32
33
34
35

```

Figure 14-7. Target ASTM\_PHP model representing the menu.

### Generation of PHP Code

Table 14-9 presents the transformation rules implemented in Acceleo to generate the PHP code which implements the *Menu* in the Wordpress platform.

Table 14-9. Transformations between the target ASTM\_PHP model to the PHP code.

ASTM_PHP	ATTRIBUTES	M2T TRANSFORMATION RULE	PHP CODE
DeclarationOrDefinitionStatement	declOrDefn= Function Definition	[template private generateFunctionDefinition(aDeclaration : FunctionDefinition)] function [aDeclaration.identifierName.nameString/]() [for (param: FormalParameterDefinition aDeclaration.formalParameters) separator(',')] [generateFormalParam(param)/]/for] ){ [for (bodySt:Statement  aDeclaration."body")] [generateStatement(bodySt)/]	function create_menu _customized _menu_webs ana()
Function Definition	accessKind = public identifierName= Name formalParameters = null body = ReturnStatement		

ASTM_PHP	ATTRIBUTES	M2T TRANSFORMATION RULE	PHP CODE
	t (actualizar)	[/for]}{/template]	

- **Transformation rule generateFunctionDefinition:** As we can see in Table 14-9, the transformation rule called *generateFunctionDefinition* generates the definition of the function *create\_menu\_customized\_menu\_websana* in PHP. Acceleo defines templates to generate the code. As we can see, the rule *generateFunctionDefinition* is defined as a template, marks the starting of the function definition with the token **function**. Then, the attribute *nameString* from the element *Name* stored in the attribute *identifierName* of a *FunctionDefinition* is taken to define the name of the function. After the name of the function, a parentheses is printed to determine the set of parameters passed to the function. For this, a *loop* (for) parsers the values stored in the attribute *formalParameters* from the *FunctionDefinition*. As we can see, this parameters are separated by commas. To print the parameters, a transformation rule called *generateFormalParam* is launched. After printing all the parameters, a parentheses is printed to indicate the end of the parameters definition. After the parameter definition, an open brace is printed to indicate the starting of the body of the function definition. Then a *loop* (for) parsing the statements defined in the *body* is implemented. For each statement found in the *body*, the transformation rule *generateStatement* is launched. To define the end of the body, a close brace is printed.

Table 14-10 presents the transformation rule implemented in Acceleo called *generateVariableDefinition*. This transformation rule allows generating the definition of the variable.

**Table 14-10. Transformations between the target ASTM\_PHP model to the PHP code.**

ASTM_PHP	ATTRIBUTES	M2T TRANSFORMATION RULE	PHP CODE
Declaration OrDefinition Statement	declOrDefn = Variable Definition	[template private generateVariableDefinition(aDeclaration : VariableDefinition)] [aDeclaration.identifierName.nameString/] = [generateExpression(aDeclaration.initialV alue)/] [/template]	
Variable Definition	Accesskind = public identifierName = Name initialValue = functionCallExpress		\$var_0 = wp_create_n av_menu('cu stomized_me nu_websana' );
Function Call Expression	calledFunction = identifierReference actualParams = ByValueParameterE xpression	[template private generateFunctionCallExp(functionCall : FunctionCallExpression)] [functionCall.calledFunction.oclAsType(I dentifierReference).name.nameString/] ( [for (param: ActualParameter	

		functionCall.actualParams) separator(',')] [generateActualParam(param.oclAsType(WithValueActualParameterExpression))] [/for][/template]	
Identifier Reference	name = Name refersTo = FunctionDefinition	[template private generateIdRef(aExpression : IdentifierReference)] [aExpression.name.nameString/] [/template]	
WithValueActual ParameterExpr ession	value = StringLiteral	[template private generateActualParam(param :WithValueActualParameterExpression)] [generateExpression(param.value)] [/template]	customized_menu_websa
StringLiteral	value = customized_menu_websana	[template private generateLiteral(aExpression :Literal)] [if(aExpression.oclIsTypeOf(StringLiteral))] [aExpression.value/] [else][aExpression.value/][if][/template]	na \$var_0

- **Transformation rule generateVariableDefinition:** This transformation rule prints the PHP code for the *VariableDefinition*. It prints the name of the variable as the left hand operand, the assign operator = and the expression as the right hand operand. To print the expression the rule *generateExpression* is launched.



*Bibliography and Online  
Resources*

---



## Bibliography

- Trias, F., de Castro, V., López-Sanz, M., & Marcos, E. (2014). A Toolkit for ADM-based Migration: Moving from PHP code to KDM Model in the Context of CMS-based Web Applications. In *23rd International Conference on Information Systems Development (ISD)*.
- Hill, H. (2014). Cascade Server. Retrieved from <http://www.hannonhill.com/products/>
- Verse, C. (2014). Content Verse. Retrieved from <http://www.contentverse.com/>
- Drupal. (2014). Drupal CMS. Retrieved from <http://drupal.org/>
- Joomla! (2014). Joomla! CMS. Retrieved from <http://www.joomla.org/>
- Wagner, C. (2014). Model-Driven Software Migration. In *Model-driven Software Migration: A Methodology: Reengineering, Recovery and Modernization of Legacy Systems* (pp. 67–105).
- ACM, S. (2014). SDL ACM. Retrieved from <http://www.sdl.com/>
- W3Techs. (2014). Web Technology Surveys. Retrieved from [http://w3techs.com/technologies/history\\_overview/content\\_management/ms/y](http://w3techs.com/technologies/history_overview/content_management/ms/y)
- Wordpress. (2014). Wordpress CMS. Retrieved from <http://wordpress.org/>
- Trias, F., de Castro, V., López-Sanz, M., & Marcos, E. (2013a). A Systematic Literature Review on CMS-based Web Applications. In *ICSOFT*.
- Eclipse Foundation. (2013). Atlas Transformation Language. *Eclipse Foundation*. Retrieved from <http://www.eclipse.org/atl/>
- BuiltWith. (2013). CMS Usage Statistics: Statistics for websites using CMS technologies. Retrieved from <http://trends.builtwith.com/cms/#>
- PHP Group. (2013). PHP manual. Retrieved from <http://www.php.net/manual/en/index.php>
- Trias, F., de Castro, V., López-Sanz, M., & Marcos, E. (2013b). Reverse Engineering Applied to CMS-Based Web Applications Coded in PHP: A Proposal of Migration. In *Evaluation of Novel Approaches to Software Engineering* (pp. 241–256).
