Universidad Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Departamento de Lenguajes y Sistemas Informáticos II

ArchiMeDeS: A Service-Oriented Framework for Model-Driven Development of Software Architectures

Doctoral Thesis

by Marcos López Sanz

Thesis Supervisor: Dr. Esperanza Marcos Martínez

Thesis Co-Advisor: Dr. Carlos E. Cuesta Quintero

February 2011

Universidad Rey Juan Carlos

La Dra. Dª. Esperanza Marcos Martínez, Catedrática de Universidad, y el Dr. D. Carlos E. Cuesta Quintero, Profesor Titular Interino, ambos del Departamento de Lenguajes y Sistemas Informáticos II de la Universidad Rey Juan Carlos de Madrid, directora y co-director, respectivamente, de la Tesis Doctoral: "*ArchiMeDeS*: A SERVICE-ORIENTED FRAMEWORK FOR MODEL-DRIVEN DEVELOPMENT OF SOFTWARE ARCHITECTURES" realizada por el doctorando D. Marcos López Sanz,

HACEN CONSTAR QUE:

Esta tesis doctoral reúne los requisitos para su defensa y aprobación.

En Madrid, a 25 de febrero de 2011

Fdo.: Esperanza Marcos Martínez

Fdo.: Carlos E. Cuesta Quintero

Entia non sunt multiplicanda praeter necessitatem [1]
(William of Ockham, c. 1288 – c. 1348)

Quod est inferius, est sicut quod est superius,
et quod est superius est sicut quod est inferius,
ad perpetranda miracula rei unius [2]
(Tabula Smaragdina, Hermes Trismegistus)

---

[1] *Entities must not be multiplied beyond necessity.*

[2] *That which is below is as that which is above, and that which is above is as that which is below, to perform the miracles of the one thing.*

RESUMEN

La dependencia de la industria en las tecnologías de la información se ha acentuado en los últimos años conforme lo han hecho sus necesidades de buscar nuevas fórmulas de negocio. Esto se debe, principalmente, a un cambio de tendencia hacia nuevos modelos de negocio basados en la externalización o el posicionamiento online. Desde el ámbito de las tecnologías de la información han surgido nuevos paradigmas de computación e iniciativas de desarrollo con el fin de ayudar en estos nuevos escenarios. La principal línea de actuación se centra en ofrecer una **solución para conseguir un alineamiento entre las entidades involucradas en la provisión de servicios de negocio y aquellas tecnologías y plataformas que soporten sus requisitos de negocio**. En este sentido, dos iniciativas han demostrado ser de gran ayuda: la **aproximación MDA**, como forma de abordar la especificación de metodologías y estrategias de desarrollo de software, y la **orientación a servicios**, como paradigma de computación. En los últimos años estos dos elementos han constituido la base de multitud de iniciativas de investigación, proyectos y propuestas enfocadas a solucionar el problema citado anteriormente. Por otro lado, la **Arquitectura**, como artefacto central de cualquier solución software, se considera, por lo general, el elemento adecuado sobre el que los principios de la orientación a servicios pueden aplicarse mejor. De igual forma, en los últimos años el uso de estrategias dirigidas por modelos para su especificación constituye una aproximación que se ha demostrado válida dentro del área de investigación de las arquitecturas software.

Esta Tesis se centra en agrupar los tres aspectos mencionados con anterioridad: tiene como objetivo la **definición de un marco dirigido por modelos para la especificación de arquitecturas software que tengan como base los principios de la orientación a servicios**. El marco descrito, bajo el nombre de *ArchiMeDeS*, representa una solución coherente para definir arquitecturas que reduce el salto existente entre las configuraciones de alto nivel de un sistema, entendido desde un punto de vista organizativo y de negocio, y su representación de bajo nivel, donde los aspectos tecnológicos determinan el aspecto final del sistema ofreciendo un soporte tecnológico a los procesos de negocio y restricciones aplicables previamente identificados.

Para alcanzar este objetivo, *ArchiMeDeS* presenta una solución integrada que sigue los principios de MDA para la especificación de arquitecturas software. Consecuentemente, y siguiendo la separación en niveles de abstracción promovida

por MDA, se han creado varios **DSLs para la representación de Arquitecturas Software**: un DSL a nivel PIM que permite la especificación conceptual de arquitecturas basadas en servicios y diferentes DSLs en el nivel PSM. El modelado específico de la plataforma (PSM) ha sido dividido en dos subniveles con el fin de reflejar, separadamente, los aspectos comunes a cualquier plataforma de servicios por un lado y, por otro lado, las particularidades de las tecnologías concretas de servicios, para lo cual se ha definido un DSL para cada aproximación tecnológica escogida.

La diferenciación en niveles PIM y PSM para la especificación de la arquitectura permite aislar las características más abstractas y conceptuales de la arquitectura del sistema (nivel PIM) de aquellos aspectos específicos de la plataforma de ejecución o tecnología de implementación del sistema (nivel PSM). Además, el modelado a diferentes niveles de abstracción permite la inclusión de información adicional referida a decisiones arquitectónicas tanto en uno como en otro nivel. A nivel PIM, por ejemplo, se incorpora en los modelos arquitectónicos información referente a la utilización de **estilos arquitectónicos**. Asimismo, y como se ha indicado anteriormente, el nivel PSM se ha dividido en dos subniveles: PDM, que congrega todos aquellos conceptos que comparten las plataformas de ejecución de servicios: y TDM, que contendrá extensiones a este modelo PDM particularizadas con las características concretas de las tecnologías de servicios basadas en Servicios Web, Servicios Grid y Servicios REST.

En cada caso, tanto a nivel PIM como PSM, se han detallado la semántica (entendida como el conjunto de conceptos que definen la sintaxis abstracta del lenguaje) y la sintaxis concreta. Para la semántica, los conceptos utilizados son aquéllos provenientes del paradigma SOC. Como consecuencia directa, las arquitecturas software modeladas se convierten en artefactos orientados a servicios, beneficiándose de esta manera de sus propiedades a la hora de representar las características estructurales de soluciones software que soporten procesos de negocio concretos. Para llegar a tal fin, los conceptos han sido definidos dentro de **metamodelos** que permiten la definición de modelos arquitectónicos de servicios. Por otro lado, la sintaxis concreta de cada uno de los DSL definidos se basa en el uso de **UML como notación gráfica**. El fin último es proveer a cada DSL de una notación específica, adaptable, fácil de comprender a simple vista y que esté ampliamente extendida y aceptada como notación para representar artefactos software. Además, la utilización de UML permite apoyarse en sus capacidades de extensión para la definición de modelos gráficos personalizados mediante **perfiles UML.**

Con el fin de mejorar los modelos arquitectónicos e incorporar información adicional (como la que puede aportar la selección de un estilo arquitectónico determinado), se ha definido un conjunto de **transformaciones entre modelos**. Por un lado, se ha definido un proceso de fusión entre los modelos de nivel PIM y aquéllos conteniendo la información sobre estilos arquitectónicos. Para ello se han utilizado modelos de *weaving* y transformaciones definidas en ATL. Por otro lado, y con el fin de adaptar la información arquitectónica conceptual recogida a nivel PIM a las peculiaridades de las plataformas de servicios, se han definido reglas de transformación, también con ATL, entre los metamodelos de nivel PIM y cada uno de los metamodelos definidos a nivel PSM para las plataformas anteriormente citadas.

Además de la definición de los metamodelos a diferente nivel de abstracción (PIM, PDM, TDM) y de transformaciones entre ellos (PIM-a-PIM, PIM-a-PDM/PIM-a-TDM), se ha definido un conjunto de **herramientas de modelado** como parte de *ArchiMeDeS*. Este conjunto de herramientas permiten la edición gráfica de los modelos arquitectónicos, la validación de los mismos de acuerdo a sus correspondientes metamodelos y la ejecución de las transformaciones de modelos definidas entre ellos. Para su implementación se ha utilizado Eclipse como plataforma de desarrollo. Además, se han utilizado diversas extensiones, específicas para el desarrollo dirigido por modelos en Eclipse, tales como EMF, GMF o aquéllas que soportan la definición y ejecución de transformaciones de modelos en ATL. En conjunción con estas extensiones se han utilizado también otros entornos de modelado especializados en la definición de procesos de *weaving* (como AMW). Finalmente, *ArchiMeDeS* ha sido validado y refinado mediante su aplicación a la **especificación de la arquitectura de diferentes casos de estudio**: un sistema para la gestión de imágenes médicas, una pasarela para el envío de SMS y la emulación de una situación de juego concreta en un partido de baloncesto como metáfora para el estudio de coreografías de servicios.

ABSTRACT

The reliance of companies on IT has been rocketed in parallel to their search towards new business models. This is mainly due to the diversification of businesses towards new formulas based on externalization or online positioning. To assist in this scenario, new computing paradigms and development initiatives have come up. The main line of attack to give support to all the needs detected by companies focuses on providing with **a solution to the alignment between the entities involved in business service provisioning and the technologies and platforms** that support their business requirements. In this direction, two initiatives have proved to be of great help: **the MDA approach,** as a way to tackle modern software development strategies and methodologies; and **the SOC paradigm,** as computing paradigm. In the last few years these two elements have been part of a myriad of research initiatives, projects and proposals targeting the abovementioned problem. In that sense, the **Architecture**, as central artefact of any software solution, is considered, by and large, as the adequate element of choice over which the principles of service-orientation may take big advantage. Similarly, the use of model-driven approaches for its specification constitutes an improvement that has been proved valid among the Software Architecture research area in the last few years.

This Doctoral Thesis is devoted to put together the previous three aspects: it **aims at the definition of a framework for the Model-Driven specification of Software Architectures that use the concepts behind the Service-Orientation** for its definition. The framework described, named *ArchiMeDeS*, represents a coherent solution for architecting the existing gap between high-level configuration of a system, describing the business entities and relationships required by the system solution, and its low-level representation, where the technological aspects determine the final shape of the system providing technical support to the previously identified business processes and constraints.

To reach that goal, *ArchiMeDeS* presents an integrated solution that follows the MDA approach for the specification of Software Architectures. Accordingly, and following the separation in abstraction levels fostered by MDA, **several DSLs for representing Software Architectures have been created:** a DSL at PIM level, allowing for the specification of conceptual service architectures, and different DSLs at PSM level. The platform specific modelling (PSM) has been divided in two in order to reflect, separately, the commonalities of

service platforms and also the particularities of concrete service technologies, for which an independent DSL has been designed regarding each technological chosen approach.

The differentiation in PIM and PSM levels for architecture specification allows the isolation of the conceptual features of the system architecture (at PIM level) from those specific of a concrete execution platform or implementation technology (at PSM level). In addition, modelling at different abstraction levels allows the inclusion of additional information within the models. In the case of *ArchiMeDeS*, it allows adding information from design decisions into PIM models in the form of **architectural styles**. Moreover, and as it was pointed out previously, the PSM level has been divided in two different sublevels: PDM (*Platform Dependent Model*), which gathers all the concepts that the target service-oriented platforms have in common; and TDM (*Technology Dependent Model*), which will comprise extensions to the PDM DSL adapted to the features of the service technologies based on Web Services, Grid Services and REST Services.

In each case, either at PIM or PSM level, both the semantics (set of concepts building up the abstract syntax) and the concrete syntax have been defined. For the semantics, the concepts used are those contained within the SOC paradigm. As a consequence, the Software Architectures modelled are transformed into Service-Oriented artefacts, thus taking advantage of its features to represent the structural characteristics of a software solution supporting the required business processes. To do so, the concepts are gathered in metamodels that allow the definition of service-oriented architectural models. On the other hand, the concrete syntax of each DSL is represented by means of a **UML-based notation**. The goal is to provide with a specific, adaptable and widely accepted notation for the representation of software artefacts. Moreover, by using UML it is possible to benefit from its extension mechanisms for the definition of personalized graphical models through the definition of **UML profiles**.

With the purpose of improving the architectural models and being able to include additional information (such as that of architectural styles), several **model transformations** have been defined. On the one hand, a merge process between PIM models and that of the architectural styles have been defined. For that aim, *weaving* models and ATL transformation rules are used. On the other hand, and in order to adapt the architectural information gathered in PIM models to the particularities of service platforms, transformation rules have been specified, also with ATL, between the PIM metamodel and every metamodel of the PSM level for each of the abovementioned service platforms.

Apart from the definition of metamodels at different abstraction levels (PIM, PDM, TDM) and transformations among them (PIM-to-PIM, PIM-to-PDM/PIM-to-TDM), the *ArchiMeDeS* framework includes a set of modelling tools. This toolkit **allows the graphical editing of models, the validation of their conformance according to their metamodels and also the execution of the transformations among models**. This toolkit is based on the features offered by the Eclipse platform. Additionally, some extensions for that platform, specifically created for modelling purposes with Eclipse, such as EMF or GMF or those supporting the definition and execution of model transformations in ATL have been used. Together with these extensions, the *ArchiMeDeS* toolkit makes use of other specialized modelling frameworks, such as AMW, for the definition of weaving processes.

Finally, the *ArchiMeDeS* framework has been validated and refined by means of its application to the **architectural specification of several case studies**: a system for the medical digital image management, a gateway for sending SMS and the emulation of a concrete game situation in basketball used to study service coordination based on choreographies.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and thankfulness towards my supervisor, Esperanza Marcos, for her continuous support and wonderful pieces of advice given since I began researching and working with her, not only in the scope of the Thesis but also in other aspects of my academic career at the URJC. At your side I have learned many lessons about professionalism, partnership, perseverance and, of course, life. Thanks for your patience and immeasurable guidance. To my co-advisor, Carlos E. Cuesta, acknowledge him those marvellous moments discussing, arguing, talking endlessly… no time was wasted at all when thinking and rethinking even about the minor detail or topics that were sometimes completely unrelated ☺. Thanks for being there and become my friend in hard times.

I would also like to acknowledge Prof. Flavio Oquendo, leader of the VALORIA research group, for allowing me to work at his valuable side. I would also like to give special thanks to Zawar Qayyum, for let me sharing with him those moments I spent there, researching side to side at Vannes.

I take this opportunity to express my profound sense of gratitude and respect to all those who helped me throughout the duration of making this dissertation. To the members of the *Kybele* group, the present and the gone ones. In particular, to Juancho and Vero, who perfectly understood the stress of writing and working with the Thesis as they have recently suffered the same process. To Valeria, for her perpetual joyfulness and guidance in services and case studies. To Diana, my office partner and the most optimistic person I know. To Belén, Paloma, Josemari, Javier, your expertise has always been an example to follow. To Feliu, Elisa, Iván, Álvaro, and other fellows without whom this Thesis would never have been possible.

I acknowledge the effort of those PFC students who have contributed significantly to my work. Like José Manuel Arévalo, that helped me with Web Services and other related technologies; Juan Manuel Ramón, for its dedication and patience with the Grid platform; Santiago Moronta, for his priceless help with the toolkit implementation. Let these paragraphs serve to recognize all your efforts.

I also give thanks to my family. To my parents, Pedro Pablo and Pilar, for being the source of all my scientific curiosity and model to follow in life. To my brother, Rubén, for being there whenever I need. To my grandparents, wherever they are, for all the good things and moments we lived together.

Finally, if there has been a reason to start, stop and follow during all these years it has been you, Erika. My life wouldn't be the same without having known you, now long time ago, in a cold summer night. People need religion, love, health and other things to live and survive… I have *you*. Thanks for being my life companion.

En primer lugar, me gustaría expresar mi más sincera gratitud y reconocimiento hacia mi directora de Tesis, Esperanza Marcos, por su continuo apoyo y los valiosos consejos recibidos desde que empecé a investigar y a trabajar a su lado, no sólo a nivel intelectual sino también en otros aspectos de mi carrera académica dentro de la URJC. A tu lado, he aprendido incontables lecciones acerca de la profesionalidad, el compañerismo, el tesón y, por supuesto, la vida. Gracias por tu paciencia y tu inestimable guía durante estos años. A mi co-director, Carlos E. Cuesta, agradecerle esos momentos maravillosos que hemos pasado discutiendo, debatiendo, cuestionando cualquier cosa, hablando interminablemente… no considero perdido ni un solo segundo hablando contigo, pensando y repensando hasta el más ínfimo detalle que nos poníamos a repasar, incluso en aquellas veces que tratábamos temas que nada tenían que ver con la Tesis ☺. Gracias por estar ahí y convertirte en mi amigo en los momentos más difíciles.

Quiero mostrar mi agradecimiento al Profesor Flavio Oquendo, director del grupo de investigación VALORIA, por permitirme trabajar a su lado. También me gustaría dar especialmente las gracias a Zawar Qayyum, por compartir aquellos momentos que pasé en Vannes investigando codo con codo con él.

Aprovecho esta oportunidad para expresar mi más profundo sentido de gratitud y respeto a todos aquellos que me han ayudado durante todo el tiempo de la Tesis. A los miembros de Kybele, los actuales y los que lo fueron en el pasado. En particular a Juancho y a Vero, con los que me siento especialmente identificado pues sé que pasaron por los mismos momentos difíciles de la Tesis no hace demasiado tiempo. Su experiencia y consejos me han servido para llevar a cabo esta labor. A Valeria por su perenne alegría y sus conocimientos de servicios y apoyo en la parte de casos de estudio. A Diana, mi compañera de despacho, sin cuyo optimismo perpetuo no habría podido continuar. A Belén, Paloma, Josemari,

Javier, vuestra experiencia siempre ha sido un ejemplo a seguir. A Feliu, Elisa, Iván y otros compañeros de trabajo sin los cuales esta Tesis tampoco habría sido posible.

Agradezco enormemente el esfuerzo de todos los proyectandos que han contribuido significativamente al trabajo que aquí presento. Estudiantes como José Manuel Arévalo, que me ayudó con los Servicios Web y otras tecnologías relacionadas; a Juan Manuel Ramón, por su dedicación y paciencia con la plataforma Grid; a Santiago Moronta, por su inestimable ayuda con la implementación de la herramienta. Que estos párrafos sirvan para reconocer todos vuestros esfuerzos.

Quiero igualmente dar las gracias a toda mi familia. A mis padres, Pedro Pablo y Pilar, por ser el origen de mi curiosidad científica y el modelo a seguir en la vida. A mi hermano, Rubén, porque sé que está ahí cuando le necesito. A mis abuelos, dondequiera que estén, gracias por los buenos ratos que compartimos juntos.

Y, finalmente, si ha habido una sola razón para empezar, parar y proseguir durante todos estos años, esa has sido tú, Erika. Mi vida no sería lo mismo sin haberte conocido, ahora ya hace mucho tiempo, en una fría noche de verano. La gente necesita religiones, amor, salud y otras cosas para vivir y sobrevivir… yo te tengo *a ti*. Gracias por ser mi compañera en la vida.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1:
Introduction

The work comprised in this Doctoral Thesis tackles the development of Software Architectures from a Service-Oriented perspective and using a Model-Driven approach. To face this issue, it is proposed to develop a modelling framework allowing the specification of Software Architectures built upon the principles of the *Service-Oriented Computing* paradigm (SOC, [182]) and following the *Model-Driven Architecture* (MDA, [171]) proposal.

This section describes the motivation that leads to present this Doctoral Thesis, the direction followed to address the challenges posed as well as the main hypothesis and objectives that this work aims to cover. Later in this chapter, the research framework, in which the presented work is framed, and the research method used are also explained in detail. This introductive chapter ends with the depiction of the global structure of the current dissertation.

1.1 Motivation

It is widely recognized that during the last decades information technologies have impregnated the inner structures of businesses. The direct consequence has arrived in the form of a dangerous dependency acquired by companies on information systems. This dependence refers not only to hardware resources and utilities but also, and possibly more importantly, to an increasing reliance on software systems. The evolution of software technologies, in that sense, has steered a bidirectional influence between businesses and software. Perhaps the most clarifying example is *the Internet* global network understood as source and target of knowledge and resources. On the one hand, it has rocketed the coming up of a new wave of enterprises and companies based completely on the Web and related online facilities. On the other hand, it has posed many technological challenges (enhanced performance, high availability, optimal scalability, standardization needs, integration environments, etc.) that IT researchers are required to solve efficiently to support new business models. This shift of attention towards the online universe implies, from the point of view of software research, the necessity to design not only new software solutions but also the specification of novel development methodologies, execution platforms, support tools and design and management strategies that, by and large, differ from the traditional ones. The Internet establishes developing conditions in which current engineering techniques are inappropriate or, at least, not enough to deal with the new needs. Moreover, in an era of economical adjustment, it is important that the research approaches defined for that context may spread easily to other environments, somehow away from the Web, thus increasing the importance of

relying on the key aspects of IT (think about Software Architecture for example) easing the development of highly flexible solutions.

This PhD dissertation is devoted to cover part of these issues leaning on three main pillars: the *Model-Driven Development* approach, to face the recent software engineering needs; the *Service-Oriented Computing* paradigm, to gather all the concepts used within the new business styles and as technological foundation; and last but not least, the *Software Architecture*, as core artefact during the development of information systems.

Next subsections try to explain in detail the main reasons that have motivated their selection, outlining the research scope in which this Thesis is framed.

1.1.1 Model-Driven Development to tackle new Software Engineering needs

History of Software Engineering shows that the existence of concrete strategies, processes and methodologies to develop software solutions ends up in an improvement of aspects such as quality, maintenance, robustness, scalability, etc. [193]. Traditionally, Software Engineering techniques have been based on the conceptualization of the system features to be developed, abstracting both the context of the solution and its environment. With that premise in mind, it seems quite obvious to think that the strategy to follow, in order to give the right solution to the challenges posed by current business trends, should be also aligned with that approach. In that sense, the use of diagrammatic reasoning, i.e. through the specification of *models*, for the whole software lifecycle, seems to be the right path to follow to settle down the principles of modern Software Engineering.

During the last years, the *Model-Driven Development* (MDD) approach [94] has grown in importance to currently become one of the most successful strategies for the development of new generation information systems. Although the possibility to represent the characteristics of a system with models and diagrams is known since the beginning of Software Engineering, the establishment of several levels of models, from different points of view and, more specifically, the ability to define transformation rules applied to models, are considered to be among the main reasons of its today success. In addition, the concepts behind the definition of *Domain-Specific Languages* (DSLs) associated to those models also convert the MDD approach in a handy alternative for conducting Software Engineering nowadays.

Among the multiple existing proposals founded on the model-driven principles there is one that stands out over the others. The *Model-Driven*

Architecture (MDA) initiative, released by the OMG Consortium in 2001 [171], has attracted the attention of both academic research groups and enterprise research divisions. This is easily corroborated by the large amount of initiatives that have appeared in the last few years following this approach. MDA, apart from considering the model as the main artefact in the definition of software development processes, suggests a separation of types of models grouped in different abstraction levels. These levels range from the modelling of concerns related to the business aspect of the system under development (*Computation Independent Model*, CIM) to those models gathering all the technological aspects of the system implementation (*Platform Specific Model*, PSM). However, the main contribution of the MDA proposal to the system development field does not lie solely in this model classification but, more importantly, on the possibility of defining (semi-)automatic transformations among the concepts specified in metamodels. These *metamodels* are models that describe the concepts used in the models that conform to them [200]. Transformation rules facilitate, in that sense, a much easier progression in system development. Eventually, models that are closer to the implementation level must serve as origin of the system source code, which should be obtained from these models automatically (or, at least, semi-automatically).

Despite the well-known benefits of using the MDA proposal, several deficiencies have been detected lately. Recent works [136][150] argue that one of the main drawbacks of MDA is the lack of a precise definition of the role the architectural viewpoint must play. In its foundational specification, there is no explicit mention to the Architecture within the model structure of an MDA-based development process. Many of the current methods and methodologies following the MDA approach avoid modelling explicitly the architectural aspect of the system. They mix the topological design of the system (structure in the form of components, connectors and relations among them) with the definition of its functionalities (behaviour given to each of the elements playing a concrete role during the execution of the system itself) [16][95]. Moreover, it is worth mentioning that the lack of a precise specification of the Architecture, as independent model within the model architecture, constraints the flexibility of the development process itself. An illustrative example can be found in the application of different architectural styles as a way to cope with environments in which changes in business requirements occur frequently. Knowing the benefits of applying architectural styles during the design and development of software [208] not including the architectural models in that process obliges to fix, by default, a concrete architectural style. In spite of the advantages of the MDA proposal, this

circumstance is considered as a strong restriction when building software solutions able to evolve in order to adapt themselves to new requirements or to changes in the system environment.

All this given, **MDA represents the first pillar of the current Thesis from a dual point of view**: a) it will be considered as the engineering approach of choice for a framework to specify Software Architectures; and b) the process of specification of the Software Architecture itself will serve to fill in the gap of current MDA-based methodologies, by covering the architectural viewpoint of the Software development process.