- Pérez-Castillo, R., Cruz-Lemus, J. a., Garcia-Rodriguez de Guzman, I., & Piattini, M. (2012). A family of case studies on business process mining using MARBLE. *Journal of Systems and Software*, 85(6), 1370–1385. doi:10.1016/j.jss.2012.01.022

- Normantas, K., Sergejus, S., & Vasilecas, O. (2012). An Overview of the Knowledge Discovery Metamodel. In *13th International Conference on Computer Systems and Technologies* (pp. 52–57). ACM.
- Trias, F. (2012). Building CMS-based Web Applications Using a Model-driven Approach. In *Sixth International Conference on Research Challenges in Information Science (RCIS)* (pp. 1 – 6).
- Pérez-Castillo, R., García-Rodríguez de Guzmán, I., Caivano, D., & Piattini, M. (2012). Database Schema Elicitation to Modernize Relational Databases. In *14th International Conference on Enterprise Information Systems* (pp. 126–132). Wrocław, Poland: Springer Berlin.
- Bigardel, L., & Garcia, M. (2012). Emfatic.
- Eclipse Foundation. (2012). Epsilon. Retrieved from <http://www.eclipse.org/epsilon/>
- Epsilon. (2012). EuGENia GMF Tutorial. Retrieved from <http://www.eclipse.org/epsilon/doc/articles/eugenia-gmf-tutorial/>
- W3C. (2012). EXtensible Stylesheet Language Transformation (XSLT) v2.0. Retrieved April 30, 2012, from [www.w3.org/Style/XSL/](http://www.w3.org/Style/XSL/)
- Rodríguez-Echevarría, R., Clemente, P. J., Villalobos, M. D., & Sánchez-Figueroa, F. (2012). Extracting Navigational Models from Struts-Based Web Applications. In *ICWE* (pp. 419–426).
- Rodríguez-Echeverría, R., Conejero, J. M., Clemente, P. J., Villalobos, M. D., & Sánchez-Figueroa, F. (2012). Generation of WebML Hypertext Models from Legacy Web Applications. In *WSE'12* (pp. 91–95).
- Jiménez, Á. (2012). *Incorporando la Gestión de la Trazabilidad en un entorno de Desarrollo de Transformaciones de Modelos Dirigido por Modelos*. Universidad Rey Juan Carlos.
- ISO/IEC. (2012). *Information technology - Architecture-Driven Modernization (ADM): Knowledge Discovery Metamodel (KDM)* (p. 354).
- Pérez-Castillo, R., García-Rodríguez de Guzmán, I., Piattini, M., & Weber, B. (2012). Integrating event logs into KDM repositories. *Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC '12*, 1095. doi:10.1145/2245276.2231949
- OMG. (2012a). *ISO/IEC 19505-1:2012 Unified Model Language*.
- Rodríguez-Echeverría, R., Conejero, J. M., Pedro J. Clemente, Pavón, V. M., & Sánchez-Figueroa, F. (2012). Model Driven Extraction of the Navigational Concern of Legacy Web Applications. In M. Grossniklaus & M. Wimmer (Eds.), *ICWE 2012* (Vol. 7703, pp. 56–70). Springer-Verlag Berlin Heidelberg 2012.

- Pérez-castillo, R., Garcia-Rodriguez de Guzman, I., & Piattini, M. (2012). Model-driven Reengineering. In *Emerging Technologies for the Evolution and Maintenance of Software Models* (pp. 200–229).
- Brambilla, M., Wimmer, M., & Cabot, J. (2012). *Model-Driven Software Engineering in Practice*.
- Moratalla, J. (2012). *PREMISA: Un Proceso para la Evolución y Modernización de Sistemas Aplicación a Red.es*. Rey Juan Carlos University.
- OMG. (2012b). Structured Metrics Metamodel. Retrieved February 12, 2013, from <http://www.omg.org/spec/SMM/>
- Rossini, A., Mughal, K. A., Wolter, U., Rutle, A., & Lamo, Y. (2011). A Formal Approach to Data Validation Constraints in MDE. In *5th International Workshop on Harnessing Theories for Tool Support in Software* (pp. 65–76).
- Pérez-Castillo, R., García-Rodríguez de Guzmán, I., Piattini, M., Weber, B., & Places, Á. S. (2011). An empirical comparison of static and dynamic business process mining. *Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11*, 272. doi:10.1145/1982185.1982249
- López-Sanz, M. (2011). *Archimedes: A Service-Oriented Framework for Model-driven Development of Software Architectures*. Universidad Rey Juan Carlos.
- Pérez-Castillo, R., Guzmán, I. G. R. de, & Piattini, M. (2011). Architecture-Driven Modernization. In *Modern Software Engineering Concepts and Practices: Advanced Approaches*.
- Pérez-Castillo, R., García-Rodríguez de Guzmán, I., & Piattini, M. (2011). Business process archeology using MARBLE. *Information and Software Technology*, 53(10), 1023–1044. doi:10.1016/j.infsof.2011.05.006
- Vasilecas, O., & Normantas, K. (2011a). Deriving Business Rules from the Models of Existing Information Systems, 95–100.
- Vasilecas, O., & Normantas, K. (2011b). Deriving Business Rules from the Models of Existing Information Systems, 95–100.
- Van Hoorn, A., Sören, F., Goerigk, W., Hasselbring, W., Knoche, H., Köster, S., ... Wittmüss, N. (2011). DynaMod Project : Dynamic Analysis for Model-Driven Software Modernization. In *1st International Workshop on Model-Driven Software Migration* (pp. 12–13). Oldenburg, Germany.
- Souer, J., Urlings, T., Helms, R., & Brinkkemper, S. (2011). Engineering Web Information Systems: A Content Management System-based Approach. In *13th International Conference on Information Integration and Web-based Applications and Services* (pp. 5–7).

- Volter, M. (2011). From programming to modeling-and back again. *Software, IEEE*, 28(6), 20–25.
- Pérez-Castillo, R., de Guzmán, I. G.-R., & Piattini, M. (2011). Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. *Computer Standards & Interfaces*, 33(6), 519–532. doi:10.1016/j.csi.2011.02.007
- Pérez-castillo, R., Fernández-ropero, M., Garcia-Rodriguez de Guzman, I., & Piattini, M. (2011). MARBLE: A Business Process Archeology Tool. In *27th IEEE International Conference on Software Maintenance* (pp. 578 – 581).
- Bollati, V. (2011). *MeTAGeM: Entorno de desarrollo de Transformaciones de modelos dirigido por modelos*. Universidad Rey Juan Carlos.
- Rodríguez-Echeverría, R., Clemente, P. J., Preciado, J. C., & Sánchez-Figueroa, F. (2011). Modernization of Legacy Web Applications into Rich Internet Applications. In A. Harth & N. Koch (Eds.), *International Conference on Web Engineering* (pp. 236–250). Springer-Verlag Berlin Heidelberg 2001.
- Pérez-castillo, R., Sánchez-gonzález, L., Piattini, M., García, F., & Garcia-Rodriguez de Guzman, I. (2011). Obtaining Thresholds for the Effectiveness of Business Process Mining. In *International Symposium on Empirical Software Engineering and Measurement* (pp. 453 – 462).
- Shreves, R. (2011). *Open Source CMS Market Share*. (W. & S. White paper, Ed.). Bali, Indonesia. Retrieved from <http://waterandstone.com/downloads/2008OpenSourceCMSMarketSurvey.pdf>.
- Pérez-Castillo, R., Weber, B., Garcia-Rodriguez de Guzman, I., & Piattini, M. (2011). Process mining through dynamic analysis for modernising legacy systems. *IET Software*, 5(3), 304. doi:10.1049/iet-sen.2010.0103
- OMG. (2011). XML Metadata Interchange. Retrieved from <http://www.omg.org/spec/XMI/2.4.1/>
- Souer, J., & Joor, D.-J. (2010). An approach to identify commonalities in web application engineering for a web content management system. *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services - iiWAS '10*, 558. doi:10.1145/1967486.1967572
- Plain Black Corporation. (2010). CMS matrix. Retrieved from <http://cmsmatrix.org/>
- Fowler, M. (2010). *Domain Specific Languages*. Addison Wesley Professional.
- Fraternali, P., Comai, S., Bozzon, A., & G., T. (2010). Engineering Rich Internet Applications with a Model-Driven Approach. *ACM Transactions on the web*, 4, 47. doi:10.1145/1734200.1734204

- Pérez-castillo, R., Garcia-Rodriguez de Guzman, I., & Piattini, M. (2010). Implementing Business Process Recovery Patterns through QVT Transformations. In *Proceedings of the Third international conference on Theory and practice of model transformations* (pp. 168–183).
- Van Amstel, M., van den Brand, M., & Nguyen, P. H. (2010). Metrics for model transformations. In *9th Belgian-Netherlands Software Evolution Workshop (BENEVOL 2010)*. Lille (France).
- Sendall, S., & Kozaczynski, W. (2010). Model Transformation - the Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5), 42–45.
- Pérez-castillo, R., Weber, B., García-Rodríguez de Guzmán, I., & Piattini, M. (2010). Modernizing Legacy Systems through Runtime Models. In O. C. S. L. (España) Orlando Avila-García, I.-É. des M. de N. (Francia) Jordi Cabot, P. S. L. (España) Javier Muñoz, U. de C. (España) Jose Raúl Romero, & U. de M. (España) Antonio Vallecillo (Eds.), *Desarrollo de Software Dirigido por Modelos*.
- Bruneliere, H., Cabot, J., Jouault, F., & Madiot, F. (2010). MoDisco : A Generic And Extensible Framework For Model Driven Reverse Engineering. In *International conference on Automated software engineering - ASE '10* (pp. 173–174).
- Barbier, G., Bruneliere, H., Jouault, F., Lennon, Y., & Madiot, F. (2010). MoDisco, a model-driven platform to support real legacy modernization use cases. In *Information Systems Transformation: Architecture-Driven Modernization Case Studies* (p. 365).
- Van Amstel, M., & van den Brand, M. (2010). Quality assessment of ATL model transformations using metrics. In *International Workshop on Model Transformation with ATL (MtATL2010)*. Málaga (Spain).
- Rodríguez-echeverría, R., Conejero, J. M., Linaje, M., Preciado, J. C., & Sánchez-figueroa, F. (2010). Re-engineering Legacy Web Applications into Rich Internet Applications. In *ICWE* (pp. 189–203).
- Kolosovs, D., Rose, L., Abid, S., Paige, R., Polack, F., & Botterweck, G. (2010). Taming EMF and GMF Using Model Transformation. In *Model Driven Engineering Languages and Systems* (pp. 211–225).
- Rothenberger, M. A., Kao, Y.-C., & Van Wassenhove, L. N. (2010). Total quality in software development: An empirical study of quality drivers and benefits in Indian software projects. *Information and Management*, 47(7-8), 372–379.
- Saraiva, J. D. S., & Silva, A. R. Da. (2010). Web Application Modeling with the CMS-ML Language. In *INForum* (pp. 461–472).
- Moritz, E., & Behrens, H. (2010). Xtext - Implement your Language Faster than the Quick and Dirty way Tutorial Summary. In *ACM international*

*conference companion on Object oriented programming systems languages and applications companion* (pp. 307–309).

Sadovykh, A., Vigier, L., Hoffmann, A., Grossmann, J., Ritter, T., Gomez, E., & Estekhin, O. (2009a). Architecture Driven Modernization in Practice: Study Results. In *2009 14th IEEE International Conference on Engineering of Complex Computer Systems* (pp. 50–57). Ieee. doi:10.1109/ICECCS.2009.39

Sadovykh, A., Vigier, L., Hoffmann, A., Grossmann, J., Ritter, T., Gomez, E., & Estekhin, O. (2009b). Architecture Driven Modernization in Practice: Study Results. In *2009 14th IEEE International Conference on Engineering of Complex Computer Systems* (pp. 50–57). Ieee. doi:10.1109/ICECCS.2009.39

OMG. (2009). Architecture Driven Modernization specification of the OMG. Retrieved from <http://adm.omg.org/>

Volter, M. (2009). Best Practices for DSLs and Model-Driven Development. *Journal of Object Technology*, 8(6), 79–102.