1.1.2 The Service-Oriented Paradigm as Computational Foundation of Novel Engineering Processes

The need for developing systems that adapt (or, ideally, *self-adapt*) to changes in the requirements is much noticeable when having a look at the direction the global Economy has taken in the last few years. The externalization and fragmentation of the industrial processes has favoured the adoption of business models based on *Services*, understood as independent economical entities with a business value and functional asset for both the enterprise and the end consumer [114]. As a consequence, it has evidenced the need for describing software development processes supporting the whole software lifecycle with that business background, in which developing software systems whose structure and behaviour is susceptible to change, is crucial. In that context, MDD approaches have proved to provide a way to create adequate processes based on those requirements, aligning the modelling of the changing business needs with current technologies. However, in order to address these new development process types and needs, an appropriate underlying computing paradigm should be used. From a technological point of view, that new business model has had a direct response in the form of the *Service-Oriented Computing* (SOC) paradigm [4][182] and its related standards and languages [165][236]. By using the principles of SOC it is possible to create loosely-coupled and dynamic systems that adapt perfectly to the forthcoming business needs.

However, the correspondence between higher-level services (as integral parts of a financial and economical procedure) and their implementation by means of current service technologies is not easy and requires a transformation process that is far from being trivial. Nowadays, the SOC paradigm represents an important change in which software is *analyzed* (for example due to the need of including business dependencies, constraints and policies), *designed* (the numerous existent standards, for instance, implies the selection of the most

appropriate for each solution), *built* (for example, the restrictions posed by execution platforms of choice affect the technological support), *deployed* (because of the inherent need for synchronism in distributed environments) and *used* (the concepts behind SOC must coexist with other technological approaches). All of this has led researchers and developers to rethink the development techniques used to construct information systems based on this paradigm. In that sense, model-driven approaches again (and the MDA proposal in particular), help to fill in the existing gap between the business services specifications and the development of service-oriented information systems.

The use of development approaches based on models (MDD) together with the SOC paradigm has long proved to be valuable in the last few years. The high number of European projects devoted to that topic [62][62], in which many of the biggest worldwide enterprises have been involved, supports this statement. As it has been previously explained, MDA is a proposal in which the model is considered as the main software artefact during the software development process. The definition of several models grouped in different abstraction levels and the specification of model transformation rules convert this approach in one of the best alternatives when deciding the way service-oriented software solutions should be developed. Indeed, because of the existence of a hierarchical model specification, the use of MDA eases the transition from high-level services to their technological counterparts, besides facilitating platform migration and enhancing the system adaptability.

The way of organizing the infrastructures and applications in a set of interactive services is usually known as *Service-Oriented Architecture* (SOA) [59]. Due to its dynamic and loosely coupled nature, the application of the SOC paradigm to software development has a direct impact in the architectural aspect. It is not only necessary to define the topological structure of the system and its constituent elements, its inner composition structures and existent relationships, but also how it evolves throughout its entire lifecycle.

The SOC paradigm, and its architectural complement, SOA, represent the second pillar that supports this Thesis. These aspects are considered, again, **dually**: a) SOC is used as foundational basis of the models of the Architecture, which means that the concept around which the architectural models are built is the *service;* and, as direct consequence of the latter, b) the architectures specified allow the modelling of systems that use service-oriented technologies as target platforms. However, since the research work is also framed in MDA, which advocates for a separation of models in levels according to the inclusion or avoidance of the platform characteristics on them, the defined framework should

be capable to support the selection of any kind of platform and technology as target and not only those based on services.

1.1.3 The Role of Software Architecture in Service-Oriented Model-Driven Development

As can be extracted from the previous subsections, in both MDA and SOA, the Software Architecture plays a key role. In the case of SOA, the Architecture is viewed as a way to structure, organize and show the behaviour and evolution of a service-oriented system. In the case of MDA, on the contrary, it is considered as a container artefact and descriptor of the root components of a system and, therefore, of the elements appearing in the models defined during the development process. The *Software Architecture*, understood as the fundamental organization of a system embodied in its components, the relations among them and its environment and the principles governing its design and evolution [208] represents, consequently, the nexus between the current model-driven methodological trends and the technological approaches that conforms the basis of the future information systems.

The concepts behind the specification of Software Architectures can be perfectly aligned with the use of SOA and MDA for system development. On the one hand, the specification of service architectures in a MDA-based environment allow for the separation of the structure of the system from other system concerns, such as storage strategies, interface definition and even from the modelling of the functionality of the system in the form of specific behavioural models. By following MDA, the Architecture specification can be designed without being affected by the constraints imposed by the platforms, standards or technologies used to implement the specified Architecture (when working at PIM level of abstraction). The separation in abstraction levels also allows for the establishment of different service design strategies or architectural styles separately, besides facilitating the migration of the system to a different target platform (not service-oriented) when needed.

On the other hand, Software Architecture modelling is important due to the role the architectural models might play within an MDA-based development methodology. Traditionally, the Architecture has been given a central role in software development processes. The *Unified Development Process* [99], for example, base its development cycle in the definition of several phases in which the system is built throughout the iterative specification of the architecture, thus playing a central role. In an MDA-based methodological framework, the role given to the architectural models is also essential. In that case, since the

Architecture specification is not an isolated part of the system, the influence of the Architecture should spread to the rest of the models. The elements specified in the architectural model decide what other models must be created and what elements inside models should be included. The content of the architectural models, as a result, guides the steps to be accomplished when building the system. Software development processes and methodologies that follow this approach are known as *architecture-centric* [150].

To sum up, **Software Architecture represents the third and final pillar of the current Thesis**, being the driving and steering concept of the whole research work. Because of that, all the research efforts and results described in the current dissertation are aimed, ultimately, at allowing the specification of the Software Architecture of information systems.

1.2 Hypothesis and Research Objectives

The **hypothesis** formulated in this Thesis is that "*it is possible to develop service-oriented software architectures using a model-driven approach in which the notion of service acts as main concept of both the architecture specified and the development process itself"*.

The **main objective** of the presented Thesis, derived directly from the previous hypothesis, is, therefore: *to specify a framework for modelling software architectures in which the specification of the architecture is obtained following a model-driven process (based on the MDA proposal) and in which the concepts of the service-oriented paradigm act as foundation for the elements found in the architecture*. The latter aspect implies, necessarily, that the models will allow, at least, the definition of architectures of service-oriented systems. However, the desirable features of the framework go beyond and include:

- That the software architectures specified may refer to any kind of system, not only those implemented using service-oriented technologies. To reach that goal, the framework must support the representation of system architectures independently of its implementation language, computing paradigm or execution platform.
- That the architectural framework to be defined allows the inclusion of well-known design decisions in the form of architectural styles. To do so, and knowing that the framework will follow a model-driven approach, architectural styles and common design patterns must be represented by means of models.

- That the framework must consider the subsequent integration of the architectural models as part of a semi-automatic generation process of software architectures by means of the corresponding tools.

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

Obj. 1.- Analysis and evaluation of previous research works and initiatives related to the topic of the Thesis. Having noted that the Doctoral Thesis leans on three clearly identified areas of concern within the Software development field, this objective can be split as follows:

Obj. 1.1. Detailed study of current initiatives in the scope of MDD focusing on their proposals for the specification of Software Architectures and the development of Service-Oriented software solutions.

Obj. 1.2. Detailed study of current proposals for the development of Service-Oriented Architectures, with a special interest in their management of the architectural viewpoint.

Obj. 1.3. Analysis of the features defining the Service-Oriented Computing paradigm that may be of use to build the system Architecture, including: service composition and coordination, service standards, service contracts and constraints, service interfaces, etc.

Obj. 2.- Specification of a conceptual view of Software Architectures using Services. To do so, a definition of a DSL for Service Architectures at PIM level will have to be given. To reach this objective, several sub-objectives must be accomplished:

Obj. 2.1. Definition of a PIM metamodel whose basic elements refer to all the relevant concepts of the SOC paradigm thus comprising the semantic of the DSL at this level.

Obj. 2.2. Definition of a metamodel allowing the specification of architectural styles.

Obj. 2.3. Definition of the syntactic notation for the previous metamodels in UML (UML profile).

Obj. 3.- Specification of the modelling elements needed to represent the particularities of target execution platforms within architectural models. That entails providing with a definition of the corresponding PSM DSLs for the specification of Service Architectures. To achieve that, the subsequent sub-objectives are marked:

Obj. 3.1. Definition of a PDM (*Platform Dependent Model*) metamodel containing the common elements needed to describe the architecture of a service-oriented software solution independently of the target service-oriented execution platform and/or technology. This will serve as foundation for a DSL for Software Architectures at this level of abstraction.

Obj. 3.2. Definition of metamodel at the TDM (*Technology Dependent Model*) abstraction level supporting the representation of software architectures regarding, at least, the next implementation platforms: Web Services (using W3C standards [236]), REST services ([69]) and Grid Services (using the *Globus Toolkit* platform [72]).

Obj. 3.3. Definition of the syntactic notation for the previous metamodels in UML (UML profile) so it completes all the DSLs at the PSM level.

Obj. 4.- Specification of model transformation rules between different abstraction levels and within the same abstraction level to include architectural design decisions in the form of architectural styles. To do so, the following subobjectives are defined:

Obj. 4.1. Definition of transformation rules from PIM models to concrete PSM models (either PDM or TDM).

Obj. 4.2. Definition of transformation rules that enable the superimposition of architectural styles in PIM models.

Obj. 5.- Creation of a toolkit supporting the creation and editing of architectural models and the transformations defined among them. To carry out this objective, it will be split as follows:

Obj. 5.1. Definition of a modelling toolkit supporting model edition and model conformance verification according to the DSLs defined.

Obj. 5.2. Inclusion of graphical modelling support to the previous modelling tool based on the concrete syntax (UML profiles) defined for the DSLs specified.

Obj. 5.3. Implementation of a tool supporting the execution of transformation rules, either at the same abstraction level (PIM-to-PIM) or from one level to another (PIM-to-PDM/PIM-to-TDM).

Obj. 6.- Validation of the architectural framework and toolkit by means of their application to several case studies:

Obj. 6.1. Evaluation of the proposal throughout the architectural specification of different case studies with the aim of testing the whole architectural framework. The target implementation technologies should be based on Web Service technologies (W3C standards), REST Services and the Grid Computing platform (*Globus Toolkit*).

Obj. 6.2. Validation of the the specified modelling toolkit and the model transformations environment to check their reliability to support the framework defined. This will be accomplished in conjunction with the previous subobjective as the architectural models of the previous case studies will be modelled using that toolkit.

1.3 Research Context

The research work of this Thesis has been undertaken in the Kybele Research Group of the Rey Juan Carlos University (URJC). Part of the research effort has been performed during a stay in an external research centre, the ARCHLog Research Group of the University of South Brittany (France) for a period of four months. Moreover, this work has been carried out within the scope of several research projects. Both the associated projects and the external stay are depicted in Table 1.1.

Table 1.1. Outline of Academic Stages, Related Research Projects and Stays.

Academic Stages	**Pre-doctoral**	**Doctoral period**		
	Grad student stage	PhD Courses	Research Courses	PhD Thesis research stage

Research Projects / *Research Stays*	**2005**	**2006**	**2007**	**2008**	**2009**	**2010**
	DAWIS					
		GOLD				
					MODEL- CAOS	
				IASOMM		
			France			

Next subsections describe in detail, first, the research scope that served as methodological basis for the proposal; then the main research projects in which this work is framed and, finally, the stay of the PhD student in an external research centre.

1.3.1 Methodological Research Scope

The three pillars mentioned in Section 1.1 are required to serve as constituent parts of a framework for the specification of Service-Oriented Software Architectures following an MDA-based approach as it has been defined in the hypothesis of the Thesis. However, although this framework will be created to be used independently, it does not represent an isolated research effort. Quite the opposite, this framework intended to take a prominent place in the definition of MIDAS, a concrete methodological framework for the development of information systems [228] and that somehow was the focus of the projects related to the current Doctoral Thesis.

MIDAS suggests following a model architecture based on the principles of the MDA proposal for the development of information systems. This model architecture is defined upon a multidimensional basis (see Figure 1-1) that spreads through several abstraction levels and concerns of the system development lifecycle. For each dimension or development concern, the models and elements inside models are considered, together with the transformation rules between models and the influence of each model to the rest of models of the MIDAS structure. Within the development process architectural models acquire a special interest since the Architecture is considered as the guiding aspect of the whole process.

As explained before, MIDAS proposes a model architecture that can be seen from different points of view, each of them focused on a particular aspect of the software development lifecycle:

- **Vertical dimension**. First classification of models comes directly from the MDA proposal, which defines three abstraction levels: *Computation Independent Models* (CIM), *Platform Independent Models* (PIM) and *Platform Specific Models* (PSM). This dimension allows the evolution throughout the system development process by means of the specification of successive models. It starts from the modelling of concepts associated to the problem domain (gathered in CIM models) to the system representation taking into account the specific features of the implementation technology or target platform (by defining the PSM models).

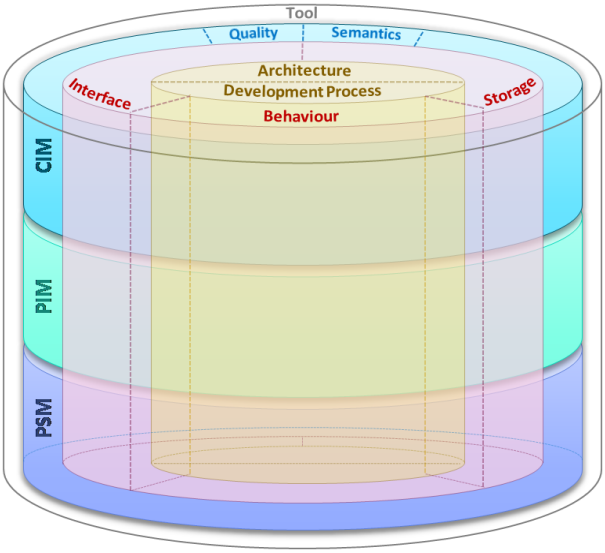

Figure 1-1. Visual representation of the MIDAS Model Architecture

- **Core models: Software Architecture**. Architectural models are placed in the middle of the model architecture as it is considered the aspect guiding the entire development process. The Architecture specifies characteristics of an information system that do not affect only to a concrete aspect of the system but to many of them. Information in the architectural models aids to identify the elements that should be modelled and even has influence on whether some models may be created or not. The definition of the development process is currently under discussion and will be left for ongoing research works.
- **Inner models layer**. The development of a software system inevitably needs the definition of particular issues regarding its constituent features. To do so, models in this layer are organized according to the different concerns that may be involved in the system. In that sense, this layer includes the modelling of Content (*Storage*, system data models), Behaviour (models depicting the system functionality) and Interaction (*User Interface*, such as the hypertext modelling in Web applications).
- **Additional models layer**. MIDAS supports the definition of other additional aspects such as *Semantics*, which defines the concrete vocabularies and meanings of the concepts and terms used in every other

model; or the modelling of *Quality* aspects and attributes. Each of these aspects has its own set of models associated.

- **Tool**. In order to give support to the MIDAS methodological framework, for each of the abstraction levels or development concerns a toolset for graphical design, transformation execution and code generation is defined.

The architectural aspect is one of the MIDAS concerns that still needed to be specified. The modelling of the Architecture within MIDAS not only describes the structure of the system, that is, its constituent elements and relations among them, but also it is supposed to become the vertebral aspect of the entire development method. This feature transforms MIDAS into an architecture-centric methodological framework (*ACMDA* in fact) [136], in which the elements specified in the architectural model have a direct influence over the rest of the models created as the development of the system evolves. The information gathered in the architecture models is needed for both deciding what elements must be present in other models and establishing and clarifying the relation among those elements and the design constraints to be taken into account depending on the desired kind of solution to build.

The proposal presented in this dissertation will help to fill in the gap of the architectural modelling within MIDAS. Given that the SOC paradigm and the MDA approach have been chosen to specify the Software Architecture, it is compulsory to define models that allow the representation, at different levels of abstraction, the main elements that form the architectural solution. This aspect includes the identification of the computational components realizing the system functionality (*services*), the potential relations among them (*service contracts* acting as connectors), their grade of specialization (*service types*) and, specially, their grouping in order to create composite elements, taking into account the different coordination means among services as composition strategy. Additionally, it is necessary to bear in mind aspects such as the representation of the dynamism inherent to any system in execution, its evolution during its lifecycle, the adaptation mechanisms to the conditions of the environments and the possibility to apply well-known organization strategies in the form of architectural styles. All of this leads to establish, accurately, the corresponding models and metamodels in order to specify the Architecture of a service-oriented system.

These models will range from the description of the Architecture at a conceptual level, agnostic of the platform chosen to execute the system, together with a mechanism to include different architectural styles; to the representation of the architecture taking into account the technologies and implementation

platforms that will have a direct influence on some of the design decisions of the software solution.

Because the framework to be defined is based on the MDA principles, the development of tools supporting the graphical specification of models is highly encouraged. Another important aspect that should be noted is that these tools must also support the execution of transformation rules from model to model. The existence of a toolkit for the framework becomes clear in the moment that it is required to validate not only the correction of the models themselves but also maintain their consistency and coherence during the transformation of the models that occurs throughout the entire development process of the Architecture.

1.3.2 Related Research Projects

The work presented in this dissertation has been carried out mainly within the scope of three research projects: DAWIS, GOLD and MODEL-CAOS.

DAWIS [TIC 2002-04050-C02-01], financed by the Ministry of Science and Technology of Spain, was a coordinated project accomplished jointly between the Rey Juan Carlos University and the Polytechnic of Madrid that was carried out from 2002 to 2005. The objectives of this project included: a) the definition of a generic architecture for the integration of digital files on the Web; b) the systematic and semi-automatic development of Web portals to access in an integrated way to multiple digital libraries. In the context of this project, the author of this dissertation worked as granted student in the development of the architecture of a Web portal that allowed the management of medical digital images based on services.

GOLD [TIN2005-00010], financed by the Ministry of Education and Science of Spain, was carried out from 2005 to 2008, and continued the research efforts accomplished in DAWIS. This project was focused on defining a model-driven approach (MDA-based in fact) for the systematization of the development of information systems, having a special look at those related with the management of medical digital images. To achieve that, GOLD had, as main objective, the development of a platform for the development of Web Information Systems (WIS). This platform was used to create an online information system for the management of digital medical images. The work of the author of this dissertation in the context of this project was the integration of the Software Architecture in the MDA model architecture of MIDAS. The case study of medical images was continued and refined in order to include the previous concerns.

MODEL-CAOS [TIN2008-03582], financed by the Ministry of Science and Innovation of Spain, started in 2009 and will last for three years, until the end of 2011. The main objective of this project is the specification of a framework for the semi-automatic development of information systems with a special attention to the use of SOC as foundational paradigm. This last project inherits the work done in the previous projects and updates 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 dissertation continues refining the integration of the architectural aspect (now fully service-oriented) within the MIDAS methodological framework, conferring it ACMDA features.

Moreover, during the research accomplished within the scope of these projects, an additional project, whose main goal was directly aligned with the main topics of the current Thesis, was carried out: **IASOMM** [URJC-CM-2007-CET-1555]**,** co-funded by the Rey Juan Carlos University and the Regional Government of Madrid. This project started in January, 2008 and ended in February, 2009. The global objective of this project was the definition of a conceptual and methodological framework for the architecture-centric and model-driven development of software systems, focusing specifically on system integration. The basic structure of the framework was centred in the definition of service-oriented architectures and, in order to simplify the process, it described a multidimensional approach leading to a framework for invasive computing.

In the scope of these projects, several Doctoral Theses have been successfully defended [36][228]. Among them, the one that has more relevance in the context of the current Thesis is the Doctoral Thesis defended by María Valeria de Castro [44]. The main topic of that Thesis was the modelling of the *behavioural* concern within the MIDAS methodological framework (presented in the preceding subsection). Moreover, a method for the service-oriented development of information systems was specifically defined. The inclusion of orthogonal aspects in MIDAS, such as the Architecture or the Semantics, made obvious a research effort about these aspects and their relation with the methodological framework. As a result, the Semantic aspect was the main topic of the Doctoral Thesis of César J. Acuña [2], in which a detailed study of the influence of the *semantics* to the development of Web Service platforms was carried out, that is, focusing on Semantic Web Services.

It is important to remark in this point that there is a Thesis recently presented (by Juan Manuel Vara [227]) dedicated to the definition of the toolkit

support and the technological strategy of MIDAS. In the current Thesis, some of the results and tools defined by J. M. Vara are used to create concrete modules for the architecture modelling tool and its specification. From a technological point of view regarding tool support, this Thesis leans directly on his research experience and outputs. Because of that, in the current Thesis, the justifications of the technological decisions taken to develop the tool (ATL as transformation language, Eclipse as execution platform, *weaving* models to annotate models, etc.) and the full explanation behind choosing DSLs for the architecture instead of merely UML profiles will not be the subject of an in deep analysis. In the sections in which these aspects need a further explanation, the corresponding references to the work of J. M. Vara will be accordingly indicated.

To conclude and as it has been explained before, the current Doctoral Thesis is focused on the development of a framework for architectural specification. Though initially conceived as the source for modelling the Architectural aspect of MIDAS, the proposed framework has gone beyond the scope of MIDAS and can be considered as an independent framework.

1.3.3 External Research Stay

A significant part of the research carried out to achieve the goals of this Doctoral Thesis was done during a 14-weeks stay (from September, 2007 to December, 2007) working in the ARCHLog Research Group (VALORIA Laboratory) of the University of South Brittany (Vannes – France). This group is specialist in the specification of formal approaches for software architectures and development processes based on the service-oriented paradigm.

During that period of time, and under the advice of Prof. Flavio Oquendo, head of the ARCHLog group, several joint tasks were completed in relation to the modelling of service-oriented architectures using the π-ADL language. The collaboration with this research group resulted in a joint publication [124] in the scope of the 2nd European Conference on Software Architectures (ECSA).

1.4 Research Method

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

The research method used in this Thesis comprises an adaptation of the one defined by Marcos & Marcos [137] for investigation in the scope of Software Engineering. This method is based on the hypothetic-deductive one by Bunge [32] which is comprised of several steps, sufficiently general, to be applied to any kind

of research activity. The main stages of the research process followed to complete the current Doctoral Thesis are depicted in Figure 1.2.

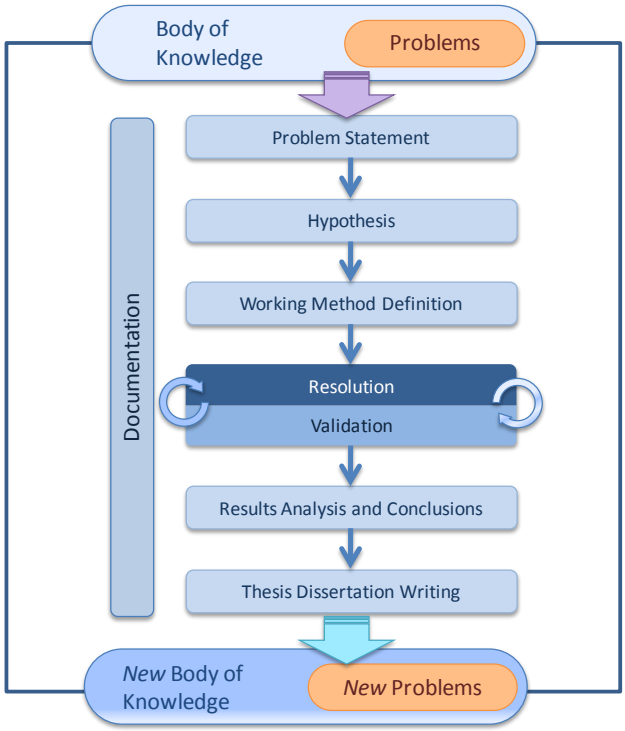

Figure 1-2. Overview of the Research Method.

The '*Body of Knowledge*' in which the Thesis is framed, gathers the issues stated in the introduction of this dissertation. It acts as a preamble for the research process and covers all the terms that the three pillars mentioned previously entail. From that context, the identification of common problems that may be solved using the principles of MDE, the foundation of Service-Orientation and the role an architectural description may play within the building of a modern, efficient and useful software solution lead to precisely define the problem to be solved ('*Problem Statement*' stage) from which a '*Hypothesis*' is formulated as starting point of the research work.

As it can be seen in Figure 1.2, the definition of the method is considered as another step within the method used. 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. In the context of the current Thesis, the research phase

corresponding to the '*Resolution and Validation*' step is of great interest as it represents the core of the research work. A deeper explanation of this stage and the working approach used are detailed in next subsection.

Once a proposal is derived from the previous research stages, it is the time to analyze the results obtained as a consequence of the application of that proposal to concrete scenarios ('*Results analysis and Conclusions*' stage). These analysis tasks serve as hotbed for the elicitation of some conclusions from the work accomplished. Next step is to gather all this research experience in a final Thesis dissertation.

Though the final step of the research process may be embodied in the task of writing the Thesis dissertation, 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 that may be the object of future research tasks and initiatives.

1.4.1 The Resolution and Validation stage

The method followed in this Doctoral Thesis for the '*Resolution and Validation*' stage conforms to an incremental and iterative process model. This research stage iterates in the refinement of the elements defining the proposal and increments it by completing the shown tasks. Moreover, each task provides a feedback to some of the other related tasks. A graphical overview of the process can be seen in Figure 1-3.