Saraiva, J. D. S., & Silva, A. R. Da. (2009a). CMS-based Web-Application Development Using Model-Driven Languages. In *4th International Conference on Software Engineering Advances* (pp. 21–26). Porto, Portugal: IEEE Computer Society.

Saraiva, J. D. S., & Silva, A. R. Da. (2009b). Development of CMS-Based Web-Applications Using a Model-Driven Approach. In *4th International Conference on Software Engineering Advances* (pp. 500–505). Ieee. doi:10.1109/ICSEA.2009.79

Gronback, R. C. (2009). *Eclipse Modeling project: a domain-specific language toolkit*. Addison-Wesley Professional.

Vara, J. M. (2009). *M2DAT: a Technical Solution for Model-driven Development of Web Information Systems*. Universidad Rey Juan Carlos.

Souer, J., Kupers, T., Helms, R., & Brinkkemper, S. (2009). Model-Driven Web Engineering for the Automated Configuration of Web Content Management Systems. In Springer-Verlag (Ed.), *ICWE 2009* (pp. 124–135). Heidelberg.

Vlaanderen, K., Valverde, F., & Pastor, O. (2009). Model-Driven Web Engineering in the CMS Domain: A Preliminary Research Applying SME. In *10th International Conference on Enterprise Information Systems* (Vol. 19, pp. 226–237). Heidelberg.

Sadovykh, A., Vigier, L., Gomez, E., & Hoffmann, A. (2009). On Study Results : Round Trip Engineering of Space Systems, (Xmi), 265–276.

Perez-Castillo, R., Garcia-Rodriguez de Guzman, I., Avila-Garcia, O., & Piattini, M. (2009). On the Use of ADM to Contextualize Data on Legacy Source

- Code for Software Modernization. In *16th Working Conference on Reverse Engineering* (pp. 128–132). Ieee. doi:10.1109/WCRE.2009.20
- Perez-Castillo, R., & García-Rodríguez de Guzmán, I. (2009). PRECISO: A Reengineering Process and a Tool for Database Modernisation through Web Services. In *Proceedings of the 2009 ACM symposium on Applied Computing* (pp. 2126–2133). ACM New York, NY, USA. doi:10.1145/1529282.1529753
- Perez-Castillo, R., Garcia-Rodriguez de Guzman, I., Caballero, I., Polo, M., & Piattini, M. (2009). PRECISO: A Reverse Engineering Tool to Discover Web Services from Relational Databases. In *16th Working Conference on Reverse Engineering* (pp. 309–310). Ieee. doi:10.1109/WCRE.2009.21
- Busch, M., Koch, N., & Unit, I. for I. / P. and S. E. (2009). Rich Internet Applications. State of the art. München, Germany: Ludwig-Maximilians-Universität München, .
- Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering – A systematic literature review. *Information and Software Technology*, 51(1), 7–15. doi:10.1016/j.infsof.2008.09.009
- Moody, D. (2009). The “Physics” of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering*, 35(6), 756–779.
- Blanco, C., Pérez-castillo, R., Hernández, A., & Fernández-Medina, E. (2009). Towards a Modernization Process for Secure Data Warehouses. In *Data Warehousing and Knowledge Discovery* (pp. 24–35). Springer Berlin / Heidelberg.
- Souer, J., & Kupers, T. (2009). Towards a Pragmatic Model Driven Engineering Approach for the Development of CMS-based Web Applications. In *5th Model Driven Web Engineering Workshop* (pp. 31–45).
- OpenArchitectureWare. (2009). Xpand. Retrieved from <http://www.eclipse.org/modeling/m2t/?project=xpand>.
- Watson, A. (2008). A brief history of MDA. *Upgrade, European Journal for the Informatics Professional*, 9(2), 7–11.
- Souer, J., & Mierloo, M. Van. (2008). A Component Based Architecture for Web Content Management: Runtime Deployable WebManager Component Bundles. *2008 Eighth International Conference on Web Engineering*, 366–369. doi:10.1109/ICWE.2008.32
- Souer, J., Honders, P., Versendaal, J., & Brinkkemper, S. (2008). A Framework for Web Content Management System Operations and Maintenance. *Journal Of Digital Information Management*, 6(4).

- Musset, J., Juliet, É., & Lacrampe, S. (2008). *Acceleo - User Guide*. Retrieved from <http://www.acceleo.org/doc/obeo/en/acceleo-2.6-user-guide.pdf>
- Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. (2008). ATL: A model Transformation Tool. *Science of Computer Programming*, 72(1-2), 31–39.
- Luinenburg, L., Jansen, S., Souer, J., & Weerd, I. Van De. (2008). Designing Web Content Management Systems Using the Method Association Approach. In *4th International Workshop on Model-Driven Web Engineering* (pp. 106–120).
- Kelly, S., & Tolvanen, J.-P. (2008). *Domain-Specific Modelling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Press.
- Budinsky, F. (2008). Eclipse Modeling Framework. Addison-Wesley Professional.
- Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2008). *EMF: Eclipse Modelling Framework* (2nd ed.). Addison Wesley Professional.
- Souer, J., Luinenburg, L., Versendaal, J., Weerd, I. van de, & Brinkkemper, S. (2008). Engineering a Design Method for Web Content Management Implementations. In *10th International Conference on Information Integration and Web-based Applications & Services* (pp. 351–358).
- Runeson, P., & Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2), 131–164.
- Vlaanderen, K., Valverde, F., & Pastor, O. (2008). Improvement of a Web Engineering Method Applying Situational Method Engineering. In *9th International Conference on Enterprise Information Systems* (pp. 147–154).
- Tikhomirov, A., & Shatali, A. (2008). *Introduction to the Graphical Modeling Framework*. Santa Clara, California.
- Selic, B. (2008). MDA manifestations. *The European Journal for the Informatics Professional*, IX(2), 12–16.
- López-Sanz, M., Acuña, C. J., Cuesta, C. E., & Marcos, E. (2008). Modelling of Service-oriented architectures with UML. *Electronic Notes in Theoretical Computer Science*, 194(4), 23–37.
- Ric Shreves. (2008). *Open Source CMS Market Share* (p. 50).
- Baxter, P., & Jack, S. (2008). Qualitative case study methodology: Study design and implementation for novice researchers. *The Qualitative Report*, 13(4), 544–559.