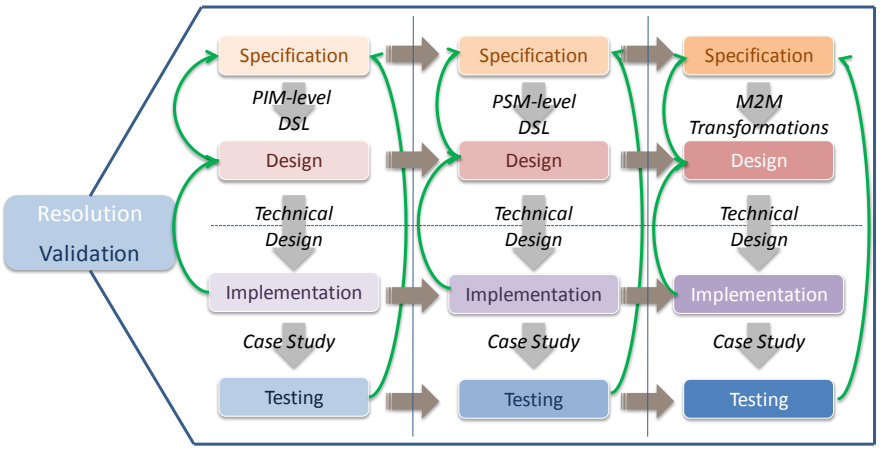

Figure 1-3. Operationalization of the 'Resolution and Validation' research stage.

As it can be observed, this stage is itself comprised of several phases or steps divided into several consecutive iterations. The first iteration centres its

attention in the definition of the part of the proposed framework supporting the modelling of the conceptual view of a Software Architecture, that is, at the PIM level of the MDA approach. The second, in turn, is dedicated to provide with a PSM view of the architectural models. The third one deals with the establishment and implementation of the rules that allow to perform model-to-model transformations (M2M Transformations in Figure 1-3). Although these three iterations are described here as being sequential, in many stages of the research they have been intertwined. In addition, some other aspects needed to fully complete the framework proposed in this Doctoral Thesis are considered during the research process.

Zooming into each of the iterations, the '*Specification'* step is focused on the definition of the aspects and foundational features needed to build up different DSLs for architectures and the theoretical aspects that support the framework proposed using services and at different abstraction levels.

After an initial outline of the metamodels that allow the specification of software architectures, it is necessary to refine the previous models and languages as well as progress into a validation of these elements. These tasks are accomplished following two complementary directions. On the first one, the framework is required to be supported by a graphical tool that eases the creation of architectural models ('*Technical design'*). This tool serves both as graphical modelling environment and support for model transformation execution; besides, it allows validating the conformance of the models and the transformation rules. The design process and its subsequent implementation are also intermediate steps considered in every iteration. The tool itself can be considered as a proof-of-concept for the DSLs defined, as it permits the establishment, execution and checking of language rules and restrictions over models.

On the other side, the feasibility of the proposed framework and its real applicability is tested throughout its use both in concrete scenarios of real-world case studies, such as a system for the medical image management or a gateway for sending SMS; and simulated ones, such as the use of a sport metaphor for service composition modelling.

To sum up, the three aforementioned steps, DSL Specification, Tool Design and Implementation and Testing with case studies, do not represent a straightforward process, but an iterative one in which the information flows back and forth to improve each of the research steps.

1.5 Structure of the Dissertation

In order to have a better comprehension of the structure of the dissertation in relation to the work accomplished, Figure 1-4 dissects the anatomy of the proposal into its constituent parts and elements created for its completion. Associated to each building block there is a reference to the chapter or subsection in which the issue is fully explained.

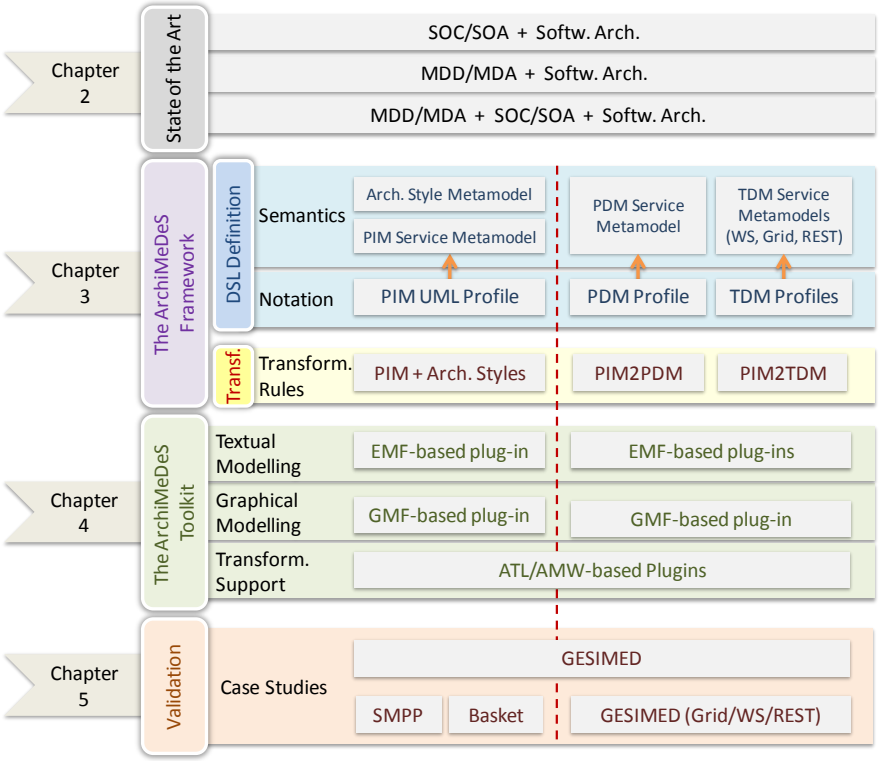

Figure 1-4. Anatomy of the proposal in relation to the structure of the dissertation.

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

- **Chapter 2. *State of the Art.*** Presents a detailed study and analysis of related works and research initiatives regarding to: a) using a model-driven approach; b) using the service-oriented paradigm as basis; c) the role played by the software architecture.

- **Chapter 3.** ***The ArchiMeDeS Framework***. Explains thoroughly the foundations and main features of a framework for the model-driven development of service-oriented architectures: *ArchiMeDeS*. This chapter includes the specification of the metamodels needed for the description of the Architecture at different abstraction levels, the transformation rules allowing the transition from one model to another and the inclusion of architectural style characteristics into the architectural models.
- **Chapter 4**. ***The ArchiMeDeS Toolkit.*** The chapter shows the development process a tool supporting the previous DSLs and model transformation defined previously.
- **Chapter 5**. ***Validation***. This chapter is devoted to present the validation means used and results obtained to verify the legitimacy of the presented framework. In it, a full case study is developed following the proposal and using the toolkit created. In addition, other complementary partial case studies are used to cover the aspects not considered by the previous one.
- **Chapter 6**. ***Conclusions and Future Works***. In this chapter the main conclusions are established starting from an analysis of achievements together with the future research lines derived from the work of this Thesis. Additionally, the contrasted results are presented in the form of the articles and paper published as result of this research work.
- ***Appendices*.** The last sections of the current dissertation include:
    - A summary of the content of the **Thesis dissertation in Spanish**.
    - The **bibliography** and literature consulted in this Doctoral Thesis together with other resources found online.
    - A list of **acronyms** used throughout the entire dissertation.

Chapter 2:
State of the Art

This Chapter presents a study on the state of the art in the research areas covered by this Doctoral Thesis. It is considered as a fundamental part of the dissertation as it helps to recognize the contributions that the Thesis may add to the existent knowledge in the scientific area of influence and to position the proposal among others.

With that premise in mind, firstly, it is compulsory to accurately define a coherent classification criterion so it is possible to analyze all the relevant works that may have an influence in the scope of the Doctoral Thesis presented. Therefore, Section 2.1 comprises the set of concepts that allow the evaluation of each related research initiative regarding Service-Orientation, Model-Driven Engineering and Software Architectures.

After that, Section 2.2 proceeds to apply those criteria to the works that are somehow related to the use of, at least, two of the three abovementioned areas. Since the research fields affected do not represent isolated areas within Software Engineering, the works analyzed will be mainly those taking advantage of more that one of the topics. For example, those that benefit from applying model-driven techniques for architecture specification or those focusing on the architectural features of the service-oriented paradigm. Obviously, the biggest interest will be put on those works considering the three areas at a time.

Finally, this Chapter will end up drawing some of conclusions reached from the analysis done.

2.1 Evaluation Criteria

As already stated, the classification criteria used for analyzing the current state of the art must include the features that characterize the core subjects of this Thesis: *Service-Orientation*, *Model-Driven Engineering* and *Software Architecture specification*. In the following, and for every area, it is given: a short introduction to the topic, the features used to analyze them and the values that will be assigned to each feature.

2.1.1 Issues related to Model-Driven Engineering

Beyond the initial hype of Model-Driven Engineering, this discipline has evolved to becoming one of the most widely accepted development approaches in Software Engineering. As a result, a number of initiatives have emerged in this field during the last years. However, the advent of so many proposals turned out in a myriad of acronyms to refer to the same approach. Different authors use different names to refer to (more or less) the same thing. Thus, it is possible to talk about *Model-Driven Software Development* [215] (MDSD), *Model-Driven*

Development [93] (MDD), *Model-Driven Engineering* [21] (MDE), *Model-Based Software Engineering* [203] (MBSE), *Model-Driven Architecture* [79] (MDA) and so on. For the sake of clarity, in this dissertation the acronym MDD will be used as common term for all of them, referring, specifically, to *proposals whose main and central element of the Software Engineering is the model*. The consideration of MDA will be maintained separately in order to differentiate the proposal of the OMG from the rest.

The aspects to bear in mind for the evaluation of proposals and research initiatives that somehow relate to MDD, considering the main goal of the current Thesis, include the next ones:

- **Role given to the Architecture within a development process.** Recent works [136][150] have shown that model-driven proposals, and more precisely those involving the MDA approach, leave aside the specific modelling of the system architecture; either avoiding its explicit specification or considering it as an independent artefact to be modelled without a direct correlation with the rest of the system models. Since the decisions posed over the architecture usually affect to the components implementing any system, the role given to the architectural modelling represent a key aspect when observing and evaluating the related works found in the bibliography.

  The values that will be used for this feature will be:

  - *Not considered [N]*. Assigned to those model-driven initiatives that do not provide with a specific model for the system architecture, including the architectural information (topological, evolutionary or dynamic) within other models.
  - *Modelled as an independent artefact [IA]*. For initiatives that consider software architecture modelling as an independent process without any correlation to the development of other concerns of the system.
  - *Positioned as central artefact (architecture-centric approach) [AC]*. This value will locate initiatives considering the architectural model as the guiding aspect of the system development process, having an influence on other modelling concerns.

- **Modelling approach**. As stated along this document, the central building blocks in MDD are models. These models must be specified using a concrete language or notation following precise construction rules.

Consequently, any model-driven proposal is commonly based on the definition of new modelling languages according to the concepts to include in the models. To that end, different approaches may be followed. The different alternatives that will serve as values for evaluating this feature will be:

- *Definition of DSLs (Domain Specific Languages) [DSL]*. As a way to support the modelling approach based on the definition of metamodels containing all the concepts, relationships, constraints and rules needed to create domain models. It entails the creation, from scratch, of all the elements needed for a complete language specification (including the notation). The use of ADLs (*Architecture Description Languages*) is understood as a concrete type of DSLs for the specification of software architectures.
- *Definition of UML extensions via UML Profiles [UML]*. This alternative is used by some initiatives taking advantage of the extension mechanisms provided by UML for modelling purposes. They benefit from the widespread that UML has gained but rely greatly on the constraints and features of the host language.
- *Hybrid approach [Hyb]*. The last option is to step at a medium point between the previous alternatives, by defining metamodels collecting the semantics of the models but using a well-known notation such as UML for the specification of the concrete syntax of the language.

- **Support for modelling tools**. In the scope of MDD, the availability of toolkits becomes particularly necessary for several reasons: the need for conformance between the models created and the corresponding metamodels, the obvious benefit of using visual tools for the creation of models or the possibility to automate possible transformations of the models are among those reasons.

  The existence of toolkits associated to development processes and methodological frameworks also helps to classify and evaluate the different proposals found in the bibliography. The logical values used in this case will be *YES* and *NO*, depending on whether the initiative under study considers the development of tooling support or not. In affirmative cases, the focus rely on their coverage of several desirable capabilities

including extensibility of the tool implementation, support for meta-modelling, support for model transformation and conformance validation, generation of editors for terminal models (create *ad-hoc* editors from metamodels), standardization, interoperability, etc. For that aim, the underlying technology or platform will be of special interest. Known the widespread of the *Eclipse* platform [93] in that context, those initiatives using this framework will be clearly identified and separated from those programming a toolkit from scratch.

- **Definition of model transformations**. One of the most promising features that have accompanied the evolution of MDD is the possibility to define rules for automating the translation of these models into code, generate documentation or even obtain other models. Through the specification and subsequent execution of transformation rules between models it is possible to evolve in the process of Architecture development, thus assuring a higher level of quality and expecting a minor number of errors during the Architecture lifecycle.

  The values to consider when analyzing model transformation concerns will be *YES* or *NO*; however, for affirmative answers, the analysis will reflect also the transformation strategy followed:

  - *Not considered [N]*. Used for those model-driven initiatives not considering model transformations as part of their proposals.
  - *Model transformations definition [Y (strategy/language)]*. It will be used for initiatives that positively define model transformations. Inside this category of proposals, it is possible to go one step beyond and specify the transformation strategy applied to models: supported by languages such as QVT [171] or ATL [104], by means of graph transformations or in other languages specific for a concrete methodological framework (such as IOM for the UML4SOA framework [142]).

2.1.2 Issues related to Service-Orientation

In the last years, several proposals have come up, with different goals and from different points of view, which tackle the problem of system development using a service-oriented perspective. The number of related initiatives is quite high because, among other reasons, the current interest shown by companies and standardization consortiums on this topic. Among all of them, this work of Doctoral Thesis focuses on those having a special look at the specification of the architectural aspect as a way of going beyond the technological scope and

extending the region of influence of the service-oriented paradigm. The following features are used to evaluate the degree of influence that the architectural aspect has inside the analyzed proposals.

- **Strategy for service composition**. One of the main features that distinguish the SOC paradigm from other computing paradigms, such as *Object-Orientation* or *Component-Based Software Engineering* (CBSE), is the way the fundamental building blocks of the paradigm are combined to create compound structures. Service composition represents a shift in the way architectural elements are composed. In the SOC paradigm, services are coordinated rather than composed. Since services are understood as independent computational entities that must always maintain that independency all over their entire lifecycle, so they have to preserve this feature even in the case they take part in service groupings. The coordination strategies for services can be divided into two: *orchestrations* and *choreographies*.

  About service composition, the analysis of the bibliography shows that research initiatives tackle this aspect of the SOC paradigm from different approaches. Accordingly, the values used to classify them are:

  - *Service-Oriented coordination scheme [Orch.][Chor.][Both]*. Most service-oriented initiatives consider some coordination strategy, either if it is based on orchestrations, choreographies or both.
  - *Component-based composition means [CB]*. This value will be assigned to initiatives using services from a component-based point of view, thus failing to take advantage of the service coordination means. These initiatives adapt the traditional component and connector scheme for the specification of service-oriented architectures.
  - *Not explicit [N]*. There are very few cases in which service composition is not explicitly considered; however, they must be included within the evaluation of the state of the art.

- **Abstraction level at which services are considered**. It is a reality that the SOC paradigm has broken the technological barrier and has impregnated other levels, away from the implementation one. Because of that evolution, various related works in this context are devoted to reconcile the points of view of the concept of service [169]. In that sense, many proposals focused on the definition of phases, stages and tasks

needed to perform to cover the gap between one abstraction level and another have appeared recently. In that case, the values assigned to that feature will comprise one to three values, depending on the levels considered, among the following:

- *Business level [B]*. Proposals that work in this level are aimed at representing services as key part of the definition of business processes, understood both as product and as describing tool.
- *Technological or implementation level [T]*. The origins of the SOC paradigm are set in the implementation level, and thus proposals that put their focus of attention on the use of standards, languages, and implementation alternatives should be taken into account.
- *Architectural level [A]*. Under this category rest intermediate approaches that cope with the representation of service systems at a more conceptual level in which the concepts behind the SOC paradigm are defined without any interference of the underlying implementation platform of choice.

2.1.3 Issues related to Software Architecture

Since the main goal of the current Thesis is the definition of a framework for the model-driven specification of software architectures, it becomes critical the analysis of the existent proposals that somehow consider the architectural view within their work. For that, the definition of the state of the art, in relation to the architectural aspect, must pay a special attention to those aspects needed for the correct and complete specification of a Software Architecture.

- **Base paradigm of the Architecture**. Every Architecture specification is built upon a set of elements that interact with each other and the environment. However, the type of elements, their semantics and the principles that guide the design and evolution of the Architecture can follow several paradigms leading to different design strategies. For that reason, it is common to find that Architecture specifications appear associated to adjectives such as 'Object-Oriented', 'Component-Based', 'Agent-Based' or 'Service-Oriented' focusing on the concepts of *object*, *component*, *agent* or *service*, respectively, to outline the system Architecture. In the scope of the current Thesis those initiatives modelling the Architecture using SOC as foundational paradigm are of interest. Related research works will later be classified using those concepts as possible values for this criterion with the following values:

Aspect-Oriented Paradigm [AS], *Object-Oriented Paradigm [OO]*, *Service-Oriented Paradigm [SO]*, *Component-Based Paradigm [CB]*.

- **Support for architectural rationale**. In order to analyze the support for organizational patterns at the architectural level, the analysis of the initiatives considered in the bibliography must include the traditional architectural styles [208]. However, and due to the change in the trends of software development and needs, several other patterns have come up associated to the Service-Oriented paradigm [35]. The support for all these new architectural styles or, more properly, *architectural design strategies or decisions*, are also aspects that facilitate the evaluation of the current proposals related to the topics of this Thesis. The values used for this feature will be:
    - *Architectural styles [AS]*. This value will be assigned to those initiatives considering the definition of ways to include traditional architectural styles as part of their proposal.
    - *Explicit design decisions [DD]*. Used for those initiatives centred in the development of architectures and considering the definition of specific design decisions as part of them, but not identified as architectural styles. This situation is quite common among service-oriented initiatives and so they will be specifically marked.
    - *Not considered [N]*. If no reference to the inclusion of architectural decisions is made as part of the initiative this value will be given.

2.1.4 Summary of the evaluation criteria

Table 2.1 collects all the features and possible values that will be used to evaluate the most relevant initiatives and proposals in the scope of the topics of the current Thesis.

Table 2.1. Summary of features and values used in the state-of-the-art.

Feature	*Value Description*	*Corresponding Acronyms*
MODEL-DRIVEN ENGINEERING		
Role given to the Architecture	Not considered, Independent artefact, Architecture-centric	N / IA / AC
Modelling Approach	DSL, UML extension, Hybrid	DSL / UML / Hyb.
Tool Support	Yes [platform] / No	Y [platform] / N
Model Transformations	Yes [strategy/language] / No	Y [strategy/language]/N
SERVICE-ORIENTATION		
Service Composition Scheme	Orchestration, Choreography, Both, Component-Based, Not explicit	Orch. / Chor. / Both / CB / N
Abstraction Level	Business, Architectural, Technology	B / A / T
SOFTWARE ARCHITECTURE		
Base Paradigm	Aspect-Oriented, Component-Based, Object-Oriented, Service-Oriented	AS / CB / OO / SO
Architectural Rationale	Architectural Styles, Explicit Design Decisions, Not considered	AS / DD / N

2.2 Related Works and Research Initiatives

The three main topics covered by this Thesis (Model-Driven Engineering, Service-Orientation and Software Architecture) have a great incidence within the scope of current Software Engineering research. In order to select and evaluate those initiatives, proposals or works that may have a significant influence on the work presented in this dissertation, this subsection classify them according to the covering degree of the aforementioned topics. This way, the **proposals evaluated** in the survey will necessarily consider the following:

- A *Service-Oriented approach for Software Architectures* (Section 2.2.1). This category will comprise the consortiums, companies and proposals releasing standards, models and other artefacts aiming at the specification of Software Architectures using services.
- A *Model-Driven approach for Software Architectures* (Section 2.2.2). This category, in turn, will gather all the relevant proposals that tackle Software Architecture specification using MDD principles.
- A *Model-Driven and Service-Oriented approach for Software Architecture* (Section 2.2.3). That is, the intersection of the three pillars

of this Doctoral Thesis and that will be of special interest in order to position the proposal among them.

Works focusing only on SOC (or SOA) and MDD (or MDA) will not be included in the study since the spirit of this Thesis is the definition of a framework for the specification of the architectural view of a software solution, not a software system as a whole. Their interests rely, normally, on the definition of service-oriented solutions using a model-driven approach without realizing the importance of defining explicitly the system Architecture as independent element. As a consequence, those works in which the specification of the (role of the) Architecture is avoided [1][154] or, at least, mixed with the definition of the functionality of the system, will be left aside from that analysis. The same criterion has been applied to works exclusively centred in only one of the issues.

2.2.1 Using a Service-Oriented Approach for Software Architecture

The Service-Oriented Computing paradigm represents a step forward in comparison to other widely accepted paradigms such as *object-orientation*, *aspect-orientation* or *component-based software engineering*. Similarly to agent-based approaches, its principles refer to entities understood from a higher abstraction level, not constrained to the programming level but to computing entities with an existence that is independent from the platform or implementation technology. Since its origins it has demonstrated its capabilities for quick evolution and spread to other scopes. It is based on entities that work, in essence, in highly dynamic distributed environments in which the critical aspect does not lie within the concrete programming of the computing elements participating in the system. On the contrary, the SOC paradigm prioritizes their interrelation, the means of communication established, the message exchange patterns performed, the availability of the resources associated to each service and the policies and restrictions applied in every task execution. Due to all this features, the importance of service-orientated systems rely greatly on their topological structure, built upon individual identifiable services, and their behaviour and evolution during its entire lifecycle. This means that the Architecture of Service-Oriented systems and applications represents one of the core aspects to be taken into account when trying to develop solutions based on the SOC paradigm.

The architectural perspective of service-oriented systems is widely known as SOA (*Service-Oriented Architecture*)[59][127][165]. However, the precise definition of this term varies depending on the aspect or research field in which

this concept is used. In the context of this Thesis, the term SOA will be considered as *a collection of interacting services building up a system Software Architecture*.

Since SOA is considered one of the key aspects in relation to the SOC paradigm, many organizations and consortiums are dedicated to the definition of any of the numerous aspects of the SOA universe. For instance, they define reference models and reference architectures for SOA, standard languages for services, try to specify layers and protocol stacks that are used to implement this paradigm, etc. In the next subsections, the work accomplished by several of these organizations is analyzed in detail. It includes: the *Reference Architecture* and *Reference Model* for SOA by OASIS, the *Reference Architecture* of NESSI, *Web Service Architecture* of the W3C with a brief look at the REST way of building SOA, the approach of *Grid Computing* and the efforts of the OGF in that direction, the *Service Component Architecture* of the Open SOA Collaboration organization, the *Reference model* of The Open Group, and some other independent works that tackle SOA as a new architectural style or as a way to fix some design decisions during the development of service-based systems.

2.2.1.1 SOA-Reference Model and SOA-Reference Architecture of OASIS

The *Organization for the Advancement of Structured Information Standards (OASIS)* is a non-profit, international consortium that drives the development, convergence and adoption of e-business standards. Their standardization efforts include the definition widely-accepted standards in the fields of Web services, security, business transactions, supply chains and interoperability within and between marketplaces [166].

Some of the activities performed by the OASIS consortium include the definition of languages, models and reference architectures for SOA. Among them, and with the objective of this Thesis in mind, two documents related to the architectural viewpoint of SOC clearly stand out: a *Reference Model for Service-Oriented Architectures* (OASIS SOA-RM), established as standard in October 2006 [165]; and, a *Reference Architecture for SOA* (OASIS SOA-RA), published as public review draft in April 2008.

The OASIS SOA-RM defines *an abstract framework for understanding significant entities and relationships between them within a service-oriented environment, and for the development of consistent standards or specifications supporting that environment*. Since it is a "Reference Model", it defines the essence of SOA from a high level without giving any particular solution to the problems that may arise when specifying concrete architectures following this approach.

OASIS SOA-RM presents a unified vocabulary of concepts, axioms and relationships within the scope of the SOC paradigm. It acts as base support for the development of reference architectures and, therefore, concrete service-oriented architectures that use those concepts of the reference model and follow the abstract solution provided by reference architectures and architectural patterns designed for that aim in particular.

This standard starts from the definition of the term SOA, as it is understood by the OASIS consortium: *a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains*. From that premise, RM-SOA establishes the fundamental principles that any architecture (or reference architecture, in general) based on services must have: visibility, interaction and effect. *Visibility* refers to the possibility to see the capabilities offered by some entity by any other entity with needs to be fulfilled. *Interaction* introduces the idea of realizing the exploit of those capabilities publicly shown by means of one-to-one communications. Finally, *(real world) effects* are understood as the result of an interaction. From all these premises arises the conception of the term *service* as the mechanism by which needs and capabilities are brought together, using a prescribed interface and exercised consistently with constraints and policies as specified by the service description.

The reference model is completed with the description of three more side-concepts needed to consistently define a service-oriented architecture: *Contracts and Policies*, presenting the conditions of use established in an agreement needed to achieve for two services to communicate each other; *Execution Context*, that comprises all the implementation requirements and features that drive the interactions between services; and a *Service Description*, considered as one of the main hallmarks of SOA, it gathers all the information from the service acting as starting point of agreement and basis of any possible interaction with the service described. Since the reference model is not tied to any technology, implementation standard, protocol or under any concrete prescribed architectural decision, the aspects included in this *Service Description* may vary from one implementation to another. It usually includes the description of the service interface, information about the behaviour model behind the service interaction, semantics used by the service and in message exchanges, data about reachability of the service and, of course, about the functionality provided by the service described.

In summary, the OASIS Reference Model for Service-Oriented Architecture is a standard that establishes the main assumptions that an application of the SOC paradigm in concrete distributed environments should make in order to be considered a true SOA-based application. It is important to remark that this

standard does not mention, in any case, how services must be composed (*architectural composability*) or how a service-oriented system may respond to changes in the environment (*architectural dynamism*). In addition, it does not reference what additional architectural elements (apart from services) should be needed to fulfil the requirements of SOA-based systems.

To cover these issues, the OASIS consortium released in April 2008 a document (not defined as standard) specifying their vision of what any service-oriented architecture should include, that is, a reference architecture description for SOA. This reference architecture is based on three main views of a SOA Architecture: *Business via Services*, which captures what SOA means for people using it to conduct their businesses; *Realizing Service Oriented Architectures*, that focuses on the infrastructural elements that are needed to support the construction of SOA-based systems; and *Owning Service-Oriented Architectures*, that addresses the issues involved in owning a SOA as opposed to using one or building one.

All in all, OASIS' SOA-RA and SOA-RM understand the architecture of a software system from the concepts of the SOC paradigm, do not consider the modelling of architectural styles or design decisions related to service configurations and identify both service composition means (orchestrations and choreographies) from a conceptual point of view without interference of technological details.

2.2.1.2 NEXOF-RA: the SOA Reference Architecture of NESSI

Within the scope of the 7th Framework Programme of the European Community [62] there is a project devoted to establish and standardize the aspects that any architecture based on services should comply with, that is, their goal is to define accurately a reference architecture for SOA. This Project is named NEXOF-RA [160] and is driven by the *Networked European Software and Services Initiative* (NESSI), a European Technology Platform dedicated to investigation in Software and Services.

NEXOF-RA (*NEXOF Reference Architecture*) project is the first step in the process of building NEXOF, a generic open platform for creating and delivering applications enabling the creation of service based ecosystems where service providers and third parties easy collaborate. Still a work in progress (the project will last until 2012), NEXOF-RA main results will be the Reference Architecture for NEXOF, a proof of concept to validate this architecture and a roadmap for the adoption of NEXOF as a whole. As it is an ongoing work most of the elements that appear in the Reference Architecture structure are still under development.

However, the master guidelines for the specification of the NEXOF-RA are fully defined.

NEXOF-RA focuses on the architecture of a service-based software system infrastructure. It is provided in the form of a construction kit that guides the construction of specific SOA infrastructure architectures. The construction kit consists of a set of building blocks implementing architectural patterns. These architectural patterns, in turn, are related to a conceptual architecture model. To do so, the NEXOF-RA Model [164] captures the relevant entities and concepts on a conceptual level as well as their dependencies that constitute such a service-oriented system. In addition, the NEXOF-RA Model fosters the communication about the relevant elements on a higher abstraction level.

NEXOF-RA understands as service "*an action performed by one entity (provider) that matches a request of another (requesting entity), according to the interpretation of the latter*". From this definition, NEXOF-RA starts to define a "Service Compliant Reference Architecture" by defining the characteristics that a Service Architecture should have in order to be able to interact under the environments defined in NEXOF. These *Compliant Platforms* (as they are named in the NEXOF-RA definition) conserve and share the features of openness, expandability, federation and interoperability that the NESSI consortium aims to foster with its definition of NEXOF-RA. It is necessary to explain that a "platform" from the point of view of NEXOF can be divided into software platforms (the operating systems), hardware platforms (the physical resources and devices executing the software platforms) and service platforms, understood as the software needed to fully execute a service (or a set of services) together with all the support it should need to provide dynamism and scalability, policy control and security enhancement. It is under this conception of platform that the efforts of NEXOF try to develop the NEXOF-RA rules for compliant infrastructures and platform compatibility.

All this concerns refer to a very simplistic view of the SOA world based on the existence of three main elements: a service provider, a service requester and a service registrar. The relation among these three elements is broadly known as "the SOA triangle". From the description of each of them, emerge the specification of message exchange, service composition, service discovery and service monitoring (or management), concepts that are used within the proposal described in this dissertation.

About the evaluation of NEXOF-RA and its corresponding model, this initiative also use services as foundational element at a conceptual level. However, it understands service composition in a traditional way, without explicitly

including the support for service orchestrations or choreographies. A remarkable aspect of NEXOF-RA is that it considers the inclusion of architectural styles as a key aspect of the architecture specification process.

2.2.1.3 The Web Service Architecture of the W3C

The *World Wide Web Consortium (W3C)* develops interoperable technologies to make the Web a robust, scalable and adaptive infrastructure for a world of information. W3C long term goals for the Web include: Universal access to the Web for all people of culture, education, material resources, etc.; a Semantic Web that permits each user to make the best use of the resources available on the Web; a Web of trust, with careful consideration for the novel legal, commercial and social issues raised by its associated technologies.

This consortium publishes many of the implementation standard languages that may be used when aiming at the development of service-oriented systems. However, this subsection focuses in the specification of the *Web Service Architecture* (WSA) as, from the point of view of the W3C, the right model that any service-oriented implementation should consider. Although it is not said explicitly, the languages delivered by the W3C in relation to the SOC paradigm, comply with the vision of WSA and support different aspects of that architectural model.

WSA is intended to provide a common definition of a Web service, understood as *a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artefacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols*. The WSA provides a conceptual model and context for understanding Web Services and the relationships between the components of this model. The Architecture does not attempt to specify how Web Services are implemented, and imposes no restriction on how Web Services might be combined. The WSA describes both the minimal characteristics that are needed by many, but not all, Web Services. Moreover, it identifies those global elements of the global Web Services network that are required in order to ensure interoperability between Web Services.

The document describes both a reference model and a reference architecture. The WSA meta- model is comprised of four different models (see [233] for more details): *Message Oriented model*, *Service Oriented model*, *Resource Oriented model* and *Policy model*. In addition, it describes both the common functionalities of a Web-Service based SOA and a common scenario for using Web Services. Regarding the topics of this dissertation, models gathering

the main concepts of interest are the *Service Oriented model* and the *Resource model*.

The *Service Oriented Model* focuses on aspects of service, actions, roles, agents among others. The primary purpose of this model is to explicate the relationships between an agent (understood as any person or organization making use or managing a Web Service) and the services it provides or request. This model is built on the *Message Oriented Model* but its focus is on action rather than message. The meta-data associated to the service takes a prominent role since it allows the description of the semantics and means of interaction behind a Web Service. Moreover, the Service Oriented Model allows the interpretation of message requests for actions and as responses to those requests. Furthermore, it allows an interpretation of the different aspects of messages to be expressed in terms of different expectations, in well understood ways, of the different parts of the message: in effect, and incremental and layered approach to service (such as the one followed by the Web Service specification) is possible using well understood headers. The set of standards and languages delivered by the W3C aims at the specification of many of the aforementioned layers included in message headers and service descriptions.

The *Resource Oriented Model* focuses on resources that exist and have owners. This model is adopted from the Web Architecture concept of resource. A resource is defined to be anything that can have a unique identifier (a URI). This architecture is only concerned with those resources that have a name, may have reasonable representations and which can be said to be owned. Resources are intrinsically related to services since any service is considered as a resource, the difference between them relies on the existence of a well-known and described interface that can be used to access a service; a resource, as isolated entity, must be accessed trough the use of a concrete service.

To sum up, although WSA conceals many aspects of SOA, it focuses mainly on the conceptual modelling of elements that will be technologically supported by the standards delivered by the W3C. It does not make any reference to the support for architectural styles or design decisions and the only composition mean mentioned is that of orchestrations.

2.2.1.4 The case of REST Web Services

In his PhD Thesis, Roy Fielding [69] proposed to constrain the vision of WSA based on Web Services in order to achieve more reliable Web applications. His proposal is known as *Representational State Transfer (REST)* and has spread as a concrete way to implement Web Services. The REST Web is the subset of the WWW (based on the HTTP protocol) in which agents provide *uniform interface*

semantics –essentially reduced to the create, retrieve, update and delete operations related to a Web Service– rather than arbitrary or application-specific interfaces, and manipulate resources only by the exchange of *representations.* Furthermore, REST interactions are stateless, in the sense that the meaning of a message does not depend on the state of the conversation.

The set of constraints imposed by the REST architecture include:

- Application state and functionality are abstracted into resources.
- Every resource is uniquely addressable using *Uniform Resource Identifiers* (URI).
- All resources share a uniform interface for the transfer of state between client and resource, consisting of:
  - A constrained set of well-defined operations.
  - A constrained set of content types.
- A protocol which is: client-server, stateless, cacheable and layered.

Since the specification of this proposal Web Services are frequently divided into REST-compliant Web Services (namely *RESTful Web Services*) and arbitrary Web Services, in which the service may expose an arbitrary set of operations. Any Web Service violating the above mentioned constraints is considered as not being a true "RESTful Web Service".

The interest in this proposal is supported mainly for the great acceptance that has achieved within the Web Service development community, but, more importantly, by the fact that this approach can be viewed as a concrete architectural style that should be taken into account in the moment of defining architectural style models to include within any Service-Oriented Architecture. Another reason to bear in mind REST services is the fact that the technological support for this kind of services does not rely on the existence of specific architectural elements (such as WSDL interface descriptions of Web Services).

2.2.1.5 SOA from the perspective of the Community of Grid Computing

According to Foster et al. [75], the paradigm of Grid Computing aims at solving the problem of *coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.* This definition comes from the late 90s when the importance of specifying new technological solutions for the emerging needs of enterprises relying on distributed technologies arose. From that time, the concepts behind the Grid Computing have evolved to prevail as foundational base for the creation of distributed environments based on services. Current grid environments are built upon the existence of a middleware allowing

to access to sharing pools of network-distributed resources to deliver any kind of applications and services seamlessly.

Among the different research groups and organizations dealing with the principles that must guide the development of Grid solutions the one that is traditionally considered as the *de facto* consortium for the establishment of best practices and standards for the Grid is the *Open Grid Forum (*OGF). The OGF consortium was formed from the merging of the *Global Grid Forum* (GGF) and the *Enterprise Grid Alliance* (EGA) in 2006. GGF had a rich history and established international presence within the academic and research communities along with a growing participation from industry. EGA, in turn, was a consortium focused on developing and promoting enterprise grid solutions.

The main goal of OGF is to accelerate grid adoption to enable business value and scientific discovery by providing an open forum for grid innovation and developing open standards for grid software interoperability. OGF provides an open forum that brings together key individuals and organizations from the grid community to align requirements; identify and remove barriers; workshop best practices that will expedite grid adoption.

Inherited from the works accomplished by the GGF, OGF has continued to refine and develop the specification of the *Open Grid Services Architecture* (OGSA) [76], a framework for distributed system integration, virtualization and management that includes the definition of a core set of interfaces, behaviours, resource models and bindings to build grid environments.

OGF has embraced OGSA as the blueprint for standards-based grid computing. 'Open' refers to the process used to develop standards that achieve interoperability. 'Grid' is concerned with the integration, virtualization, and management of services and resources in a distributed, heterogeneous environment. It is 'Service-Oriented' because it delivers functionality as loosely coupled, interacting services aligned with industry-accepted Web service standards. The 'Architecture' defines the components, their organizations and interactions and the design philosophy used. OGSA-WG is developing the architecture and its constituent specifications and profiles in collaboration with a number of fellow working groups.

OGSA defines a set of services that must be present in any grid middleware to guarantee the correct and complete use and management of a Grid environment thus maintaining an acceptable quality of service level. In that sense, in the OGSA specification, there are groups of services dedicated to manage security, to execute jobs in resources of the grid, to manage data and resources and even to control and

monitor the status of the grid environment itself (see [72] for an overview of the OGSA layered architecture and scope of influence within a system).

As it can be seen, OGSA is devoted to define what service must be present in a distributed environment in which several services offer different capabilities in order to serve as basis for other upper level tasks. In that sense, the OGSA vision of SOA is mainly focused on how these middleware serves communicate and relate to each other, relying on well-established standards and languages for the implementation of those services.

2.2.1.6 The Service Component Architecture from the Open SOA Collaboration

The *Open SOA Collaboration* (OSOA) represents an informal group of industry leaders that share a common interest: defining a language-neutral programming model that meets the needs of enterprise developers who are developing software that exploits SOA characteristics and benefits. The OSOA Collaboration is not a standards body; it is a set of vendors who wish to innovate rapidly in the development of this programming model and to deliver specifications to the community for implementation. One of their central working areas is the definition of the *Service Component Architecture* (SCA) [179].

SCA is a set of specifications which describe a model for building applications and systems using a Service-Oriented Architecture. SCA provides a programming model for building applications and systems based on a Service Oriented Architecture but using a componentization of the business logic.

SCA encourages a SOA organization of business application code based on components that implement business logic, which offer their capabilities through service-oriented interfaces called services and which consume functions offered by other components through service-oriented interfaces, called references. SCA divides up the steps in building a service-oriented application into two major parts:

- The implementation of components which provide services and consume other services
- The assembly of sets of components to build business applications, through the wiring of references to services.

The idea behind the SCA initiative is that business functions are provided as a series of components offering their capabilities through service-oriented interfaces, which are assembled together to create solutions that serve a particular business need. These composite applications can contain both new services created specifically for the application and also business function from existing systems and applications, reused as part of the composition. SCA provides a model both for the composition of services and for the creation of service

components, including the reuse of existing application function within SCA compositions.

In addition, SCA is a model that aims to encompass a wide range of technologies for service components and for the access methods which are used to connect them. For components, this includes not only different programming languages, but also frameworks and environments commonly used with those languages. For access methods, SCA compositions allow for the use of various communication and service access technologies that are in common use, including, for example, Web services, Messaging systems and *Remote Procedure Calls* (RPC).

This initiative marks a milestone within the proposals trying to standardize the concepts and features of SOA and its implementation. SCA is the main representative mixing the CBSE scope with that of services. This situation allows taking advantage of all the works that consider the architecture specification from a component-based point of view including architectural styles, platforms and technologies, etc.

2.2.1.7 The Reference Architecture of The Open Group

The Open Group is a vendor-neutral and technology-neutral consortium, whose main role is to capture, understand, and address current and emerging requirements, establish policies and share best practices; to facilitate interoperability, develop consensus and evolve and integrate specifications and Open Source technologies; to offer a comprehensive set of services to enhance the operational efficiency of consortia; and to operate a industry's certification service. Its main goal can be framed in the works that try to enable access to integrated information within and between enterprises based on open standards and global interoperability.

The works by The Open Group in the field of services and service-orientation covers many aspects of the SOC paradigm, including the definition of SOA Ontologies, SOA Governance Frameworks or Service Integration Maturity Models. The interest placed in this consortium is centred in their *SOA Reference Architecture* [220].

Like other proposals for the definition of a reference architecture for SOA, the point of view of The Open Group is intended to support the understanding and implementation of common systems, industry, enterprise and solution architectures leveraging the principles of a SOA. These aspects are materialized in the form of their *SOA Reference Architecture*. In particular, it provides the basis, for a system Architecture but aiming at the enterprise point of view, so an

enterprise architect can use that template or blueprint as a standard that will be instantiated during each individual project or solution that is being developed.

This SOA Reference Architecture is designed to support different kinds of scenarios including those involving consumer organizations, assisting and guiding them in the design and implementation of a SOA by providing a concrete basis for evaluating and making architectural and design decisions; vendors, allowing them to map their specific products to the service architectural models; and other standard bodies and other projects from The Open Group, so they can rely on a concrete context of common SOA understanding model against they can map their related works.

According to that *SOA Reference Architecture*, SOA is an architectural style that supports service orientation as a way of thinking in terms of services and service-based development and the outcomes of services. A service, in turn, is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit; provide weather data, consolidate drilling reports), self-contained that may be composed of other services and that is viewed as a "black box" to consumers of the service.

For the Open Group, the SOA architectural style has the following distinctive features:

- It is based on the design of the services –mirroring real-world business activities– comprising enterprise (or inter-enterprise) business processes.
- Service representation utilizes business descriptions to provide context (i.e., business process, goal, rule, policy, service interface, and service component) and implements services using service orchestration.
- It places unique requirements on the infrastructure – it is recommended that implementations use open standards to realize interoperability and location transparency.
- Implementations are environment-specific – they are constrained or enabled by context and must be described within that context.
- It requires strong governance of service representation and implementation

Apart from defining 9 interconnected layers, The Open Group does not mention in their current works any reference to how services must be created or assigned to each of the layers. Moreover, there is no allusion to the Software Architecture as key artefact in the development of service-oriented systems but as a viewpoint of the topological structure that a system of this kind must have. Since their proposal is framed in the group of *reference architectures* for SOA, the Open Group does not give any particular solution or approach to development processes

based on services, only a common understanding of the SOC terms for the enterprise.

2.2.1.8 Summary

Previous subsections have tried to analyze the current and past situation on what a Service-Oriented system should include and how to represent and model the architecture of this kind of systems from the point of view of several consortia, collaboration groups and organizations. In this big and entangled picture it is important to point out the efforts that have recently come up from three of those institutions: OASIS, OMG and The Open Group. Seeing that all of them, basically where addressing the task of specifying a complete reference architecture for SOA. In [169] these three organizations propose to join their efforts in order to reach to a consensus for a common vocabulary in the scope of Service-Orientation in general and in the field of specifying the elements contained within a SOA environment in particular. Figure 2-1, extracted from [82], shows a global view of the relationships among the different initiatives of these consortia. Collaboration is also been successful between the OASIS consortium and some W3C working groups.

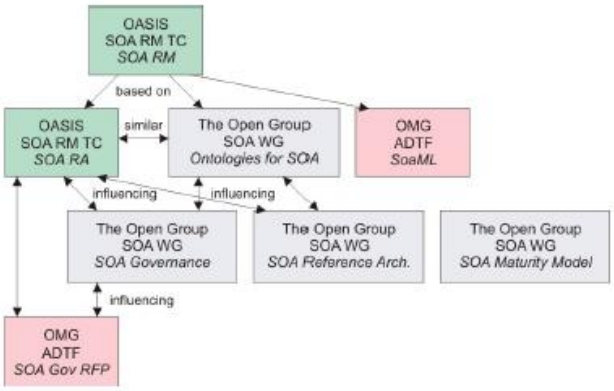

Figure 2-1. Relationship between relevant SOA Open Technical Products.

Table 2.2 shows a summary of the initiatives evaluated in previous subsections according to the criteria established in Section 2.1. This table also includes the decisions taken in the proposed framework (*ArchiMeDeS*) regarding service-orientation and software architecture specification. All the concepts defined by all the analyzed consortiums will be considered in order to define the foundational features of the *ArchiMeDeS* framework either at a conceptual level or at a more technological one.

Table 2.2. Summary of works using Service-Orientation for Software Architecture.

	SERVICE-ORIENTATION			SOFTW. ARCHITECTURE	
	SOA Representation	*Abstraction level*	*Service composition*	*Architectural rationale*	*Base Paradigm*
RA / RM (OASIS)	None	A	Both	N	SO
NEXOF-RA (NESSI)	UML	A	CB	AS	SO
WSA (W3C)	Web Std.	T	Orch.	N	SO
REST (Fielding)	None	T	N	DD	SO
SCA (Open SOA)	Ad-hoc	A	CB	AS	CB
OGSA (OGF)	None	T	Orch.	N	SO
SOA-RA (Open Group)	None	B / A	N	DD	SO
ArchiMeDeS	UML	A / T	Both	AS	SO

2.2.2 Using a Model-Driven Approach for Software Architecture

When putting together the principles behind the Model-Driven approach as software development approach and Software Architecture as key artefact of a system and its influence in the development process, there are several different points of view that can be taken under study. First, it is possible to analyze and understand how the specification of concrete Architectural models may affect to a Model-Driven development process, that is, as it was explained in the previous criteria subsection, answer questions such as *what is the role given to Architecture within the context of an MDD process? Do exist there a common consensus of the Architecture as guiding model? Which should be the real influence of the Architecture in MDA?* Second, just the opposite, it is possible to view the influence of both issues from the perspective of Software Architecture development, that is, analyze how the definition of model sets or the use of a MDD process can help or ease the specification of Architectures. Again, the questions to answer would include: *is there any proposal taking advantage of the benefits (if any) of following a MDD process for that aim? Are models used to cover any other concept associated to the definition of Software Architectures (such as architectural knowledge representation or architectural style definition)?* This subsection is devoted to decipher the ideas behind initiatives of the latter case (as explained in the objectives of this Thesis); however, these proposals sometimes are framed in wider research efforts covering partly the first issue that, in the case of the current work, is also of interest as side research effort (see section 1.2 for objectives details).

To answer the questions that arise in the model-driven development of Software Architectures, it is important to note that, to the best of our knowledge, there are not many initiatives that involve both concepts in a consistent way and simultaneously. Some of these proposals are analyzed in the following subsections.

2.2.2.1 ATRIUM / PRISMA

One of the best representatives among the existent proposals on Model-Driven specification of Software Architectures is the work done by Navarro [158] and the **ATRIUM** (*Architecture Traced from RequIrements applying a Unified Methodology*) methodology [156]. ATRIUM is a methodological design framework for the concurrent definition of Requirements and Software Architecture, defining the automatic/semiautomatic support for traceability throughout its application. It follows an MDD approach so that every stage of the software development process is described by establishing clearly its associated metamodels along with forward and backwards traceability links among them.

The goal of this work is to obtain a proto-architecture of the system instantiating the PRISMA Architectural Model [186] using Aspect-Orientation as base paradigm of the Architecture.

PRISMA focuses on the definition of architectural metamodels for the definition of software architectures using an aspect-oriented approach for that aim. Additionally, it also follows an MDD approach with tools supporting the methodology and defining architecture evolution by means of defining model-to-model transformation at the metamodel level.

Both ATRIUM and PRISMA have a toolset associated allowing the graphical design, validation and transformation of each model in their specification called MORPHEUS [156]. Their transformations are based on the standard QVT specification by the OMG. These works put a special interest in the definition of the principles that leads to the specification of every Software Architecture, that is, the *architectural knowledge*. This aspect converts ATRIUM in a proposal centred in modelling both the design rationale and the design decisions taken by the involved stakeholders, being all these concepts supported by the MORPHEUS tool.

Regarding the criteria explained previously, after analyzing these two joint initiatives, it is possible to state that: the base paradigm of the architecture is *aspect-orientation*; they follow a *model-driven development process* for the specification and evolution (via *QVT model transformations*) of the architecture in which the architecture model plays a central role. The support for *architectural styles* represents an inherent feature of this framework. In addition, it is supported by a *toolkit based on the Eclipse platform*. Though they *define its own language* for architecture description, it uses *UML as graphical notation*.

2.2.2.2 Mattsson et al.

The work by **Mattsson et al**. [139] aims also at the definition of a process for the specification of architectural design rules using MDD techniques. Again, the use of models is not focused on the definition of the architecture itself but on the inclusion of design rules within the specification of the architecture during an MDD process (not used for the architecture but for the development of the system). This last issue does not imply that in their proposal the architecture is not modelled, but that the architectural model does not evolve together with the system. The *architecture follows an independent development process* (using a *product-line* approach) that ends-up in a document describing the main features of the system architecture. Authors defend that using models to represent the requirements and design of the architecture is a good approach, but lack to give specific rules to do so. However, although stating that this should be the way to do

so in a MDD context, they finally create a *textual document* with the architecture specification *with its own syntax and semantics*. The role given to the architecture is, in that proposal, similar to that given in other development processes such as the *Unified Development Process* [99]: they highlight the importance of architectural modelling without specifying a concrete model-driven associated process. In the end, the proposal by Mattsson et al. admits their inability to model architectural design rules, and as a consequence, the Architecture. However, it represents one of the few efforts for modelling architectural design decisions using MDD techniques.

Works trying to align MDD and Software Architecture are more likely concerned with modelling architectural rules, design decisions or the architectural rationale that leads to specify the architecture in a concrete manner rather than with the discussion of how to create a specific process for the specification of the architecture through models. In that research line there are some other related works trying to reason about the modelling of architectural knowledge such as the ones by Tyree and Akerman [5] or Tang, Babar et al. [217] focused on defining ontologies supporting the concepts of that architectural knowledge; there are also works centred on giving tool support for that architectural reasoning such as EAGLE [64].