- OMG. (2008a). QVT. Meta Object Facility (MOF) 2.0. Query/View Transformation Specification. *OMG*. Retrieved from <http://www.omg.org/spec/QVT/1.0/PDF>

- Rossi, G., Urbieta, M., Ginzburg, J., Distante, D., & Garrido, A. (2008). Refactoring to Rich Internet Applications. A Model-Driven Approach. In *ICWE* (pp. 1–12). Yorktown Heights, NJ. doi:10.1109/ICWE.2008.41
- OMG. (2008b). Semantics of Business Vocabulary and Business Rules. Retrieved from <http://www.omg.org/spec/SBVR/1.0/>
- Kolosovs, D., Rose, L., García-Domínguez, A., & Paige, R. (2008). *The Epsilon Book* (p. 211). Retrieved from <http://www.eclipse.org/gmt/epsilon/doc/book>
- Kroiss, C., & Koch, N. (2008). The UWE Metamodel and Profile - User Guide and Reference. München: Ludwig-Maximilians-Universität München (LMU), Institute for Informatics.
- Saraiva, J. D. S., & Silva, A. R. Da. (2008). The WebComfort Framework: An Extensible Platform for the Development of Web Applications. In 34th Euromicro Conference on Software Engineering and Advanced Applications (Ed.), *34th Euromicro Conference Software Engineering and Advanced Applications* (pp. 19–26).
- Koch, N., Knapp, A., Zhang, G., & Baumeister, H. (2008). UWE- UML-based Web Engineering. In *Web Engineering: Modelling and Implementing Web Applications* (pp. 157–191). London, UK: Springer London.
- Valverde, F., Valderas, P., Fons, J., & Pastor, O. (2007). A MDA-based Environment for Web Applications Development: From Conceptual Models to Code. In *6th International Workshop on Web-Oriented Software Technologies*.
- Fischer, G., Lusiardi, J., & Wolff von Gudenberg, J. (2007). Abstract Syntax Trees - and their Role in Model Driven Software Development. In *International Conference on Software Engineering Advances (ICSEA 2007)* (p. 38). Cap Esterel: IEEE Computer Society. doi:10.1109/ICSEA.2007.12
- OMG. (2007). ADM task force.
- Garcia-Rodriguez de Guzman, I., Polo, M., & Piattini, M. (2007). An ADM Approach to Reengineer Relational Databases towards Web Services. In *4th Working Conference In Reverse Engineering* (pp. 90–99). IEEE.
- Polo, M., García-Rodríguez de Guzmán, I., & Piattini, M. (2007). An MDA-based approach for database re-engineering. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(6), 383–417. doi:10.1002/smre
- De Castro, V. (2007). *Aproximación MDA para el desarrollo orientado a servicios de sistemas de información Web: del modelo de negocio al modelo de composición de servicios Web*. Universidad Rey Juan Carlos.
- Khusidman, V., & Ulrich, W. (2007). *Architecture Driven Modernization: Transforming the enterprise*. Retrieved from <http://www.omg.org/docs/admtf/07-12-01.pdf>

- Whitworth, L. (2007). *Co-active coaching: New skills for coaching people toward success in work and life.* (D.-B. Publishing, Ed.).
- Souer, J., Honders, P., Versendaal, J., & Brinkkemper, S. (2007). Defining operations and maintenance in web engineering: A framework for CMS-based Web applications. In *2nd International Conference on Digital Information Management* (pp. 430–435). IEEE. doi:10.1109/ICDIM.2007.4444261
- Kitchenham, B., & Charters, S. (2007). Guidelines for performing systematic literature reviews in Software Engineering.
- Vara, J. M., Vela, B., Cavero, J. M., & Marcos, E. (2007). Model transformation for object relational database development. *The 22nd Annual ACM Symposium on Applied Computing*.
- Fleurey, F., Breton, E., Baudry, B., & Nicolas, A. (2007). Model-Driven Engineering for Software Migration in a Large Industrial Context. *Engineering*, 482–497.
- Kraus, A., Knapp, A., & Koch, N. (2007). Model-Driven Generation of Web Applications in UWE. *Model-Driven Web Engineering*, 261, 23–38.
- Canfora, G., & Penta, M. D. (2007). New frontiers of reverse engineering. In *Future of Software Engineering 2007*. IEEE Computer Society.
- Valverde, F., Valderas, P., & Fons, J. (2007). OOWS Suite : Un Entorno de desarrollo para Aplicaciones Web basado en MDA. In *CIBSE* (pp. 253–256).
- Acuña, C. J. (2007). *PISA-Arquitectura de integración de portales Web: un enfoque dirigido por modelos y basado en Servicios Web semánticos.* Universidad Rey Juan Carlos.
- García-Rodríguez de Guzmán, I. (2007). PRESSWEB : A Process to Reengineer Legacy Systems towards Web Services. In *14th Working Conference In Reverse Engineering* (pp. 285–288). IEEE.
- Souer, J., Weerd, I. Van De, Versendaal, J., & Brinkkemper, S. (2007). Situational Requirements Engineering for the Development of Content Management System-based Web Applications. *International Journal of Web Engineering and Technology*, 3(4), 420–440.
- Pino, F. J., García, F., & Piattini, M. (2007). Software process improvement in small and medium software enterprises: a systematic review. *Software Quality Journal*, 16(2), 237–261. doi:10.1007/s11219-007-9038-z
- Molina, F., Lucas, F. J., Toval, A., Vara, J. M., Cáceres, P., & Marcos, E. (2007). Towards quality web information systems through precise model driven development. *Handbook of research on web information systems quality*, Idea Group Publishing, 317–355.

- Rodrigues, A., Saraiva, J., Silva, R., & Martins, C. (2007). XIS – UML Profile for eXtreme Modeling Interactive Systems. *Technology*, (March 2007), 55 – 66.
- Garcia-Rodriguez de Guzman, I., Polo, M., & Piattini, M. (2006a). A Methodology for Database Reengineering to Web Services. In *Model Driven Architecture–Foundations and Applications* (pp. 226–240).
- García-Rodríguez de Guzmán, I., Polo, M., & Piattini, M. (2006). A Methodology for Database Reengineering to Web Services, 226–240.
- Castro, V. de, Marcos, E., & López-Sanz, M. (2006). A model driven method for service composition modelling: a case study. *International Journal of Web Engineering and Technology*, 2(4), 335–353.