2.2.2.3 Mikkonen et al.

However, one aspect associated to architectural design decisions and rationale is the definition and inclusion of architectural styles within software architecture. Architectural styles define the rules and vocabulary specific for a common structural and behavioural organization of the system [208]. They have a special interest for the current Thesis as they affect the architectural models of the system. In that direction, works like the one by **Mikkonen et al.** [150] have had an important influence in the research accomplished in the current Doctoral Thesis. They discuss the role of *architectural styles in the context of MDA* proposing to add a specific model to gather all the information related to architectural styles (both vocabulary and constraints as structural pattern elements), called *Architecture Specific Model* (ASM). This model results from the merging of the PIM-level models and a concrete model containing the features of the architectural styles (as roles attached to model elements). As a result, they place this ASM model between the PIM models and the PSM models, obtained by merging both the ASM model and the information of the target platform. The PIM-to-ASM transformation is done semi-automatically by following a process *based on name matching*. In addition, since they decide to base model-to-model transformations in the use of patterns in transformation processes, it obliges the architect to mark

which elements in the architecture take which role indicated in the architectural style. The strategy followed as modelling approach in this initiative is not clearly stated. However, it seems that they inherit the concepts of *UML as notation* for their models without explicitly defining a comprehensive set of concepts for their proposal, since they use *Object-Orientation as base paradigm*, but not a UML profile. However, for the representation of the architecture they define *a formal language* named DisCo, a sort of DSL for architecture specification. As for tool support, they use an *ad-hoc tool based on C++* for modelling and transformation execution. They explicitly state that they follow an *architecture-centric software development process.*

2.2.2.4 Perovich et al.

Other relevant work dealing with the use of MDA for the development of Software Architectures is that of **Perovich et al.** [186]. They start from a separation of architectural views [197] that is modelled throughout the progressive incorporation of quality and functional attributes obtained from the initial requirements specification. Since their proposal is based on the MDA approach, they sequentially define, first, Computation Independent models for the architecture (*Computation Independent Architecture – CIA*); from that models they incorporate the information of well-known patterns, tactics and perspectives to the architecture models so it is possible to obtain models of the architecture from a Platform Independent perspective (*Platform Independent Architecture – PIA*); to finally reach the bottom-most level of the architecture (*Platform Specific Architecture – PSA*) by means of including platform-specific patterns, frameworks and information form COTS (*Commercial Off-The-Shelf*) components into the architectural model. The method for Architecture specification proposed by Perovich et al. uses a *notation based in UML* but without describing which concrete elements are included in each model. The evolution of the architecture description through the MDA process is achieved by means of *ATL transformations*. As explained in their proposal, both design decisions and design rationale behind each architectural concern included in their models is encapsulated in the transformation rules used for model-to-model transformation. The inclusion of architectural styles is not defined explicitly, but its usage may be extracted from the *"tactics and pattern" inclusion in their PIA models.* In addition, their proposal is partly supported by an *Eclipse-based prototype toolkit* allowing graphical modelling and model transformation, though it is, at the moment of writing the current state of the art, an ongoing research work.

2.2.2.5 COSA

A similar approach to that of Perovich et al. is followed in the research works described by **Alti et al.** [7] that aim at the integration of graphical ADL representations using *UML profiles* and *model transformations* in the context of a MDA-based framework. Their proposal, named COSA (*Component-Object based Software Architecture*) starts from the definition of a meta-model at the PIM level of MDA that gathers all the concepts of a component-and-connector traditional architecture and its correspondent UML profile. From the models that conform to that meta-model they allow the transition to platform specific models (PSM-level) by means of the definition of several ATL transformation rules. They support these transformations in a set of plug-ins based on the IBM Rational Software Modeller for Eclipse 3.1 for both the modelling and transformation steps of architecture development. They explicitly state that their work does not include any support for the inclusion of architectural style aspects within their model-driven process.

2.2.2.6 DUALLY

Another viewpoint for taking advantage of a model-driven approach for the specification of Software Architectures is that followed in the proposal by **Di Ruscio et al.** [52]. They start from the premise that, in order to adequately analyze and describe the Architecture of a system, it is possible to follow a notation based on UML. Their proposal is based on the creation of a framework, named DUALLY [98], that allows the specification of Software Architectures following a component-and-connector approach. Inside this framework they have defined a *UML profile* for model notation (though defining the core architectural concepts using a formal ADL, also called DUALLY), a *tool for graphical modelling* support and a development process for Software specification and evolution. DUALLY serves as a way to describe what to model (the core architectural elements) and how to model (via their profile). The framework is completed by a set of mechanisms to extend and adapt the core component-based models of DUALLY to new domains [130]. For that aim, their proposal is accompanied by a rigorous set of *model transformations based on Abstract State Machines*, as a formal and flexible platform on which basing a solution for model transformations. They also use ATL in order to weave the different models during these transformations.

2.2.2.7 Mansurov et al.

The proposals described in the previous subsections follow a top-down approach for the description of Software Architectures. However, there are other

proposals that take the opposite path: building a coherent representation of the Architecture of a system starting from existing code and using a Model-Driven approach. That is the case of **Mansurov et al.** [134], that propose a method for the evolution of existing software assets [134] by means of following the next principles: i) extracting an Architecture Model from code; ii) achieving the sufficient level of abstraction by composition and Architecture Refactoring; and iii) proactive enforcement of architecture integrity to build a *managed architecture* (Software Architecture is considered to be managed when the organization understands the up-to-date, precise and quantifiable situation of the components, their dependencies and configurations [36]). By following an *architecture-centric approach*, they manage to extract MDA's PSM and PIM models from existing code. They achieve that by using reverse engineering tools that, at least from their experience, allow obtaining a component-based representation of the system under analysis. The scope in which this work is framed (enterprises needing to change existing applications) enforce the idea of using a systematic approach based on models that permit the modification and refactoring of a system by means of back an forth *automatic transformations*. The base paradigm used for the architecture is based on components and connectors

2.2.2.8 ArchMDE

ArchMDE [57] is another proposal tackling the model-driven development of software architectures. It is based on the use of a component-based specification of the architecture but taking into account a specific model of the hardware counterpart of the system and also including information from potential architectural styles applicable to the architecture design. Authors propose a complete architecture-centric process based on the use of QVT model transformations for the evolution of the architectural models. No reference is done about the tool support of that proposal.

2.2.2.9 Summary

Table 2.3 shows a summary of the aforementioned works in comparison with the proposal of the current Thesis. In it, together with the summary of the previously analyzed works, the proposal of this Thesis (*ArchiMeDeS*) is outlined with the choices done regarding model-driven engineering of software architectures.

Table 2.3. Summary of works using an MDD approach for Software Architecture.

	MODEL-DRIVEN ENGINEERING				SOFTWARE ARCHITECTURE	
	Model transformations	*Tool support*	*Architecture Role*	*Modelling Approach*	*Architectural Rationale*	*Base Paradigm*
ATRIUM / PRISMA	Y [QVT]	Y [Eclipse]	AC	Hyb.	DD	AO
Mattson et al.	N	N	IA	DSL	DD	CB
Mikkonen et al.	Y [pattern matching]	Y [ad-hoc]	AC	DSL	AS	OO
Perovich et al.	Y [ATL]	Y [Eclipse]	IA	UML	AS	OO
COSA	Y [IBM RSM]	Y [Eclipse]	IA	UML	N	CB
DUALLY	Y [ASM / ATL]	Y [ad-hoc]	IA	Hyb.	N	CB
Mansurov et al.	Y [code analysis]	Y [ad-hoc]	AC	UML	N	OO
ArchMDE	Y [QVT]	N	AC	DSL	AS	CB
ArchiMeDeS	Y [ATL]	Y [Eclipse]	AC	Hyb.	AS	SO

2.2.3 Using both a Model-Driven Approach and Service-Orientation for Software Architecture

Previous sections have shown a brief analysis of ongoing and past works that somehow deal with two of the topics of the current Thesis. However, none of them consider the task of specifying Software Architectures using the principles of the model-driven approach for its development and the concepts of service-orientation as basis for the elements contained in the Architecture altogether. This subsection presents the study of existing works that take into account these three aspects simultaneously. In it, the criteria explained in section 2.1 is used to analyze their main features, their advantages and drawbacks and their relation to the proposed framework. Although most of them refer to research efforts in the scope of European projects, next subsections also include some others coming from the industry or research groups that also work in these topics.

In the case of European Projects, it is significant to remark the importance and influence of the EU Funding activities in the field of *Information and Communication Technologies* (ICT) and, more specifically, the support for research efforts in the scope of service-orientation. Since the 5th edition of the *Framework Programme* (FP) the number of projects related to manage and improve the understanding, acceptance and spread of service-orientation, in all its forms, has been increased greatly, counting more than 50 projects related to that topic in the last edition[1] (7th Framework Programme 2007-2013). This subsection includes the analysis of those projects whose progress best fit with the proposal of the current Thesis: SeCSE, PLASTIC, SENSORIA and a brief remark on NESSI-2010 (the main assets are mentioned in section 2.2.2.2). This subsection ends with the analysis of the works accomplished by IBM in this field and some other independent research efforts in this direction.

2.2.3.1 SeCSE Project

Started in 2002 as one of the numerous projects of the *6th Framework Programme* (FP6, [62]), the main goal of SeCSE [13] is to create methods, tools and techniques for system integrators and service providers and to support the cost-effective development and use of dependable services and service-centric applications. The approach to achieve this goal is based on four different complementary areas: Service Engineering, Service Discovery, Service Delivery

[1] *Project search at ISTWeb: http://cordis.europa.eu/ist/st/projects.htm*

and Service-Centric System Engineering. The topics that may have relevance according to the topics of the current Thesis fall into the Service-Centric System Engineering section of the SeCSE project. This project includes the specification of a system Software Architecture based on a conceptual model defining all the concepts behind a service-oriented view of a system. This conceptual model acts both as starting point for the specification of a global view of the architecture and as common vocabulary supporting the rest of research works accomplished in this project.

As in any SOA-related research initiative, all the terms of the SeCSE conceptual model revolve around the service concept. Among the features of the SeCSE project about this concept there is one that stands out: it clearly identifies that a service must be understood differently according to the domain where it is used. In addition, the SeCSE project remarks the importance of identifying the stakeholders that come into relation with the services appearing in a service-oriented system.

About service composition, the SeCSE project focuses its interest in giving a full support for the definition of complex service environments. To get that, SeCSe states that there are three critical aspects to consider when dealing with service composition: management of the service state (statefulness vs. statelessness), compositeness (simple vs. composed) and, more importantly, the influence that the selection of a concrete architectural style (design strategy for the service composition) may have on the composition [41]. This aspect converts SeCSE in one of the few research initiatives including architectural styles as constituent part of the service architecture modelling.

To support the specification of service composition, SeCSE defines a language based on two different views of the composition: The *service interaction view*, which describes the service roles that participate in the composition and the way these roles interact (interaction model) and the *service process view*, which describes the composition from the view point of a specific role in the composition. Both models are based on the definition of XML documents shared by the participant services on the composition.

Another important aspect in the SeCSE project is the specification and performance of service discovery. In SeCSE three different ways of service discovery are proposed: they call *early discovery* to the inclusion of new services in the service specification during the elicitation of system requirements; secondly, at design time (what they call *Architecture-time discovery*); and, thirdly, all the processes and artefacts needed to allow the discovering of services when the system is running.

2.2.3.2 PLASTIC Project

Similarly to the efforts of SeCSE for defining a common vocabulary for services, the PLASTIC Project [190] (also part of the FP6) aims at providing tools and methodologies to develop service-based applications that are adaptable to the context and able to offer the best tradeoffs between offered Quality of Service (QoS) from the platform and required QoS from users. The interest of this project was mainly focused on developing a software development environment enabling the rigorous development of SLA and resource-aware services, which may be deployed on the various networked nodes, including handheld devices

In order to achieve its goals, the PLASTIC project uses both service-orientation and CBSE as base paradigms to build a layered platform infrastructure. On the one hand, the service layer supports the abstract description of high level functionalities (providing a uniform interface), manages QoS aspects and allows for the dynamic composition of services according to user needs. On the other hand, the component layer represents the computing environment for networked services that is able to manage the lifecycle of the software components while delivering functionalities for a resource from anywhere in the network [14].

It is important to highlight that the PLASTIC project puts a special interest in the management of the context of the system. The PLASTIC proposal include aspects that allow respecting the QoS requests made by the consumer being the context the aspect that drives the component adaptation to the run-time environment.

Both SeCSE and PLASTIC use UML 2.0 and have created tools (based on the Eclipse framework) for the creation of graphical representations of the artefacts they propose to use to build service-oriented software systems. These tools include model editors, non-functional analysis tools and code generation tools.

2.2.3.3 Project SENSORIA

The SENSORIA Project is probably one of the research efforts that best suits the topics of the current Thesis. As part of the 7th Framework Programme, the SENSORIA Project aims at the development of a novel comprehensive approach to the engineering of software systems for service-oriented architectures where foundational theories, techniques and methods are fully integrated in a pragmatic software engineering approach. This project ranges from the definition of formal languages to accurately describe service-oriented architectures (*SENSORIA Reference Model Language* – SRML) [68], graphical DSL languages supporting the graphical representation of service-oriented architectures using extensions of UML (UML4SOA), CASE tools [141], etc.

On the foundational basis of the models delivered by SENSORIA about service-orientation, they fall into the category of the works that use the SCA vision of the SOC paradigm to build service-oriented systems. Starting with the definition of UML profiles for SOA [16][95], the SENSORIA project has specified a complete language (UML4SOA) allowing the description of service-oriented architecture based on components and that supports the specification of complex relationships among services based on service composition. However, that language only supports orchestrations, leaving aside the specification of service choreographies.

Transformations between the different architectural models are achieved by the execution of automatic graph transformations, which are also used to validate the architectural models [88].

The inclusion of architectural styles, is encoded (consisting of the class diagram of the style, a set of graph transformation rules capturing the dynamic behaviour and the model instance for the business application) into a transition system using model-checking techniques. Business application scenarios are formalized using a set of reachability properties and an ulterior validation using simulation over the obtained models [16].

The use of Model-driven techniques applied over the models defined with the profile to generate automatically executable code allow to get BPEL [10][109] code as for orchestrations, and to executable languages such as Jolie or Java.

2.2.3.4 SOMA-ME: the proposal of IBM

From the enterprise world, the works accomplished by IBM in the area of using Model-Driven techniques for the development of service-oriented software architectures present a complete vision supported by development methods, conceptual models, tools and practical experiences.

IBM's proposal is materialized in the form of SOMA-ME, a platform for the model-driven design of SOA solutions. It follows a model-driven approach to transform high-level business specifications to executable code implementing the standards related to Web services technologies.

In order to advance in the definition of a service-oriented system, SOMA-ME proposes to execute model-to-model transformations of different kinds: Refactoring transformations (to perform changes within a model according to different criteria), pure model-to-model transformations (to convert some model specification into a different one according to some external information flow) and model-to-code transformations (to generate executable code from a model). The strategy followed by IBM to perform these transformations is to use an extension of the *Rational Software Architect* tool (RSA). It is important to remark that this

process generates not only new models or code, but also some documentation related to the transformation performed. [12].

The support for architectural styles is an aspect greatly taken into account in the research accomplished by IBM in this field. Their proposal includes methods and techniques to include SOA design patterns obtained from their experience with service-oriented technologies. Additionally, there are works that model not only the architectural styles but also both the design decisions behind the selection of those styles and the decisions placed over the Software Architecture under development [244].

Service composition in the IBM proposal is tackled as part of a wider effort for service integration (*Service-Oriented Integration* - SOI). As a consequence, service composition is not reduced to supporting the modelling of orchestrations and choreographies, but it includes the different trends that currently exist when deploying a service-oriented service in which there exist several services trying to perform a concrete behaviour. In that sense, the concepts of orchestration and choreography are reduced to implementation concerns depending greatly on the technology used. At a more conceptual level, the IBM proposal is based on a layered view of the Architecture very similar to that of the OASIS consortium (IBM is one of the participants in that consortium).

2.2.3.5 Other relevant works

There are other individual works that worth to mention regarding the model-driven specification of service-oriented software architectures. This is the case of the work by **Zhang et al.** [243] that, just like the SCA proposal, considers the modelling of software architectures founded on the conception of service components as first-class modelling entities. Regarding other architectural specification concerns, they do not tackle the possibility of modelling architectural styles or design decisions, which represents, as it was justified previously in this chapter, an important aspect in service-oriented modelling environments. The notation used for their proposal is based on the use of UML profiling techniques comprising all the main identifiable concepts of the SOC paradigm. In that context, service composition is managed through the specification of concrete elements supporting the modelling of service choreographies. No further reference is done on the modelling of orchestrations, the tool support of their proposal or the inclusion of model transformations (they uniquely work at a conceptual abstraction modelling level).

The works by **Zdun et al.** [242], in turn, can be considered as a much evolved proposal according to the criteria established for this state of the art. They define a DSL for SOA together with a set of UML2 profiles in order to be used as

part of a process-driven integration of services based on the MDD principles. In that context, they use a hybrid approach for service-oriented architecture specification. With that aim, they use activity diagrams, class diagrams and component diagrams in order to support the modelling of business processes, message flows and software architecture specification. They claim to use an underlying formal approach for their models based on FCL (*Fuzzy Control Language*). Moreover, they defend the use of tools supporting their development process; however, they lack of additional information of what concrete MDE tool could be used to support their proposal. About the inclusion of design rationale in their models, they defend the use of high-level patterns in workflows that provide guidance on how to assemble the workflow patterns within their models.

2.2.3.6 Summary

As can be extracted from the previous subsections, there are not many proposals that aim at the development of Software Architectures using a model-driven approach (and MDA in particular) and relying on the concepts of the SOC paradigm (using service as foundational concepts for architecture specification). Table 2.4 shows a brief review of what the main projects analyzed offer in the field of research covered by this Thesis. Additionally, this table shows the decisions taken in the proposal of this Thesis (*ArchiMeDeS*) regarding a model-driven and service-oriented approach for software architecture specification.

Table 2.4. Summary of works using both MDD and Service-Orientation for Software Architecture.

	MODEL-DRIVEN ENGINEERING				SERVICE-ORIENTATION		SOFTWARE ARCHITECTURE	
	Model Transformations	*Tool Support*	*Modelling Approach*	*Arch. Role*	*Abstr. Level*	*Service Composition*	*Arch. Rationale*	*Base Paradigm*
SECSE	Y [Graph transf.]	Y [Eclipse]	UML	IA	A	Orch.	DD	SO
SENSORIA	Y [Graph transf.]	Y [Eclipse]	Hyb	IA	T	Orch.	AS	SO
PLASTIC	Y [Graph transf.]	N	UML	IA	T	Orch.	DD	SO
SOMA-ME (IBM)	Y [WSX]	Y [RSA plugin]	DSL	IA	B / A / T	Both	DD	CB
Zhang et al.	N	N	UML	N	A	Chor	N	CB
Zdun et al.	N	Y [ad-hoc]	Hyb.	IA	A / T	Orch.	DD	OO / CB
ArchiMeDeS	Y [ATL]	Y [Eclipse]	Hyb	AC	A / T	Both	AS	SO

2.3 Concluding Remarks

This Chapter has focused on how some essential aspects of Model-Driven Engineering and Service-Orientation are addressed by existing consortia, companies and academia for Software Architecture specification. For that purpose, a coherent set of evaluation criteria was described to analyze them.

The accomplished evaluation determined that most of the proposals providing an architectural approach to service-orientation were related to the definition of reference architectures or reference models. However, it has been detected that most of them do not meet all of the defined criteria.

For example, concerning compositionality, few proposals support all kinds of service composition. Similarly, when considering information about design decisions, not many of them are able to superimpose architectural styles. In addition, most of them consider the definition of the architecture only at one abstraction level (business, technological or conceptual) thus constraining the flexibility of their architectural developments.

Also, it is quite rare to find full-fledged proposals that tackle Software Architecture specification from a Model-Driven perspective and using the principles of Service Orientation. Most of the works analyzed belong either to projects in the *European Framework Programme* or to some efforts from big companies. Nevertheless, the synergies among these three perspectives have not been fully exploited yet. As a result, none of them cover all the features expected in that context, as shown by the evaluation in Section 2.2.3.

Finally, the analysis of model-driven proposals for software architecture specification has shown a tendency towards the use of a hybrid approach as the more adequate choice for that purpose. Hence, the proposal in this Thesis has been also designed using this approach.

The *ArchiMeDeS* framework, presented in this dissertation, aims at covering all these issues by offering a modelling framework which supports, among other features, the architectural specification of any kind of service composition (both orchestration and choreographies) and also at any abstraction level. Besides, it grants the possibility of superimposing architectural style information to architectural models. The objective of this Doctoral Work is to provide with a complete framework that fill in some gaps detected in the state of the art regarding the aspects listed in Section 2.1. By using a hybrid approach, *ArchiMeDeS* benefits from both a UML notation and the independence of creating a DSL for specific goals. Besides, it provides tool support for modelling and model transformation purposes, thus bringing all the advantages of the MDE approach to service-oriented software architecture specification.

Chapter 3:
The ArchiMeDeS Framework

Having analyzed some of the most relevant works in the research fields related to the topics of this Thesis, it is time to present a detailed vision of the proposal outlined by this Doctoral Thesis: *ArchiMeDeS*, a framework for the specification of Software *Archi*tectures following a *M*od*e*l-*D*riv*e*n approach and using *S*ervices and the Service-Oriented paradigm as semantic basis for the architectural models.

In order to make a clear description of *ArchiMeDeS*, this chapter is structured in four sections. The first one starts establishing the decisions taken to build up the framework, their implications and the strategy issued in order to successfully create the aforementioned framework. Next, second section puts the focus on the description of *ArchiMeDeS* as modelling environment for Service Architecture specification. That segment of the dissertation exposes the arguments that allow solving the issues posed in the hypothesis of this Doctoral Thesis by means of describing several DSLs for that purpose. The last section will serve both to show how the framework can be integrated in a concrete development methodology and how it drives the associated model-driven development process. To finish, the final section wraps up some concluding remarks about the proposal.

3.1 General Approach and Framework Development Strategy

Before diving into the explanation of the constituent features that define *ArchiMeDeS*, it is important to clearly set some premises and assumptions that have lead to its development. The issues to discuss include: the reasons behind the choice of Service-Orientation for the description of Software Architectures, the role that MDA may play within the scope of Architecture specification and an introductory discussion on the convenience of defining specific DSLs based on models for the development of Software Architectures.

3.1.1 Using Service Orientation as a Foundation for Software Architectures

The definition given for the Architecture in the previous chapter states that it is developed upon the enumeration, classification and connection of different building blocks compiled under the name "*components*". The term "component" is used there as an architectural element that encapsulates a set of related functions (or data). However, the specification of Software Architectures is commonly based on the use of a concrete paradigm providing the semantic basis for the concepts used within the Software Architecture.

The modern age of Software Architecture (i.e. the developments made during the 90s and onwards) came up within the context of a component-oriented

culture. Therefore, even though architectural components are not exactly like implementation level components, there are obvious parallels between them. Regarding that, the CBSE (*Component-Based Software Engineering*) [216] paradigm can be considered as a close conceptual relative of 'traditional' software architecture specification.

The CBSE paradigm puts the emphasis on the decomposition of the engineered system and aims at realizing software reuse by developing systems combining both pre-existent software artefacts and newly created ones. Though the largely known advantages of using this paradigm for Software Architecture specification and the many languages, tools and implementations associated to it, there are noteworthy drawbacks that suggest using a different approach to build and describe modern Software Architectures. Among them, the one that probably stands out more clearly is the tight coupling that is established when creating composite architectural configurations [119]. This issue constrains the flexibility and adaptability of developed systems, a desired circumstance in current implementations required by modern business processes and needs.

Of course, there are also other possibilities that may serve for the purpose of defining Software Architectures. *Classes* as defined by the Object-Oriented paradigm [199] or *Aspects* in the Aspect-Oriented paradigm [107] may be used too. The interest of this dissertation is centred in the Service-Oriented paradigm and the concept of *Service* in particular.

Accordingly, in this dissertation it is proposed to use the concept of *Service* and its related paradigm, SOC [181], as the conceptual cornerstone for architectural definition. The reasons for that decision are listed in the following:

- **The SOC paradigm can be understood as an improvement of CBSE**. At the core of architecture specification, services can offer the same capabilities as components. They represent computational entities within the system that may be interconnected, offer an interface that exposes the ways of communicating with it and may be composed to define complex structures following established rules and behaviours. Furthermore, services may help to overcome the problem of creating flexible software solutions. These building blocks are considered, in any environment or at any abstraction level, as autonomous entities whose capabilities can be accessed independently their occasional participation in a composite structure. Thus they can be perfectly suitable as a base for architectural definition.
- **Describing service architectures eases the alignment of technology with business models**. As stated previously, current trends in Software

Engineering tend to align high level descriptions of business processes (derived from using BPM [230] approaches and techniques for instance) and needs with the software solutions that are developed to support them [114][59]. Because of that, an architectural description based upon principles and concepts that are closer to that high-level definition may aid to build more accurate solutions. In addition, the evolution of technologies into more distributed environments establishes that current service-based technologies may be applied more easily when understanding the system from a service-oriented perspective.

- **Service-Orientation implicitly involves the definition of complex connection elements.** As businesses become more complex, so does the relation among the participants of their business processes. With the externalization of these processes as part of a company market strategy to improve its revenues, the reliance on distributed technologies has become a crucial aspect. In terms of the Architecture providing support to that situation, this issue is transformed into a need for incorporating complex information policies and trading rules that must be specified when designing the Software Architecture. Since its origin, the Service-Oriented paradigm has incorporated ways of dealing with complex communication means and connection constraints in the form of *service contracts*. Due to this feature, the use of Service-Orientation for Architecture specification represents a step forward over precedent approaches.
- **Services have the capability to remain independent while participating in composite structures.** Though it was mentioned previously, it is important to remark the fact that service orientation offers an added value to traditional architecture representations based on components in the way composite elements are created. Due to its loosely coupled nature, core elements of the paradigm (services) act differently to components, being considered as independent entities even when they take part in a composition. In addition, service composition is based upon the definition of organizational schemes easier to adapt to business strategies than other alternatives. These composition or *coordination* schemes are materialized in the form of orchestrations and choreographies.
- **Service-Oriented Architectures deal intrinsically with architectural dynamism**. Another change detected in current business trends is the high variability of the elements that are consumed for obtaining a desired

result (product). With the partition of businesses and business processes in sequences of tasks to be performed by external entities, the need for adaptation to market needs and tight costs requires a level of organizational dynamism that is also reflected in the computational resources associated to those processes. In terms of architectural configurations supporting these flexible scenarios, it has become almost mandatory that the whole software solution count with an adequate level of flexibility and element exchangeability (plus the need for service discoverability). These features foster a loose coupling between architectural elements which is an aspect deeply bound in the foundations of Service-Orientation from its beginnings.

3.1.2 Using the MDA Approach for Software Architecture Development

The definition of what Software Architecture is, or should be, may vary depending on the field in which this term is referenced. Following the definition given by the IEEE [96], it refers to "*the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution*". According to that definition, the Architecture must contain information about both structural and behavioural features of a system and its environment. The accuracy of such information, its inner components and attributes and the rationale that lead to establish a concrete architectural configuration represent some of the key aspects of any software development. Being the Architecture the artefact that gathers all this features, it is clear that the process that leads to its specification must be carefully observed.

In order to give a proper description of the Architecture many approaches can be followed [39]. Some initiatives address the specification of the Architecture using a **formal approach**. This way, concrete languages are defined to encompass either the architectural design or the associated design decisions (or both). These languages, namely ADLs (*Architecture Description Languages*), may be used to describe either general purpose architectures or may refer to specific domains. Independently of their scope, the describable semantic and features granted to the Architecture are obtained as a consequence of a precise grammar and vocabulary [146]. It is widely accepted that the use of formal languages ends in architectural specifications that enable and ease the analysis of architectural properties such as completeness, consistency or ambiguity of the architectural designs [188]. However, the direct use of formal methods during the development process may

drastically affect to the timing of the development process itself. Not only for the intrinsic restrictions or constraints posed by the use of formal languages but also for the training needed in the use of those languages [193].

In the last few years, however, Software Engineering has shifted its attention to techniques, methods and methodologies in which the system specification does not rely on architectural formalisms but on **modelling approaches** [215]. In that sense, the abstraction level required is closer to the architect's idea of Software Architecture for the system and may not deviate its attention with the use of a complex grammar, tightly tied to strong rules with, occasionally, a concrete and restrictive syntax. The advantage of using models for the specification of architectures arises, among other reasons, due to the possibility of separating the different views of a system, either structural, functional, presentational, behavioural or whichever be the point of view used. Though these ideas have been present in the Software Engineering literature for several years, it has been in the last few years when they have reach a sufficient level of maturity to impregnate some areas of the Software Engineering field, being the Software Architecture one of the most favoured areas [204].

It is important to remark here the difference between the terms *architectural model* and *model architecture*:

- In the context of MDD, the term *model architecture* refers to the structure of the set of models we are dealing with during an MDD process, and it can also be used to refer to the set of models itself. The model architecture is not the generic definition of the structure provided by the MDD definition (in fact, it would refer, for instance, to the distinction between PIM and PSM models provided by the MDA approach), but the concrete set of models we are dealing with at every step and every moment during the MDSD process.
- In contrast, the term *architectural model* refers to the model containing a concrete specification of the system architecture. The reasoning can be seen as follows: the architecture is an abstraction of the structure of a system, therefore an abstract entity. But this entity can be made concrete by means of an explicit representation, which is also informally referred to as "the" architecture; but it has also the classic name of architectural description, usually provided in terms of an Architecture Description Language (ADL). Of course, an ADL is just a particular kind of domain-specific language which provides a certain model of the system. Therefore, the architectural description is also the *architectural model* of the system.

As it was reflected in the state-of-the-art section of this Thesis dissertation, several initiatives have come up using the principles behind model-driven developments to reach a coherent view of the system architecture [133][150]. The framework proposed in this dissertation also follows that approach, and, being more specific, the MDA approach.

To justify the use of the MDA approach for the specification of Software Architectures, the next reasons can be given:

- **It is possible to specify Architectures with models**. A model is used to represent an abstraction of the reality. In the context of Software, it can be used to represent any aspect of a system in which a set of terms define accurately some of its features. Regarding the Architecture, it is possible to properly model the "*components and their relationships*" that build up a system architecture as stated in the aforementioned definition.
- **It makes easier the identification of architectural features by following an approach based on abstraction levels**. MDA defends the definition of sets of models depending on their abstraction level. The level of abstraction given to the architectural representation of a system may range from a global, generic and agnostic view of the system to a more specific one in which implementation or platform dues are included. Consequently, it is possible to state that the MDA can be used to represent different stages in the evolution of the architecture model. In addition, by having separate models, both model and design reuse is also encouraged.
- **It aids to materialize and relate different architectural views**. The use of model-driven techniques for system development and the possibility to define inter-model relationships and transformations among them encourages a specialization of the models reflecting different aspects of the system (*architectural views*). The *architectural model* may be seen as a perspective orthogonal to the development of other concerns during the development of a software system [136]. By having a specific model for the Architecture (and, in fact, also a *model architecture*), the information gathered in that model can help to maintain the interrelation among the data collected by models supporting other concerns. In addition, the architectural model itself can focus, for instance, either on the structural or the behavioural view of the system independently, thus avoiding the interference of other system features (non-functional requirements, implementation of data persistence and data structures, system deployment, etc).

- **There exists an emerging community dedicated to the creation of supporting tools for modelling approaches**. Finally, it is possible to take advantage of existing modelling tools that allow not only the graphical design of models but also checking its conformance to metamodels, the transformation to other representational views and even obtaining executable code specification for a further use.

3.1.3 Selecting the approach for Software Architecture Modelling

Previous sections have analyzed the reasons behind the selection of MDA and the Service-Oriented paradigm for Software Architecture specification. However, as the work is framed within a MDE research context, the strategy for the specification of architectural models is in itself a key aspect that needs a clear answer.

Architectural modelling can be tackled following two different approaches. It can be done either by **using an existent modelling language** (such as UML) and extending it **or** it is possible to completely **design a new language** for specifying Software Architectures. The former alternative is known to end up in the definition of UML profiles. The latter, in turn, would require defining the corresponding grammar and vocabulary together with an associated notation (almost) from scratch. In the following paragraphs both alternatives are presented together with a discussion on them in order to justify the final approach adopted: a hybrid solution based on the creation of a new DSL with a UML-based notation.

3.1.3.1 Representing Software Architectures using UML profiles

The UML language has long proved its potential to represent almost any view of the system needed during the software lifecycle. Through the specification of different types of diagrams, containing both static and dynamic information, the use of UML has spread to impregnate many areas of Software Engineering. One of the reasons behind this success is its adaptability to represent concepts and features of specific domains thanks to the extensibility means provided. In that sense, the term *UML Profile* is used to define a package containing model elements that have been customized for a specific domain or purpose using extension mechanisms, such as stereotypes, tagged definitions and constraints [175].

A *UML Profile* usually contains a set of stereotypes that extend the meaning of the core elements of the superstructure of the UML Standard. These derived concepts serve to create a new corpus of terms that may be useful to define the principles of specific domains. By defining a particular group of

stereotypes it is possible to build a DSL to be used in those particular domains instead of using the general concepts of UML.

New stereotypes are created in a way that they extend, restrict or override the meaning of an existing UML meta-class over which the stereotype is used, together with associated attributes (representing tags for tagged values), operations and constraints. The profiling mechanism proposed by the last version of UML (2.1) allows the **specification of several views of the same model by the simultaneous use of different profiles on a common base UML model**. Obviously, the stereotypes must complement the information gathered in the model and be non-excluding. Taking as example the information represented with a class diagram it would be possible to use a UML profile that add performance information to the model while using, in the same model, information about security aspects. Both added-values could be analyzed independently by a tool aware of the characteristics of any of the profiles used. In addition, and due to the independency required from the UML Profiles applied, it would be possible to hide or show the information given by one or another profile without affecting the underlying UML model in any way.

The use of UML profiles for the specification of concrete software issues may **take advantage of the numerous tools and expertise** that have flourished since its conception. Most of the current tools that support UML modelling allow the specification of customized stereotypes and thus make them available to be used over concrete diagrams.

In subsequent revisions of UML, the notion of *profile* has been refined in order to provide more structure and precision to the definition of stereotypes and tagged values. Thus, the UML2.0 infrastructure and superstructure specifications have carried this further, by defining it as a **specific meta-modelling technique**. Stereotypes are specific meta-classes, tagged values are standard meta-attributes, and profiles are specific kinds of packages [175].

It is worth mentioning that the **first MDE methodological proposals adopted UML profiles as modelling language**. Since UML was the *de facto* graphical modelling language of choice, they opted for extending it to support the abstractions considered in their proposals. For instance, the most recognised proposals for Web Engineering, like UWE [110], MIDAS [35] or OO-H [88] were (initially) based on the use of UML profiles.

3.1.3.2 Creating DSLs for Modelling Software Architectures

In the field of Software Engineering, a *Domain-Specific Language* (DSL) is a specification or programming language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution

technique. The concept isn't new—*special-purpose languages* and all kinds of modelling/specification languages have always existed, but the term has become more popular due to the rise of domain-specific modelling.

The opposite to this kind of languages is:

- A general-purpose programming language, such as *C* or *Java*,
- Or a general-purpose modelling language such as the *Unified Modeling Language* (UML).

Therefore, DSLs are languages tailored to a specific application domain. They offer **substantial gains in expressiveness and ease of use compared with general-purpose languages** (GPLs), such as UML, in their domain of application [138]. In other words, while a DSL is designed to solve a delimited set of problems, GPLs are supposed to be useful for much more generic tasks and thus they cross multiple application domains. Also, a GPL aims to provide with a way to represent abstractions from any particular domain. A given DSL provides means for **expressing concepts derived from a well defined and well-scoped domain of interest** [104], such as the ADLs in the context of Software Architecture specification. Furthermore, the rules of the domain can be included into the language as constraints, disallowing the specification of illegal or incorrect models. Examples of DSLs range from the *Structured Query Language* (SQL) [99] to the regular expression language used in Linux utilities such as `sed` or `awk`.

Since the architecture field of research uses its own set of concepts, restrictions and features, it can be therefore considered as a whole and complete domain In that sense, the use of DSLs can be also traced within the Software Architecture research field. In that area of interest, **specific languages for architecture description have been defined under the acronym ADL** (*Architecture Description Languages*) [146]. There are a vast number of languages that, from one point of view or another, have tried to support the representation of software architectures. Among them it is worth to mention Acme [83] (developed by CMU), AADL [66](standardized by SAE), C2 SADL [146] (developed by the University of California-Irvine), Darwin [129] (developed by the Imperial College of London), Wright [6] (developed also by CMU), π-ADL [180] or PiLaR [43]; even using a MDE approach for that aim like [52]. Having a closer look at the elements that build up the previous languages, it is clear how they define a grammar based on components and connectors, provide with a strategy to represent architectural configurations and, so they can be considered as truly usable and useful, some of those initiatives are accompanied by a more or less complete tool support.

3.1.3.3 Discussion: The Hybrid Approach

The preferred strategy for modelling Software Architectures, upon which the current Thesis rests, has been reached by gathering a consensus between the use of both UML and the definition of a new DSL: the idea is to **create a new DSL containing the relevant concepts for architecture description** (based on the SOC paradigm) **and give a UML graphical notation for that DSL** (by defining the corresponding set of stereotypes for UML in a UML profile). This is what will be understood as *hybrid approach* from now onwards.

Although in the beginning there was a notable trend towards extending UML as a way to define new DSLs, a close analysis of the evolution of initiatives in the field reported that some years later UML profiles were not taking off. As a matter of fact, quite a lot of methodological proposals based on MDE were initially based on UML profiles. However, when researchers started to develop the technical support for their proposals, drawbacks such as inheritance of undesired features from UML or the inclusion of distracting elements in the models, became more apparent. As a result, those **proposals, originally based on the use of UML profiles, moved towards the use of DSLs**. This is the case of the already referred MIDAS or UWE. We can find even works that use UML profiles as a formal way of specifying their proposal, but uses MOF-based DSLs to deploy them [143].

Because of the fact that the core concepts of the architecture are founded on the principles of the SOC paradigm, the use of any general modelling language may not be suitable for that aim. Hence, from an inner perspective, the creation of a DSL for modelling service-oriented software architectures appears more obvious instead of the option of extending the UML concepts through a UML profile. Following this strategy, the **core principles of the SOC paradigm are maintained without any semantic interference** of the underlying description language. To maintain this independency and isolation, the newly created DSL will be based on MOF, a standard language for specifying metamodels, i.e. a meta-metamodel language.

An additional reason to choose the DSL approach for architecture modelling can be analyzed from the perspective of model management. According to the OMG, the organization behind the standardization of UML, UML models can be made persistent (i.e. *stored*) by using the XMI standard [176]. However, analyzing the current tools and initiatives that give support to this language, it is significant that, though trying to cope with a standard specification, each of them uses a different version of XMI. This poses an additional challenge when trying to deal with UML models from start as it interferes in the interoperability usually marketed by UML. The use of a DSL for the architectural models overcomes this

problem since **it is possible to manage model persistence in personalized XML formats without depending on current XMI implementations**.

This situation about the storage of the models leads to discuss the convenience of using one or another toolkit supporting the architectural modelling. Due to the relevance of this aspect in the current Doctoral Thesis, this aspect it will be further analyzed in the next subsection.

Despite all the benefits of using a DSL at the low level of the architectural specification (for semantic foundations), at a higher level of modelling (practical use of models) the preferred approach may differ. Since the purpose of *ArchiMeDeS* is to provide with a framework easing the specification of architectures, the DSL itself needs to be accompanied by a well-known graphical notation and an appropriate modelling environment. The use of UML for modelling purposes is widely spread so it makes sense that **the graphical notation of choice be UML**. In that sense, the direct definition of a UML profile would have eased that part of the framework creation but, as stated before, would have kept some undesired features associated with UML. The solution to that stage of the architecture modelling steps on the creation of an UML profile that provides with a UML-based notation to the DSL created.

All things considered, the *hybrid approach* can be seen from two different viewpoints. At the inner level of specification, the plan to obey is to start defining the intrinsic characteristics of the DSLs for Software Architecture with services. At a higher level of specification and after the metamodel has been refined, to port these concepts into the UML notation by defining concrete stereotypes and tagged values to be used in different UML diagrams.

To sum up, Table 3.1 shows a comparison between the approaches analyzed for Software Architecture modelling.

Table 3.1.Comparison of modelling approaches for Software Architecture specification.

	MODELLING WITH UML PROFILES	MODELLING BASED ON DSL	HYBRID MODELLING
Graphical notation	UML-based	Not predefined	UML-based
Tool support	UML-based tools	Ad-hoc tools	Ad-hoc tools
Suitability for architecture	Same as UML	Adequate	Adequate
Semantic independence	Low	High	High
Expressiveness for SOC	Not defined	Adequate	Adequate
Ease of use	Well-known	New knowledge	Hybrid
Acceptance	Widespread	Increasing	Increasing
Ease of creation	Easy	Complex	Complex
Model storage	XMI-based	Model persistence	Model persistence

3.1.4 Selecting a Tool Support and Modelling Environment

From the previous subsection it can be extracted that the supporting modelling framework must cope with the definition of the features of customized DSLs and give support to a visual representation with UML. As any other language, the DSL with the foundations of service architectures must include two different syntaxes: *abstract syntax* and *concrete syntax* [104].

- The **abstract syntax** contains the vocabulary (concepts) and relations among the terms in that vocabulary. Since the aim of the DSL is used to define models, and the proposed framework is based on the MDA approach, the abstract syntax will be represented trough the specification of MOF-based metamodels compiling all the DSL terms.
- The **concrete syntax**, in turn, is the way models are depicted and visually (or textually) specified. In the scope of the current Thesis, and again following the standards related to the MDA approach for modelling, the concrete syntax is set upon the use of extensions of the UML language. Hence, the corresponding UML extension elements are confined into UML profiles containing the needed stereotypes that fully correspond with the terms of the aforementioned metamodels.

The goal of reducing the complexity of the architecting process explicitly requires the definition of a tool. As stated before, it is highly recommendable to provide with a set of tools that ease not only the modelling process but also that may serve as validation and model-checking tool. This way, it is possible to ensure the coherence and conformance of the created models to the principles governing the DSL created.

A deep analysis of the convenience of using concrete tools and a further discussion of what would be the most adequate for the scope of MIDAS was largely considered in the PhD Thesis of J. M. Vara [227]. As a consequence, this work will inherit its conclusions and, therefore, select the outcomes provided by the *Eclipse Modelling Project* (EMP) [93] as main support framework to achieve the goals of the current Thesis. Next paragraphs briefly explain the key aspects that motivate its selection and preference over other modelling environments.

With the spread of MDE ideas and principles, the number of related technologies and supporting tools has proliferated greatly. Many software companies and research groups are considering the development of their *own* tool for supporting their *own* MDE method (following the MDA, Software Factories, Product Lines, Generative Programming of whatever other more specific model-driven proposal). Facilities provided in the context of the *Eclipse Modelling Framework* (EMF) and other DSL frameworks, like the *Generic Modelling Environment* (GME) or the *DSL Tools* of Microsoft, have shifted the focus from simple UML-based drawing approaches to more complex MOF-based modelling and metamodelling ones.

In the context of EMP several facilities have come up to aid developers in the tasks surrounding a model-driven process: EMF for metamodel specifications, meta-editors like GMF, transformation engines like ATL or VIATRA, code generators like MOFScript, etc. As a consequence, model engineering proposals are developing their tools as Eclipse plug-ins, like the OOWS suite [224] and M2DAT [227], or at least, upgrading or re-defining them to be "Eclipse compliant", like WebRatio [238] or ArgoUWE [110]. This dissertation follows also this approach. The details have been included as part of Chapter 4.

Other reason for having chosen Eclipse as tooling support is that the DSLs proposed in *ArchiMeDeS* can be represented easily using *Ecore models* (the Eclipse version of the MOF standard) within the Eclipse modelling framework as it will be shown later on.

3.1.5 Selecting the Target Platform for Service Architecture Modelling

The last decision to point out before starting to explain the main contents of the *ArchiMeDeS* framework is the establishment of the target platforms that will be used at the lower levels of the architectural modelling, that is, at the PSM level of the MDA proposal. The platforms of choice will be those supporting the technologies of **Web Services**, **REST-compliant Web Services** and **Grid Services**.

To justify this choice is important to have a look at the evolution of MIDAS, the methodological framework in which *ArchiMeDeS* is framed. Originally conceived as a methodology for the development of WIS (*Web Information Systems*), MIDAS has evolved towards a more service-oriented methodological viewpoint [36]. From that perspective, the target implementation platforms have broadened its scope ranging from Web-based to other technologies and platforms in which services play a prominent role. *ArchiMeDeS*, as integral part of MIDAS, aims at covering the gap of the architecture modelling within that framework. For this reason, **the initial implementation platforms to consider should be those using a service-oriented approach**. Since the modelling approach followed by MIDAS, and also by *ArchiMeDeS*, is the MDA proposal, the target platforms will be contemplated at the PSM level of abstraction. It is at that level where the features of the concrete implementation alternatives have a direct influence on the architectural models.

It is important to remark here that, because of being based on the MDA approach, though the architectural modelling at this level (PSM) follows a service-oriented approach, the target platform may not be based on a service-oriented technology or, being so, have a service-oriented basis but not follow any of the current service technologies. As it will be shown later on this dissertation, this issue will be favoured by the fact of separating the PSM level in two: an upper level with the commonalities of any service-oriented platform (PDM) and a lower one gathering the particular features of service technologies. In that sense, it is key factor to differentiate the paradigm used within the models and the target platform represented with those models: Web Services, REST Services and Grid Services

To be more precise in this platform addressing it is important to mention the success that **Web Services** have gained within the enterprise. Its *interoperability capabilities*, the promotion in the *use of standards*, the *number of programming languages* providing primitives for its support or the fact that communications among services are performed over *the HTTP transfer protocol*

are some of the numerous benefits that have given rise to the consideration of Web Services as technology of choice.

Once *ArchiMeDeS* has the capacity to represent the architecture of Web Services, in order to expand the number of platforms supported, several other existent alternatives may be considered. In parallel with the development of Web Service standards, some variations of their principles have been suggested to either overcome some of the drawbacks detected or increase the functionalities initially drawn for that technology.

The first additional platform can be identified reading directly the Web Service standards delivered by the W3C. In them, the consortium offers a vision of Web Services in which "*two major classes of web services can be identified: **REST-compliant Web services**, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of 'stateless' operations; and arbitrary Web services*[1]*, in which the service may expose an arbitrary set of operations*" [236]. According to this definition, the support for **REST services** [69] must be included within any proposal trying to drive the architectural modelling of a service-oriented platform (based on Web Services). In the corresponding section it will be studied in detail whether the support for this kind of services requires a modification of the DSL for Web Services or not.

Among the benefits usually given to REST-compliant services are: the use of a stateless client/server protocol, a *set of well-defined operations* applied over all kinds of resources, a *universal syntax* for accessing to resources (based on their URI) and the reliance on *hypermedia as the engine of application state* (through the use of XML or HTML to represent it).

Another view of how service-orientation could be deployed and implemented can be traced in the principles behind the Grid Computing paradigm and the ideas of **Grid Services** usage. From the beginning of the Grid Computing it has fostered the creation of *Virtual Organizations* with the aim of being capable of creating distributed environments in which use of shared resources can be used to reach a common goal [75]. With that aim, the Grid community shifted their attention from low-level ad-hoc implementations to the use of services, widely known as Grid Services [178]. This way, they could connect distant loosely coupled, heterogeneous and geographically dispersed resources acting in concert

[1] *These 'arbitrary Web services' refer to the standard conception of Web Services.*

to perform very large tasks. The definition of OGSA and WSRF, as normative architectural definitions of what a Grid should be and include; and the specification of standard languages such as WS-Resource or WS-Management, have contributed to the success of Grid Computing.

In the end, the Grid Computing field of research has tended to constitute an alternative to massive distributed computing using services as preferred low-level technology. Moreover, in the last years it has served as basis for other upcoming initiatives such as the Cloud Computing, being the consideration of the *on-demand* term the main distinction among both initiatives.

The main difference between standard Web Services and the conception of service done by the Grid Computing paradigm lays in the definition of the concept of *resource*. In a Grid environment, a special interest is placed on *the incorporation and management of the state* to both the communication means and the implementation of the service itself. This distance from the Web Service standard has motivated the definition of concrete languages to support that paradigm (such as WS-Resource [91]). In addition, the idea of *Virtual Organizations* fostered by the Grid Computing paradigm represent a concept not initially considered by traditional Web Service platforms or those based on REST services. Other distinguishable features include the importance of enabling resource virtualization or on-demand provisioning [192].

To recapitulate, the target platforms to be modelled at the PSM level are set to Web Services, REST-compliant services and Grid Services. Figure 3-1 shows an overview of the targeted service-oriented platforms.

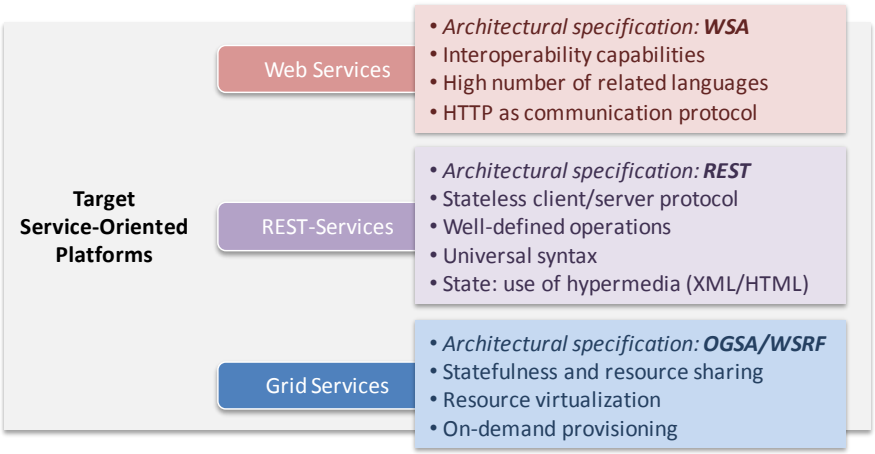

Figure 3-1. Target Service-Oriented platforms of choice.