- Weerd, I. Van De, Brinkkemper, S., Souer, J., & Versendaal, J. (2006). A Situational Implementation Method for Web-based Content Management System-applications: Method Engineering and Validation in Practice. *Software Process: Improvement and Practice*, 11(5), 521–538. doi:10.1002/spip
- Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., & Valduriez, P. (2006). ATL: a QVT-like transformation language. In *21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (pp. 719–720). Portland, Oregon, USA.
- Bozzon, A., Comai, S., Fraternali, P., & Toffetti, G. (2006a). Capturing RIA concepts in a web modeling language. In *International Worl Wide Web Conference* (pp. 907–908). Edinburgh, Scotland: ACM New York. doi:10.1145/1135777.1135938
- Bozzon, A., Comai, S., Fraternali, P., & Toffetti, G. (2006b). Conceptual Modeling and Code Generation for Rich Internet Applications. *International Conference on Web Engineering*. ACM New York. doi:10.1145/1145581.1145649
- Eclipse Foundation. (2006). Graphical Modeling Project (GMP). Retrieved from <http://www.eclipse.org/modeling/gmp/?project=gmf-notation#gmf-notation>
- Vela, B., Fernández-Medina, E., Marcos, E., & Piattini, M. (2006). Model driven development of secure XML databases. *ACM Sigmond Record*, 35(3), 22–27.
- Cáceres, P., de Castro, V., Vara, J. M., & Marcos, E. (2006). Model transformations for hypertext modeling on web information systems. *Proceedings of the 2006 ACM symposium on Applied computing - SAC '06*, 1232. doi:10.1145/1141277.1141567
- Schmidt, D. C. (2006). Model-Driven Engineering. *Computer*, 39(2), 25–31.
- Stahl, T., Volter, M., & Czarnecki, K. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons.

- Garcia-Rodriguez de Guzman, I., Polo, M., & Piattini, M. (2006b). Obtaining Web Services from Relational Databases. In *CSMR 2006. Proceedings of the 10th European Conference on Software Maintenance and Reengineering* (pp. 4–7).
- Guttman, M., & Parodi, J. (2006). Real-Life MDA: Solving Business Problems with Model Driven Architecture. Morgan Kaufmann.
- OMG. (2006). *The Meta Object Facility (MOF) Core Specification*. Retrieved from <http://www.omg.org/mof/>
- Jouault, F., & Kurtev, I. (2006). Transforming Models with ATL. In *Satellite Events at the MoDELS 2005 Conference*. (Vol. 3844, pp. 128–138). Springer Berlin / Heidelberg.
- Carmo, J. L. V. d. (2006). Web Content Management Systems: Experiences and Evaluations with the WebComfort Framework. *Instituto Superior Técnico*. Portugal.
- Kappel, G., Präßüll, B., Reich, S., & Retschitzegger, W. (2006). *Web Engineering: The Discipline of Systematic Development of Web Applications* (Wiley.). New York, NY.
- Kappel, G., Pröll, B., Reich, S., & Retschitzegger, W. (2006). *Web Engineering: the Discipline of Systematic Development of Web applications* (John Wiley.). New York, NY.
- Chinnici, R., Moreau, J.-J., Ryman, A., & Sanjiva, W. (2006). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. Retrieved from <http://www.w3.org/TR/wsdl20>
- German company Itemis. (2006). Xtext project. Retrieved March 12, 2013, from <http://www.eclipse.org/Xtext>
- Sjoeberg, D. I. K., Hannay, J. E., Hansen, O., Kampenes, V. B. Karahasanovic, A., Liborg, N. K., & Rekdal, A. C. (2005). A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering*, 31(9), 733–753.
- OMG. (2005a). Abstract Syntax Tree Metamodel specification of the OMG. Retrieved from <http://www.omg.org/spec/ASTM/1.0>
- Garcia-Rodriguez de Guzman, I., Polo, M., & Piattini, M. (2005). An integrated environment for reengineering. In *ICSM'05. Proceedings of the 21st IEEE International Conference on Software Maintenance* (pp. 165 – 174).
- Wimmer, M., & Kramler, G. (2005). Bridging Grammarware and Modelware. In J.-M. Bruel (Ed.), *MoDELS* (p. 159). Montego Bay, Jamaica: Springer.
- Sneed, H. M. (2005). Estimating the Costs of a Reengineering Project. In *12th Working Conference on Reverse Engineering*.

- Tratt, L. (2005). Model transformations and tool integration. *Software & Systems Modeling*, 4(2), 112–122.
- Noda, T., & Helwig, S. (2005). Rich Internet Applications. Technical Comparison and Case Studies of AJAX, Flash, and Java based RIA. Madison: UW E-Business Consortium.
- Biolchini, J., Mian, P. G., Candida, A., & Natali, C. (2005). Systematic Review in Software Engineering. *Engineering*, (May).
- OMG. (2005b). Unified Modelling Language 2. Retrieved from <http://www.omg.org/spec/UML/>
- Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4), 316–344. doi:10.1145/1118890.1118892
- Van Berkum, M., Brinkkemper, S., & Meyer, A. (2004). A Combined Runtime Environment and Web-based Development Environment for Web Application Engineering. In A. P. and J. Stirna (Ed.), *CAiSE* (pp. 307–321).
- Lee, S. C., & Shirani, A. I. (2004). A component based methodology for Web application development. *Journal of systems and software*, 1(2), 177 – 187.
- Castro, V. de, Marcos, E., & Cáceres, P. (2004). A user service oriented method to model Web information systems. In *WISE 2004* (pp. 41–52).
- Favre, J. (2004). Foundations of Model (Driven) (Reverse) Engineering : Models Episode I: Stories of The Fidus Papyrus and The Solarus. In *Dagstuhl Seminar on Model Driven Reverse Engineering*.
- Bézivin, J. (2004a). In search of a basic principle for model driven engineering. *Novatica Journal*, 2(Special Issue), 21–24.
- Bézivin, J. (2004b). In search of a basic principle for model driven engineering. *Novatica Journal*, 2(Special Issue), 21–24.
- Mellor, S., Scott, K., Uhl, A., & Weise, D. (2004). MDA Distilled. Paper presented at the Advances in Object-Oriented Information Systems. In *OOIS 2002 Workshops*. Montpellier, France.
- Zou, Y., Lau, T. ., Kontogiannis, K., Tong, T., & McKegney, R. (2004). Model-driven business process recovery. In *11th Working Conference on Reverse Engineering* (pp. 224–233). IEEE Comput. Soc. doi:10.1109/WCRE.2004.30
- Rugaber, S., & Stirewalt, K. (2004). Model-driven reverse engineering. *IEEE Software*, 21(4), 45–53. doi:10.1109/MS.2004.23
- Favre, J. (2004). Towards a Basic Theory to Model Driven Engineering. In *Workshop on Software Model Engineering, WISME 2004 joint event with UML2004*. Lisbon, Portugal.

- Pedro J. Molina. (2004). User interface generation with OlivaNova model execution system. In *9th international conference on Intelligent user interfaces* (pp. 358–359).
- Acerbis, R., Bongio, A., & et al. (2004). WebRatio, an innovative technology for Web application development. In *4th International conference on web engineering*. Munich, Germany.
- Hoick, J. (2003). 4 perspectives on Web information systems. In IEEE (Ed.), *36th Annual Hawaii International Conference on System Sciences* (p. 8).
- Polo, M., Piattini, M., & Ruiz, F. (2003). Advances in software maintenance management: Technologies and solutions. In Hershey (Ed.), . PA:Idea Group Publishing.
- Yin, R. K. (2003). Case Study Research: Design and Methods. *Sage Publications*, 5.
- Czarnecki, K., & Helsen, S. (2003). Classification of Model Transformation Approaches. In *2nd Workshop on Generative Techniques in the Context of MDA*. Anaheim, USA.
- Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., & Matera, M. (2003). *Designing Data Intensive Web Applications*. (Morgan Kaufmann, Ed.).
- Clements, P., Bachman, F., Bass, L., Garland, D., Ivers, J., Little, R., ... Stafford, J. (2003). *Documenting Software Architectures, Views and Beyond*. (A. Wesley, Ed.).
- Bianchi, A., Caivano, D., Marengo, V., & Visaggio, G. (2003). Iterative reengineering of legacy systems. *IEEE Transactions on Software Engineering*, 29(3), 225–241. doi:10.1109/TSE.2003.1183932
- Kleppe, A., & Warmer, J. (2003). *MDA Explained: the Model Driven Architecture: Practice and Promise*. Addison-Wesley.
- Miller, J., & Mukerji, J. (2003). *MDA Guide Version 1.0*. Retrieved from OMG document -ormsc/01-07-01
- Flore, F. (2003). MDA: The proof is in automating transformations between models. *OptimalJ White Paper*, 1–4.
- Kulkarni, V., & Reddy, S. (2003). Separation of Concerns in Model-Driven Development. *IEEE Software*, 20(5), 64–69.
- Selic, B. (2003). The programatics of Model-Driven development. *IEEE Software*, 20(5), 19–25.
- Ralyté, J., Deneckère, R., & Rolland, C. (2003). Towards a generic model for situational method engineering. In J. Eder & M. Missikoff (Eds.), *CAiSe 2003* (Vol. 2681, p. 1029). Heidelberg: Springer.

- Eriksson, H.-E., Penker, M., Lyons, B., & Fado, D. (2003). *UML 2 Toolkit*. John Wiley & Sons.
- McKeever, S. (2003). Understanding Web content management systems: evolution, lifecycle and market. *Industrial Management & Data Systems*, 103(9), 686–692.
- Frankel, D. (2002). *Model Driven Architecture: Applying MDA to Enterprise Computing*. New York, New York, USA: John Wiley & Sons.
- Ian Sommerville. (2002). *Software Engineering* (p. 712). Addison-Wesley.
- Gerber, A., Lawley, M., Raymond, K., Steel, J., & Wood, A. (2002). Transformation: The missing link of MDA. In A. Corradini, H. Ehrig, H. Kreowski, & G. Rozemberg (Eds.), in *Graph Transformation* (Springer B., Vol. 2505, pp. 90–105).
- Deshpande, Y., Murugesan, S., Ginige, A., Hansen, S., Schwabe, D., Gaedke, M., & White, B. (2002). Web Engineering. *Journal of Web Engineering*, 1, 3–17.
- Ginige, A. (2002). Web Engineering: Managing the Complexity of Web Systems Development. In *SEKE* (pp. 721–729).
- Lowe, D., & Henderson-Sellers, B. (2001). Characteristics of web development processes. In *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*.
- Corporation, B. G. (2001). *Content Management Report*. Retrieved from <http://ovum.com/section/home/>
- Ektron, & NH. (2001). Effective Web Content Management Empowering the Business user while IT Maintains control. Ektron.
- Baresi, L., Garzotto, F., & Paolini, P. (2001). Extending UML for modeling web applications. In *34th Annual Hawaii International Conference on System Sciences*. Los Alamitos, CA, USA: IEEE Computer Society.
- OMG. (2001). *Model Driven Architecture. A technical perspective*.
- Pfleeger, S. L., & Kitchenham, B. A. (2001). Principles of survey research: part 1: turning lemons into lemonade. *SIGSOFT Softw. Eng. Notes*, 26(6), 16–18.
- Ginige, A., & Murugesan, S. (2001a). The Essence of Web Engineering - Managing the Diversity and Complexity of Web Application Development. *IEEE Multimedia*, 8(2), 22–25.
- Boiko, B. (2001). Understanding Content Management. *Bulletin of the American Society for Information Science and Technology*, 28(1), 8–13. doi:10.1002/bult.221

- Murugesan, S., Deshpande, Y., Hansen, S., & Ginige, A. (2001). Web Engineering: a New Discipline for Development of Web-Based Systems. *web Engineering*, 3–13.
- Ginige, A., & Murugesan, S. (2001b). Web Engineering: An introduction. *MultiMedia, IEEE*, 8(1), 14–18.
- Deshpande, Y., & Hansen, S. (2001). Web engineering: creating a discipline among disciplines. *IEEE Multimedia*, 8(2), 82–87. doi:10.1109/93.917974
- Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). *Web services description language (WSDL) 1.1*.
- Gnaho, C. (2001). Web-Based Information Systems Development—A User Centered Engineering Approach. In *Lecture Notes in Computer Science 2016* (pp. 105–118). Springer Berlin: Heidelberg.