The implications of developing software solutions with Web, REST and Grid Services are the starting point to build up service architectures at the PSM level. However, from that premise, it will be possible to grow and extend the architectural modelling to other related platforms. The ultimate goal is to build a DSL at the PSM-level general enough for modelling not only service-oriented technologies but any kind of execution platform. Further support for other platforms and technologies will be more clearly specified in the *Future Works* section later in this dissertation.

3.2 Modelling Architectures with ArchiMeDeS

Previous section has stated the design decisions taken for approaching the development of *ArchiMeDeS* as Service-Oriented Framework for the Model-Driven specification of Software Architectures. Recalling the ideas, it is founded upon the definition of DSLs following the principles of MDA and SOC, it provides a UML-based graphical notation for that DSLs and focuses its attention on Web Services, REST-compliant services and Grid Services at the implementation level. Figure 3-2 shows a global overview of the metamodels defined for each abstraction level as well as the proposed model-to-model transformations.

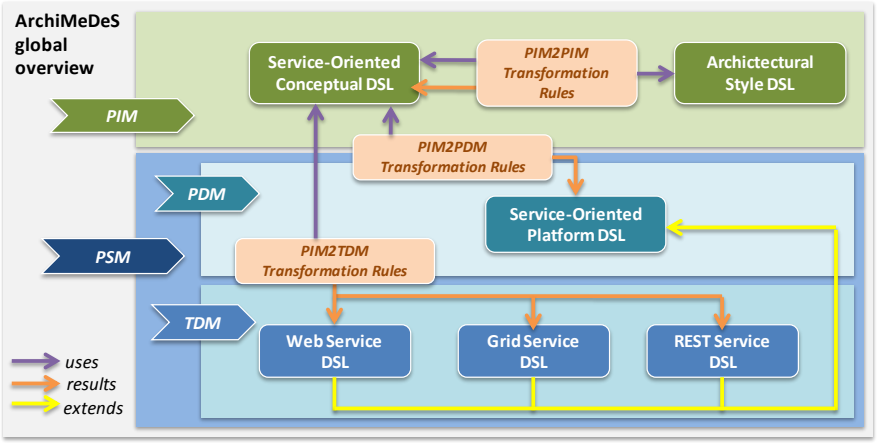

Figure 3-2. Global overview of the ArchiMeDeS framework.

It is important to highlight here the explicit model separation done at the PSM level. This level has been divided into PDM (*Platform Dependent Models*) and TDM (*Technology Dependent Models*). Reasons for that division are summarized in the following:

- PDM models collect all the commonalities that all service-oriented technologies may share. This way it is possible to provide a common model vision that is, in essence, complete and independent from the underlying service technology, thus facilitating platform migration.
- Additionally, the separation in two different levels also fosters the extensibility of PDM models, so it is possible to support other service-oriented platforms somehow away from current service technologies.
- Finally, the constant evolution of the underlying service technologies has derived in a series of changes in their related implementation language and standards. By and large, these changes have not affected to the basic principles of these platforms. To illustrate this, it is possible to look at the Web Service standards evolution in the last years, to observe the evolution of Grid platforms from OGSI [177] to WSRF [91], or the coming up of REST services as a different service technological approach. Despite these changes, the underlying principles of service-oriented platforms have remained (almost) unaltered. The PDM model is, therefore, the same, independently of whether the TDM is modelled, for instance, to represent past OGSI Grid architectures or the current WSRF-based Grid approaches.

The following subsections describe the elements needed to harness the architectural modelling process with services. Sections 3.2.2 and 3.2.3 describe the abstract syntax and semantics for both PIM and PSM levels respectively. At the end of each of these specifications, the graphical notation for each DSL using a UML profiling technique is exposed as concrete syntax. Finally, Section 3.2.4 gathers all the model to model transformations defined within the *ArchiMeDeS* framework. It is worth to mention that the transformations do not only refer to the evolution of the architectural specification from PIM to PSM (section 3.2.4.2) but also to the inclusion of architectural styles as PIM-to-PIM model transformations (section 3.2.4.1) as part of the PIM modelling process.

3.2.1 PIM Architectural Specification

If the definition of Software Architecture [208] is merged with the MDA modelling conception at the PIM level of abstraction [170], the result is *a conceptual representation of a system that comprises all the components that build up the software solution without any implementation implication or restriction, identifying just the building blocks, their attributes and the*

characteristics defining the relations established among them. Having that premise in mind, in the following, a complete DSL for that aim is presented.

To model the system architecture with this DSL, several sources of information have to be taken into account. As it will be later explained (Section 3.3), this information is primarily obtained, among other sources, from behavioural models defined as an independent concern of the related methodological framework [49]. However, it is important to remark that the architectural model comprises mainly structural information about the system.

3.2.1.1 Abstract Syntax

The first aspect of the PIM level DSL to describe is its abstract syntax, in which the underlying semantic is defined by means of describing the concepts supported and the relationships among them. When using a MDSD approach for creating DSLs, the most common way of representing those concepts is by means of a metamodel plus a set of restrictions obtained from the analysis of the domain [171]. Accordingly, the metamodel that contains all the concepts for the service-oriented architectural DSL at PIM level can be seen in Figure 3-3. After it is clearly explained, the restrictions that apply are stated.

3.2.1.1.1 Service Providers

Generally speaking, a Service-Oriented Software Architecture is built upon independent entities which provide and manage services. Because of the fact that service-orientation is widely used as a way to integrate enterprise applications, the existence of these providers is justified by the necessity to project the business organizations involved in those processes into the architecture model. These providers act as service containers in charge of presenting the services contained to the world. In comparison with traditional architecture descriptions, these elements could be considered as the service equivalent to the *package* concept but without the implication of requiring an explicit interface definition.

A closer study of these days' business trends shows that a software solution does not represent an isolated piece of software. When observing the structure of the business context in which a system is normally placed, external required software entities may emerge so that it is possible to perform specific tasks involved in the business processes. As a consequence, these outer entities must be somehow reflected in the architectural description, together with the way an inner element of the system communicates with it.

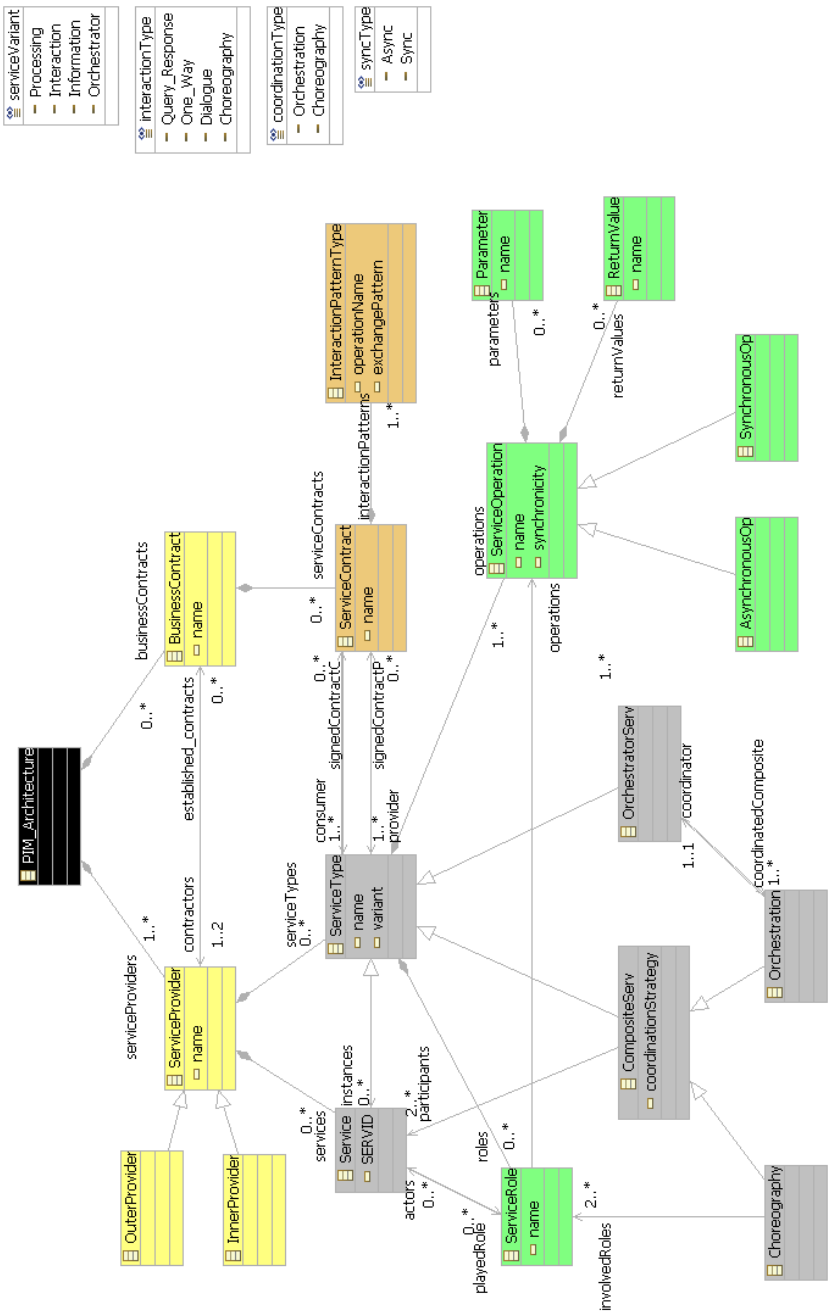

Figure 3-3. Metamodel associated to the PIM DSL for Software Architectures.

Taking this dichotomy under consideration, service providers can be classified in two main groups:

- **Inner service providers** (*innerProvider* in the metamodel), which are internal organizations to the system designed. They can be also understood as the part of the software solution whose inner elements are being designed in detail.
- **Outer service providers** (*outerProvider* in the metamodel), which are identified as external entities containing services which collaborate with the system to perform a specific task of value for the system but which are not under its control or whose internal structure is not known or valuable for the development of the current architecture model.

The relationship between two service providers appears in the moment that a business need arises and the services involved in its resolution belong to each of them. The existence of a potential communication is understood as a business dependency between communicating providers. Because the interconnected elements represent business entities, the relation among them is understood as a '**business contract**' (*bussinessContract* in the metamodel) that represents a sort of communication between the services belonging to the providers. The content of a business contract is understood as the set of service contracts that can be signed between those services.

This element can be found in the bibliography under several other names (with slight differences in concept or role) such as *container* [4], *service manager* [114], *market-maker*, *service aggregator* or *service provider* [181], *context* [242] or *service owner* [51].

3.2.1.1.2 Service Description: Identity, Operations and Roles.

According to the SOC principles [181], services support the functionalities offered by the system and derived from both the requirements stated by the user and the information gathered in the upper-level models. The vision of the service concept in *ArchiMeDeS*, at the PIM level, is aligned with that of the OASIS reference model for services [165]. A **service** is, therefore, *the atomic artefact, within the architectural model, that allows the practical support for the system features and business processes identified in higher abstraction layers during the development process*. In addition, services are understood as computational entities in charge of a resource (whichever its nature or features be: data storage, statefulness, processing capabilities, etc.) and which offer access to that resource in the form of a well-known interface.

The main elements that allow the description of a service at the PIM level of abstraction are: a unique identifier, named **SERVID**; a **set of operations**, exposing the functionality associated to the service; and the possibility to define a subset of these operations to be used under concrete conditions, i.e. a **service role**. With these three elements it is enough to clearly model a service within a concrete architectural configuration. Next, these properties are explained in detail:

- **SERVID property.** Due to the fact that a service has to be searched, located and reached, it must be identified. This is accomplished by defining the SERVID property as part of the service description. That property allows placing a service within an architectural configuration and, therefore, to identify a service univocally within a SOA environment represented with the PIM-level model.

  Another reason to include an identifier inside the definition of a PIM model is that the architectural model represents **instances of services** (and not 'classes' as in usual object-oriented approaches in which class diagrams are used). At a lower level this identifier will be transformed in a concrete address or implemented identifier.

  To enforce this idea of modelling service instances, the concept of service can be further analyzed from two different perspectives: from a global point of view and from a more concrete and detailed one. From a *concrete* point of view, the idea is to represent the actual services that entail the functionalities offered. This way, they must be considered and clearly identified as individual services (hence talking about 'service instances'). This issue is gathered in the metamodel of the DSL under the **service** concept. From a more general or *global* point of view, in turn, it should be needed to model the functionality that is expected to be provided by some service in the architectural configuration, without identifying a concrete service exemplar. In that context, it is possible to use the term **service type** (*ServiceType* in the metamodel) to gather all the operations any potential service occurrence may provide.
- **Service operations** are considered as atomic functionalities, offered by a service, that collaborate to build a joint description of the service. Moreover, they represent the only way of interacting with a service as they outline its interface. Their properties include their **name** and the eventual **parameter**(s) and/or **returned value**(s) they may need to fulfil its functionality.

  It is important to note the distinction between operation types depending on its *synchronicity*. In **asynchronous operations**

(*AsynchronousOp* in the metamodel) the requester of the operation does not wait for the answer or return value (if any). In turn, in **synchronous operations** (*SynchronousOp* in the metamodel) the requester will wait for the answer or return value (that should always exist). The reason behind identifying the synchronicity of an operation is related to the influence it may have in the moment of defining the information flow associated to the tasks needed to perform a business process.

- **Service role**. When a service acts following a concrete **role**, its base behaviour is modified to adapt to the context in which the service participates. In that sense, the service offers a subgroup of the operations initially defined in its service type. That is, a service may be able to perform a certain set of operations, but it is only able to provide some of them according to the assigned role in a certain moment. To reflect that feature within the architectural model, a service role will be comprised by an identifying **name** and a **subset of the operations** that the service playing that role is able to perform.

  At this point, it is worth to note how the concept of service as building block of the architecture is modelled separately from the roles played in a concrete architectural configuration. In that sense, service roles could be easily paired with the traditional idea of *interface* as it exposes the set of operations that conforms the external view of a service and which are the only available ways to communicate with the service when acting following that role. This aspect is further explained in the moment of considering the composition means among services within the architecture.

3.2.1.1.3 Service Interaction: Contracts and Interaction Patterns.

The main difference between the concept of service (as understood within the *ArchiMeDeS* framework and the SOC paradigm in general) and other architectural paradigms (Object-Orientation, CBSE, Aspect-Orientation, etc.) relies on how interaction is achieved among architectural elements. First, observe the fact that **services are consumed** by interacting agents instead of being instantiated or encapsulated to reach a concrete goal. Second, instead of using method invocation on an object reference, service orientation shifts the conversation to that of **message passing** (though not explicitly considered at this level of abstraction). Finally, interaction is based on the tacit agreement to act in a concrete manner gathered under a **contract signed** by the participant services. To include these features in the architectural models, this part of the dissertation is

focused on the definition of the concepts related to service interaction: **service contracts** and **interaction patterns**.

- **Service contracts.** Services relate, communicate and interact with each other according to agreed **contracts** [59] (represented by *ServiceContract* in the metamodel). In the architectural description of the service models, these 'contracts' are understood as *connectors*, specifying point-to-point relationships between the services that 'sign' those contracts. Accordingly, these contracts serve to specify the adhering conditions under which contracted services agree to communicate. Contracted services take the roles of *provider* and *consumer* depending on whether the contractor is the one offering the functionality or the consumer of it.

  Observing the traditional architectural conception of 'connector' [208], these contracts might also specify the channel used to communicate and exchange messages; however, at the abstraction level that service-oriented PIM models represent, the inclusion of this aspect in the DSL would force the specification of a concrete standard or technological support mean. Therefore, as it is considered to be platform-dependent information, it will be consigned to the PSM level of abstraction. Nevertheless, the main properties of a contract, apart from its **name**, are the **interaction patterns** gathering all the allowed interactions among service contractors. Policies and other aspects related to the support for non-functional requirements within the architectural models (QoS, security, etc.) have been left out intentionally in the development of the current Doctoral Thesis and will be subject of future research efforts.

  Finally, service contracts must be understood as the concrete conditions under which the business contracts specified among service providers take place.
- **Interaction Patterns**. These interactions are defined as pairs '*operation name-interaction kind*'; being the former element a reference to the set of operations offered by the provider service and the latter the kind of exchange pattern that will be followed when using that operation. Since service-orientation is based on the idea of *service consumption* as the way to access the functionalities provided by a service, the number of service contracts established among two services will depend on the business interest each service has in order to fulfil its assigned responsibility. For example, if two services need to access to the operations of the other, two contracts will be signed, each of them exposing the available operations

and interaction pattern needed to consume the operations offered by one and another service.

An important aspect to comment is the fact that, though in most architectural configurations services will expose and sign a single contract with each service desiring to use the facilities provided, the DSL created has enough flexibility to support configurations in which services sign different contracts. In those situations, the number of available operations agreed or the exchange pattern agreed may vary depending on the service contractors or even, when contracting the same service consumer, during the lifetime of the system.

The kinds of patterns for message exchange are reduced to four alternatives. '*One-way*', in which no response is expected when an operation request is made; '*Query/Response*', in which there is an explicit answer to the operation requested (note that it can be sent either synchronously or asynchronously); '*Dialogue*', in which the concrete protocol can be complex and, therefore, it should be accompanied by an additional state machine; and, finally, '*Choreography*' which will be used in scenarios where several services agree to collaborate to reach a common goal (see *service compositions* for more details about choreography modelling).

This explicit behavioural description in the form of interaction patterns must be respected by both requesting and providing services.

3.2.1.1.4 Service Composition: Orchestrations and Choreographies.

As it has been stated previously in this dissertation, a feature that considerably differentiates SOA from other architectural alternatives is the way elements are composed. Composition within SOA is accomplished by means of an *interactive collaboration among its participants*. These collaborations are not tied to a single way of composition by direct interface-to-interface connection as it happens in a typical CBSD configuration; in contrast, it is possible for several services to build a complex interaction scheme based on different criteria *without constraining the independence of the composed elements*. This aspect is a great advantage when developing hybrid architectures in which services can participate in different parts of a system. Another distinctive aspect of service composition is that *the compound element can be seen as a new service* available in the system environment, thus allowing to scale the system functionality by aggregating ("*composing*") these services into larger composite applications, and making the outputs available for consumption by any business user.

The classification of service composition alternatives (identified under the *CompositeServ* element in the metamodel) is done according to the **coordination scheme** used to build up the composition. In that sense it is possible to define service **orchestrations** and service **choreographies**. The modelling of any of these two kinds of composition establishes different levels of detail that can be used when developing architectural models. In that sense, it will be possible to model an element, identified as a composite, comprising the capabilities that may provide as service or it could be feasible to model the inner structure of the composite by identifying the participant services.

- **Orchestration**. This kind of coordination is founded upon the idea of having a special service in charge of directing the whole compound element. This special service (marked as *OrchestratorServ* in the metamodel) knows the flow of service consumptions that must be done in order to accomplish the functionality desired. In that sense, the orchestrator service acts as the enactor of a workflow by means of a coordinated interaction with other services.
- **Choreography**. The other possible way of combining the capabilities provided by each service with the aim of reaching a desired asset is by considering an interacting environment among equivalent services. This "equivalence" means that there is no service mastering over any other or directing the flow of information. The evolution in the system architecture is accomplished by the changed produced in the roles played by each participant.

  Recalling the principles behind the PIM DSL defined by the *ArchiMeDeS* framework, it is important to remark that, when talking about element compositionality, the models defined with that DSL represent only structural information (concrete elements building up the system) and reflect the structural consequences in the architecture that may imply the participation of an element in one or another compound structure. In the case of service choreographies the architectural influence is reflected in the behaviour performed by the participating services. In particular, it affects to the operations they provide in each moment and, hence, the role played.

  The operations that a service provides in each moment are defined as part of the **role** that a service plays within the context of a choreography. As it was explained previously, a **service role** represents the interface offered by a service comprising the available functionalities to be used when using that concrete interface.

3.2.1.1.5 Service Taxonomy: Information, Interaction, Processing and Orchestration variants.

The principles behind service-orientation clearly set the service as central artefact around which any other concerns revolve independently of the nature of the service (service contracts, operations, composition, etc.). However, from the architectural point of view, a more detailed classification is recommended to *ease the identification of the responsibilities* each service may have to realize the business processes the systems aims to cover. An additional reason to reflect service variants within the architectural models is that, at the PIM level of abstraction, developers already know which will be the main function of a service as part of the software solution. These 'knowledge' is architectonically understood as *a design decision agnostic of the underlying technology* or platform and, therefore, it might be included as part of the PIM DSL. Finally, the identification of several sorts of services facilitates the specification of what elements are needed at lower levels of the architecture, that is, by discriminating the purpose of a service at the PIM level *it is possible to know which elements must be present at the PSM-level* modelling of the system architecture.

At this point, there are four clear variants that can be identified in a service-oriented architectural specification: **information**, **interaction**, **processing** and **orchestration**. In the PIM DSL defined by *ArchiMeDeS* all these variants are associated to a service in the form of a feature (*variant* in the metamodel) comprising one or more of these values. A brief explanation of these variations is presented in the following.

- **Information.** A service whose main purpose is to serve as access point to stored information, independently of the storage strategy chosen, will have assigned the *information* value to the variant attribute.
- **Interaction.** Services may also serve as boundaries of the system. In that sense, services will be identified with the *interaction* value. These kinds of services usually expose special behaviours depending on the external element that wants to take advantage of the functionalities provided by the service. For example, when the external actor is a human being, the service responding to its needs may require complex interaction means (based on dialogues) either synchronously or asynchronously.
- **Processing.** Other kind of service that may be easily identified is that in charge of performing information processing.
- **Orchestration.** Finally, an important service variant within the architectural model is that in charge of handling the flow of the business process defined for the system.

By identifying different service variants at PIM level we support the automation process advocated by the MDA approach as it will be easier to separate them when describing what services belong to the platform (services performing communication tasks) or to upper implementation levels (discovery services), which of them will be comprised of a physical resource granting the access to a service (for example a Web interface allowing to access a concrete Web Service), or even the contrary, a service with the capability to access a physical resource (for instance when a service acts as the access point to a storage facility).

3.2.1.1.6 Domain restrictions

Apart from defining a service metamodel for software architectures, the creation of a DSL is completed with the enumeration of a set of restrictions that may apply when using the DSL. These constraints are summarized in the following:

- Operation names that are paired with interaction patterns in a service contract must appear as operations defined by the service type of the service acting as provider in that service contract.
- Services taking part in a choreography must explicitly define what service roles play within that choreography.
- To give a complete vision of the architectural model, it is possible to define state machines diagrams associated to each interaction defined as 'Dialogue'.
- Contracts established within choreographies contain interaction patterns for all the operations that offer the service types involved. The exchange pattern for each operation is defined, by default, as *Choreography*.

3.2.1.1.7 Additional considerations

- **On the modelling of pre- and post-conditions.** It is possible that for two services to communicate they fulfil some pre- or post- conditions related, for instance, to non-functional requirements. In the context of this Thesis these clauses have not been included; however, they could be easily represented as restrictions associated to the *serviceContract* element and using OCL for its description.
- **On the modelling of messages.** Messages, in the scope of the DSL presented, would represent the communication item exchanged between services. Each message (or, more properly, *message type*) should be related to a service contract. The most important feature of messages is

that they have to be understood by both the emitter and receiver services, so the attributes of this concept should include references to the ontology model or semantic constraints involved in the message.

Although the semantics associated to the message exchange appear in the architecture model (in the form of an *interaction pattern*), the message format does not. The format of a message is an issue that depends mostly on the implementation technology and so it should be modelled in the correspondent PSM-level models.

- **On the modelling of service state.** Though many of the references found in the bibliography understand services as stateless entities [59][114][127], this is an aspect that depends mostly on the service implementation technology and so, at PIM level, no reference on the state should be modelled explicitly (as part of the service description).

3.2.1.2 Concrete Syntax

Once the terms that conform the semantics of the DSL that allow modelling the Architecture at the PIM level of abstraction have been exposed, it is time to provide a way to give a visual notation that ease the use of that DSL. As it was discussed in section 3.2.1, the option selected is the definition of a UML profile containing a set of stereotypes that permit the use of UML as base graphical notation for the DSL. To illustrate this, Figure 3-4 shows a UML 2.0 profile diagram with the corresponding stereotypes. It shows the actual details of the profile, joining each stereotype with its *meta-class* using the extension notation (filled arrow head).

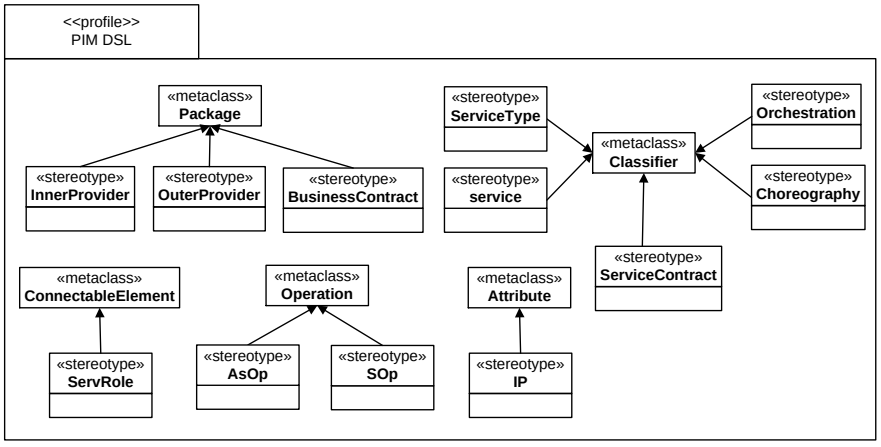

Figure 3-4. UML Profile for the PIM DSL.