- Hay, D., Healy, K., & Hall, J. (2000). *Defining Business Rules - what are they really?*
- Gómez, J., Cachero, C., & Pastor, O. (2000). Extending a conceptual modelling approach to web application design. In Springer-Verlag (Ed.), *International Conference on Advanced Information Systems Engineering* (pp. 79–93).
- Müller, H. A., Jahnke, J. H., Smith, D. B., Storey, M. A., Tilley, S. R., & Wong, K. (2000). Reverse Engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering* (pp. 47–60). Limerick, Ireland: ACM.
- Bennett, K. H., & Rajlich, V. T. (2000). Software maintenance and evolution: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*. Limerick, Ireland: ACM.
- Jacobson, I., Booch, G., & Rumbaugh, J. (2000). *The Unified Software Development Process* (p. 464).
- Ceri, S., Fraternali, P., & Bongio, A. (2000). Web Modeling Language (WebML): A modeling language for designing web sites. *Computer Networks*, 33, 137–157.
- Piero Fraternali. (2000). Web Ratio. Retrieved from [www.webratio.com](http://www.webratio.com)
- Overmyer, S. P. (2000). What's different about requirements engineering for web sites=. *Requirements Engineering*, 5(1), 62–65.
- Turban, E., McLean, E., & Wetherbe, J. (1999). *Information Technology for Management* (2nd editio.). New York, NY: John Wiley and Sons: New York.
- Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The Unified Modeling Language User Guide* (p. 464). Addison Wesley Longman.

- Marcos, E., & Marcos, A. (1998). An Aristotelian Approach to the Methodological Research: A Method for Data Models Construction. In L. Brooks & C. Krimble (Eds.), *Information Systems-The Next Generation* (pp. 532–543). Mc Graw-Hill.
- Schwabe, D., & Rossi, G. (1998). An Object Oriented Approach to Web-Based Application Design. *Theory and Practice of Object Systems*, 207–225.
- Lehman, M. M. (1998). Implications of evolution metrics on software maintenance. In *International Conference on Software Maintenance* (pp. 208–217). doi:10.1109/ICSM.1998.738510
- Kazman, R., Woods, S. G., & Carrière, S. J. (1998). Requirements for integrating software architecture and reengineering models. In *Proceedings of the Working conference on Reverse Engineering (WCRE-98)* (pp. 154–163). IEEE Computer Society.
- Isakowitz, T., Bieber, M., & Vitali, F. (1998). Web Information Systems. *Communications of the ACM*, 41(7), 78–80.
- Berson, A., & Smith, S. J. (1997). *Data warehousing, data mining and OLAP*. Mc Graw-Hill.
- Herbsleb, J., Zubrow, D., Goldenson, D., Hayes, W., & Pault, M. (1997). Software quality and the Capability Maturity Model. *Commun. ACM*, 40(6), 30–40.
- ISO/IEC. (1996). ISO/IEC 14977:1996 - EBNF. Retrieved February 02, 2013, from [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=26153](http://www.iso.org/iso/catalogue_detail.htm?csnumber=26153)
- Kelly, S., Lyytinen, K., & Rossi, M. (1996). MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In *CAiSE* (pp. 1–21). London: Springer Berlin / Heidelberg.
- Pressman, R. S. (1996). *Software Engineering: A Practitioner's Approach* (4th editio.). Higher, McGraw-Hill Education.
- Basili, V. R. (1996). The role of experimentation in software engineering: past, current, and future. In *18th international conference on Software engineering* (pp. 442–449). Berlin, Germany.
- Stake, R. E. (1995). *The art of case study research*.
- Schwabe, D., & Rossi, G. (1995). The Object Oriented Hypermedia Design Model. *Communications of ACM*, 38, 45–46.
- Arnold, R. (1993). Software Reengineering. *IEEE Computer Society Press*.
- Dorling, A. (1993). SPICE: Software Process Improvement and Capability Determination. *Software Quality Journal*, 2(4), 209–224.
- Kunii, H. S. (1990). Data Manipulation Language. In *Graph Data Model* (pp. 29–39).

- Chikofsky, E. J., & Cross, J. H. (1990). Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, 7(1), 13–17. doi:10.1109/52.43044
- Eisenhardt, K. M. (1989). Building Theories from Case Study Research. *The Academy of Management Review*, 14(4), 532–550.
- Biggerstaff, T. J. (1989). Design recovery for maintenance and reuse. *Computer*, 22(7), 36–49.
- OMG. (1989). Object Management Group. Retrieved from <http://www.omg.org/>
- Basili, V. R., Selby, R. W., & Hutchens, D. H. (1986a). Experimentation in software engineering. *IEEE Transactions on Software Engineering*, 12(7), 733–743.
- Basili, V. R., Selby, R. W., & Hutchens, D. H. (1986b). Experimentation on Software Engineering. *IEEE Transactions on Software Engineering*, 12(7), 733–743.
- IEEE. (1983). Std 729-1983.
- Bunge, M. (1979). *La investigacion científica*. Barcelona: Ariel.
- Dijkstra, E. W. (1976). *A Discipline of Programming*. (NJ:Prentice Hall, Ed.). Englewood Cliffs.
- Parnas, D. L. (1972). On the criteria to be used in Decomposing Systems into Modules. *Communications of ACM*, 15(2), 1023–1058.
- Wagner, C. (1959). MODEL-DRIVEN REENGINEERING AND SOFTWARE MIGRATION.
- Kish, L. (1959). *Some statistical problems in research design* (pp. 328–338).

## *Acronyms*

---



## Table of Acronyms

<b>ACRONYM</b>	<b>DESCRIPTION</b>
ADM	Architecture Driven Modernization
API	Application Programming Interface
ASTM	Abstract Syntax Tree Model
ATL	ATLAS Transformation Language
CIM	Computation Independent Model
CMS	Content Management System
DSL	Domain Specific Language
EMF	Eclipse Modelling Framework
GMF	Generic Modelling Framework
IS	Information System
KDM	Knowledge Discover Metamodel
M2M	Model to Model
MDA	Model-Driven Architecture
MDSD	Model-Driven Software Development
MDE	Model-Driven Engineering
MDRE	Model-Driven Reengineering
MOF	Meta-Object Facility
OCL	Object Constraint Language
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query/View/Transformation
SBVR	Semantics Of Business VocabularyAnd Rules

SQL	Structured Query Language
URI	Universal Resource Identifier
UML	Unified Modelling Language
XML	eXtensible Markup Language