The UML diagram chosen to represent the architectural model is the Class diagram. The main reason behind this decision is that it allows representing a static view of the system whereas composing a structural perspective of the system. The stereotypes defined have associated the closest concept that may be found in the UML metamodel.

3.2.1.3 PIM DSL Summary

Table 3.2 shows a summary of the terms that build up the DSL created for modelling Software Architectures at the PIM abstraction level. In that table it is depicted the concept, the associated semantics and the graphical stereotype used to represent it with UML. In addition, the derived meta-class used and any restriction posed over the concept are depicted.

Table 3.2. Summary of concepts and stereotypes of the PIM DSL.

CONCEPT	SEMANTICS	NOTATION		RESTRICTIONS
		BASE UML METACLASS	STEREOTYPE	
Service Provider	Representation of a business organization	Package	---	---
Inner Provider	Service Provider controlled by the system	Package	<<InnerProvider>>	---
Outer Provider	External provider of services	Package	<<OuterProvider>>	---
Business Contract	Business relationship among providers	Package	<<businessContract>>	---
Service Type	Global description of a service	Classifier	<<ServiceType>>	---
Service	Subset of the system functionality	Classifier	<<service>>	Must have a SERVID
Service Contract	Agreed contract among services	Classifier	<<ServiceContract>>	---
Service Operation	Atomic functionality provided by a service	Operation	---	---
Synchronous Operation	Operation. The consumer waits for the response of the operation	Operation	<<SOp>>	---

CONCEPT	SEMANTICS	NOTATION		RESTRICTIONS
		BASE UML METACLASS	STEREOTYPE	
Asynchronous Operation	Operation. The consumer does not wait for the response of the operation	Operation	<<AsOp>>	---
Interaction Pattern	Pair operation name-exchange pattern. Indicates how an operation is consumed	Attribute	<<IP>>	The operation name must belong to the provider service
Service Role	Subset of the functionality that played by a service.	Connectable Element	<<ServiceRole>>	Define a subset of operations
Composite Service	Aggregation of services	Classifier	---	---
Orchestration	Composite Service. A service governs the information flow in the composition	Classifier	<<Orchestration>>	Must contain an Orchestrator
Choreography	Aggregation of services in which none masters over another	Classifier	<<Choreography>>	---

3.2.2 PSM Architectural Specification

Models at the PSM abstraction level, as described in the MDA proposal [170], should reflect aspects related to a concrete technological platform (i.e. the influence of using a specific programming language, features of communication layers, the use of a database or an concrete implementation framework, etc.) and closer to the final coding of the system. The number of platforms currently available entails that any modelling initiative for that level must be able to cope with the potential diversity of implementation alternatives.

In the case of the *ArchiMeDeS* framework, at this level of abstraction, a DSL for modelling software architectures **using services** is encouraged, as it was discussed at the beginning of the current chapter. To reach that goal, different platform aspects that may affect the architectural configuration of service-oriented solutions should be taken into account. For instance, the technological standards followed (e.g. WSDL, WADL, WSRF, SOAP, etc.) or the execution environment

(single computer, clusters of workstations, network of distributed resources, etc.) supporting the system under development.

Regarding the topics of this Thesis a special mention should be made on some of the most spread *approaches that foster the use of services as underlying supporting platform.* In particular, some of the currently most relevant service platforms should be taken into account: the **Web Service platform** (through the support for both the WSA initiative [228] and REST-compliant services [69]) and **Grid Computing platforms** (following the OGSA initiative [178]). Next subsections cover the definition of a DSL for modelling architectures of these service-aware platforms.

Though these platforms define and treat the architectural elements differently, a common set of concepts can be extracted. This leads to the creation of a **base metamodel** that can be used as kernel for any service-oriented platform as it is explained in section 3.2.2.1. As a direct consequence, the PSM level of the MDA architecture is divided in two separate levels: PDM (*Platform Dependent Model*), which gathers all the concepts that the target service-oriented platforms have in common, that is, a DSL with a 'core' metamodel; and TDM (*Technology Dependent Model*), which will comprise an extensions to the PDM DSL adapted to the features of the three technologies chosen, i.e., Web Services, Grid Services and REST Services.

After the PDM DSL is specified, three extensions at the TDM level are presented corresponding to the support for architecting with Web Services (section 3.2.2.2), Grid Services (section 3.2.2.3) and REST-compliant services (section 3.2.2.4). Afterwards, the concrete syntax is presented in the same way as it was made for the PIM level, indicating accurately the concepts used for each technology.

3.2.2.1 PDM Abstract Syntax

To detect the architectural commonalities shared between different service platforms, some **premises** about the nature of this kind of environments should be stated:

- The **environment** in which services are executed is *inherently distributed* and, thus, concepts such as 'public interface' (participating elements do not see the internal workings, implementation details, or resource representations of other elements), 'communication mean' (the channel used for the communication among services sets up an exchanging protocol) or 'executing agent' (distributed environments connect computational elements in charge of executing the services) play a prominent role in service-oriented architectures.

- **Communication** among services is *based on message exchanges*. This implies that both the exchange protocol and the format in which messages are delivered represent a key aspect of service interaction. Though not interfering in the structure of the system configuration, this information may be helpful when obtaining the final coding of the system.
- **Functionalities** offered by services refer to the *control of a resource* for which they act as interface and access mean [59]. That resource can be either a physical storage resource (file, database, etc.) or a more 'ethereal' one in the sense that it may refer to a processing functionality or the persistence of a state between invocations. Likewise, resources may allow act as entry points for the use of services within architectural configurations.
- **Basic features** of services include the fact of being *publishable, discoverable and reachable*. As a direct consequence, they must offer a well-known interface and, therefore, the concept of 'location' is mandatory.

All these aspects represent information from the implementation environment that have an influence over the system architecture, and thus modify the semantics that were gathered at the PIM level models. Having all these issues in mind, and after studying the initiatives WSA (for Web Services), OGSA (for Grid Services) and the principles of REST (for REST-compliant services), it is possible to provide a coherent description of a set of *common concepts* that allow building up a DSL for modelling the PDM level of the architecture: **service agents**, **resources**, **services** and **service contracts**. These concepts can be seen in the metamodel shown in Figure 3-5. Afterwards, they are further explained together with their properties and attributes.

3.2.2.1.1 Service Agents: supporting infrastructure for services and resources.

Services and resources in a service-oriented environment must be executed and managed by some physical entity. At higher levels, this responsibility fell on the business organizations identified in the business context and reflected, accordingly, in the PIM architectural model. At lower levels, in contrast, this job is accomplished by physical entities globally known as **service agents** [228].

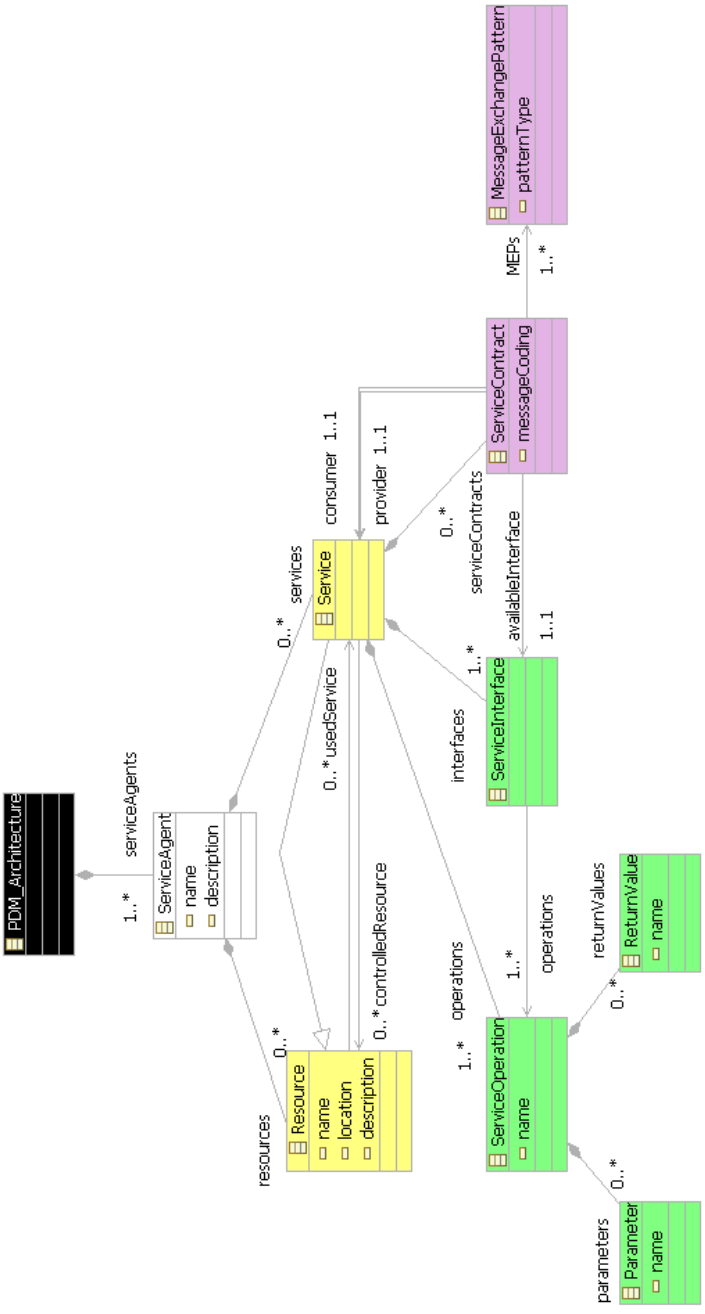

Figure 3-5. Service-Oriented Core Metamodel at the PDM abstraction level.

Service agents may be understood as the elements that serve as the factual executing substrate of the services implemented over it. This concept allows the encapsulation of many of the concerns needed for service consumers to access to a desired and concrete service with regard to service discovery, security credentials, intelligent caching of time-sensitive data (e.g. token for a concrete message pattern), as well as invocations of the service of course.

Architectonically speaking, service agents may be paired with the concepts of *node* or *server*, in the sense of acting as deploying infrastructure over which services are executed and resources are positioned. The main difference with those elements is that the identification of service agents within architectural models may not refer uniquely to hardware components but, in contrast and from the point of view of the *ArchiMeDeS* framework, they may also represent software artefacts allowing for the performance of the tasks provided by a service under its charge.

Essentially, when an application needs to use functionality provided in a service, a service agent manages the semantics of communicating with that particular service. For example, the business components of a retail application could use a service agent to manage communication with the credit card authorization service, and use a second service agent to handle conversations with the courier service. Service agents isolate the particularities of calling diverse services from applications, and can provide additional services, such as basic mapping between the format of the data exposed by the service and the format your application requires.

Properties of service agents include, apart from its **name,** the identification of what services and resources are under the responsibility of the managing service agent.

3.2.2.1.2 Resources: identifiable architectural entities.

According to RFC 2396 [19], a resource can be *anything that has identity*. This general definition covers any electronic entity that may be computationally described, managed and accessed [228]. Therefore, it is possible to identify as *resources* elements such as a Web page, an image, a text file or a database storing information, but also a processing facility, a computing platform (in the sense of an execution substrate) or any software artefact usable by any external entity in order to obtain a value derived from its interaction with the resource. Following that definition it is easily acceptable the consideration of any *service* implementation (based on whichever the current service-related technology be used) as a special kind of resource.

From that conception it is possible to derive the properties that a resource has: a **description** (in the form of meta-information about the resource and its attributes and capabilities), a **location** (a piece of information that allows its addressing and discovery) and, of course, a **name**, allowing to identify it uniquely within the architectural configuration. Depending on the technology or paradigm used to implement the resource all these distinguishable properties will be defined jointly (think of a REST-compliant service for example) or separately in different software elements (e.g. a URL plus a WSDL for the description of a Web Service).

3.2.2.1.3 Services: active architectural elements.

As it has been pointed out previously, a service, at a lower abstraction level, is a special kind of resource. Consequently, it inherits all its properties; however, it differs from a resource in its purpose as part of a software solution. A service may be clearly distinguished due to the fact that it acts as an *active* entity in charge of performing a specific function in its context, either on its own or jointly with a specific resource. This way, a service represents a piece of software that performs part of a business task clearly defined and implemented over a concrete platform.

About the possible relationships between services and resources it is important to remark that there are situations in which the consumption of the functionalities offered by a service requires the previous use of a resource (for example when a Web page allows the access to a specific Web Service). The existence of this kind of contexts requires an explicit description of the relationships allowed between resources and services:

- **From service to resource**. Access may take many forms, including retrieving a representation of the resource, adding or modifying a representation of the resource (in some cases may change the actual state of the resource if the submitted representations are interpreted as instructions to that end), and deleting some or all representations of the resource (which in some cases may result in the deletion of the resource itself). In the DSL, these particularities are reflected in the form of a dependency, named 'control' established as a property of service elements.
- **From resources to services**. In these cases, since resources are considered to be 'passive' elements within the architecture, these elements will provide only a standard way of referencing and accessing the capabilities defined by the service (e.g. a link to the service entry

point or URL). To that end, the access means will depend mostly on the service implementation technology and the agent in charge of managing the resource (i.e. interpreting the information inside the resource for accessing the service, for example, the factual invocation of a URL found in a HTML Web page addressing a concrete Web Service). In the core PDM DSL, these particularities are reflected in the form of a 'use' association established between resources and services and following that direction.

All things considered, the relationship between services and resources may be classified from two different points of view: services in charge of allowing the access to a concrete resource (a database for example) or static resources allowing the access to a service (such a Web Page that acting as interactive interface to access the features provided by a service). Additionally, services may act as independent entities whose assigned task does not entail any interaction with a resource.

Apart from the properties derived from the condition of being resources, services, as architectural elements, must define another concrete property: a well-known **interface**, which represents the boundary of a service alike the resource description element, with the difference that, in the case of a service, a *service interface* contains a description of the offered **operations**.

3.2.2.1.4 Service contracts: defining architectural interaction.

Interaction in service-oriented environments is accomplished by the active elements, that is, among services, through the exchange of messages (i.e. tokens of information). The conditions under which messages are exchanged are defined in an agreement signed by the contractor services. The architectural element that defines all these conditions is known as **service contract**.

At the PIM level of abstraction, a service contract represented the conditions under which some client could use the operations defined by a service. At the PDM level, in turn, these conditions may define not only the *message interaction pattern* but, more consistently, the *protocol* used to encode the communication process, the *message format* used or the concrete *interface* of the provider service that will be able to use the consumer. This situation can be easily tracked within platforms such as Web services (where a WSDL file includes both the message exchange pattern and a reference to the scheme that messages follow, apart from defining the interface of the service itself) and Grid Services (where standards such as WS-Addressing [232] or WS-Coordination/WS-Transaction [32][34] allow the definition of the interconnection conditions at a given moment).

Apart from identifying the **contractors**, relevant attributes of service contracts include the definition of the technology or standard used in the **message coding** (for example, when implementing with Web Services, the contract should clearly indicate that the SOAP protocol will be used) and the **message exchange pattern** (MEP) that will be followed. At PDM level, the values assigned for those will retain the values defined for the PIM level since the concrete interaction pattern depends greatly on the language used for describing the service contract.

3.2.2.1.5 On service types and its architectural relevance.

Services at the PDM level refer to computational entities that have their own responsibility and capabilities within the system scope but, apart from the implementation technology, do not differ architectonically from any other service in the system. Accordingly, the kinds of services identified at the PIM level (Information, Interaction, Processing and Orchestration) are justified by the need for the identification of element patterns within the architecture at the PSM level.

That means, for example, that when modelling an *information service* at the PIM level of abstraction, at the PSM-level there will exist a service in charge of allowing the retrieval of the information related to of a persistent resource (independently of the form it has, whether it is a 'logical' state or a table in a database) or, focusing on the boundaries of a system, that if the PIM-level model reflects the existence of an *interaction service*, the PSM-level counterpart will include a static resource granting the use of the capabilities provided by a service.

3.2.2.1.6 On service composition.

Composition of services at this level of abstraction depends greatly on the technology chosen to develop the composition. Further comments on composition means will be tackled in deep in next subsections since there are different languages defined to implement composition strategies based on orchestration (BPEL, BPEL4People) or choreographies (WS-CDL, WS-CAF, WS-CF, etc.)

3.2.2.2 TDM Abstract Syntax: Web Services

In order to properly define a DSL for service-oriented platforms, the PDM metamodel presented previously must be concreted to some of the most spread platforms that are used nowadays having a special emphasis on the technologies used. Under the **Web Service** concept lays a technology that has reached the inner structures of companies and that has been pointed out as one of the most promising technologies for the implementation of software solutions within the enterprise [114]. Behind this technology there is a myriad of languages and standards defined to optimize its usage [236]. The W3C is the consortium that

leads this standardization initiative and which is under charge of developing those specifications.

Focusing on the architectural viewpoint, the W3C is also responsible for deploying the WSA initiative, a proposal for the definition of both the functional components and the relationships among them that may arise within in an environment of this kind [228]. In addition, since the WSA specification recommends the use of WSDL for the description of Web Services, the WSDL specification will be used as basis to properly define each of the architectural properties building up the PSM level of the *ArchiMeDeS* proposal for Web Service Architectures,.

It is important to remark that this subsection does not pretend to give a deep explanation of the WSA standard (as it can be consulted in [228]) nor the WSDL standard, but, in contrast, an approach on how these specifications fit within the *ArchiMeDeS* framework in the form of an extension for the core metamodel that allows specifying software architectures at this abstraction level.

The metamodel of the extension for the WSA proposal can be seen in Figure 3-6. It also shows the relationships defined between them and corresponding elements of the core metamodel. In the following, as it was done with the PIM-level version of the DSL, these concepts will be clearly explained and exemplified with the Web Service implementation of the *GESiMED* system.

3.2.2.2.1 Resource (*WSResource*)

Although resources, in general, can be almost anything, the WSA architecture is only concerned with those resources that are relevant to Web services and therefore have some additional characteristics. In particular, they incorporate the concepts of **ownership** and **control**: a resource that appears in this architecture is a 'thing' that has a name, may have reasonable representations (resource descriptions) and which can be said to be owned. The ownership of a resource is critically connected with the right to set policy on the resource (not contemplated within the scope of this Doctoral Thesis). The *control* property, in turn, refers to the service agent that is in charge of managing the resource and rules the policies that govern its access. It may correspond with the owner of the service or not.

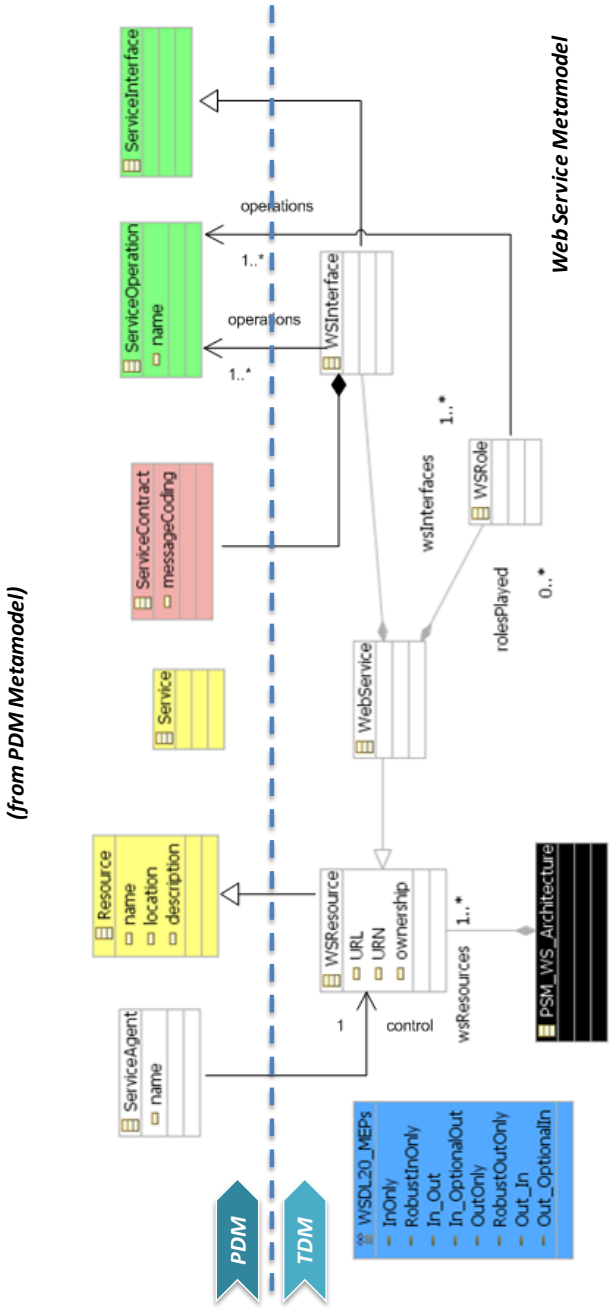

Figure 3-6. Concepts for modelling Web Services at TDM abstraction level.

In accordance to what was defined in the core metamodel of the PDM level, the WSA specification avoids making any explicit reference to the activeness role of that resource, relegating it to the concept of Web Service. It is worth to mention that the Web Service element is also considered to be a special kind of resource as it shares the same descriptive properties: **name**, **location** and **description**. However, in either a *WSResource* or a Web Service, these properties are specialized to comply with the standards of the W3C, as follows:

- The **location** property identified in the kernel version of the DSL becomes a **URL**, allowing the addressing of the resource.
- The **name** property (inherited from a resource) becomes the **URN** as it is defined in RFC2141 ("URN Syntax").
- The **description** property remains unaltered just like it was defined in the core metamodel.

3.2.2.2.2 Web Service (*WSService*)

The main concept of the WSA specification is the *Web Service*. According to WSA it is *an abstract resource that represents a capability of performing tasks that represents a coherent functionality from the point of view of provider entities and requester entities.* This element inherits, in the *ArchiMeDeS* proposal, directly from the *Service* concept of the core metamodel, sharing all its properties and associated relationships with other elements. Since a Web service is considered to be a kind of resource in WSA, so it will inherit its properties from the *WSResource*.

The **description** of a service, as it is understood by the WSA, is a set of documents that serve to describe what a service is and which is the information that allows communicating with him. In that sense, and so it is described in the WSA standard, the service description is matched with the union of the service interface and a semantic description of the abstract functionality provided by the service. The current extension for Web Services does not include any kind of architectural support for semantic descriptions. As it was explained in the introduction, this end was the subject of a previous PhD Work by Cesar Acuña [2] though no integration has been done to date.

3.2.2.2.3 Service Interface (*WSInterface*)

As it is defined in the WSA specification, the interface of a service is the abstract boundary that a service exposes. It defines the *types of messages* and the *message exchange patterns* that are involved in interacting with the service, together with any conditions implied by those messages. The definition of

exchange patterns inside the interface implies the definition of the conditions under which a service allows communications, thus making reference to the *service contract* it may establish with potential service consumers. As a consequence, in the DSL definition, the service interface element keeps a reference to the corresponding service contract. Further details on this element are specified next.

However, that definition of interface does not include the set of operations through which the service receives invocations and offers its functionalities. In contrast, the language recommended by WSA for the description of a Web Service (WSDL) already includes explicitly the definition of the operations offered by a service[2].

Due to this semantic inconsistency between WSA and WSDL, in the extension proposed for modelling architectures of Web Service-based environments, the concept of **service operation** is preferred to be part of the service interface architectural element leaving the specification of the message exchange patterns to the service contract element.

3.2.2.2.4 Service Contract

As it happened to the service operations, no explicit reference to the service contract is made inside the description of the WSA proposal. However, current implementations of Web Services [103][114][214] consider that the service contract, signed between a service and any potential clients that may take advantage of the functionalities offered by it, relies in the acceptance of the features and constraints defined as part of the service interface. That is, inside the WSDL document that is publicized showing the ways a service may use to interact with another. More complex contracts may be signed in environments not relying tightly to the W3C standards and proposals. In that sense, the service contract may appear mandatory after a handshake process is done between requester and provider.

Since this service contract inherits its properties from the core metamodel, it defines the *message exchange pattern* indicating the handshake protocol that

---

[2] *In WSDL, the service interface is defined under the <<portType>> element of WSDL 1.1 and under the <<interface>> element in a WSDL 2.0 document. It contains a description of the operations that can be used and the input/output messages that will be used with each operation.*

agreed by both the consumer of the service and the provider[3]. However, in case of using the standards recommended by the W3C (i.e. WSDL), the allowed exchange patterns are reduced to 8 possibilities (according to the version 2.0 of that specification [234]):

- **In-only**: Here a service operation only receives an inbound message, but does not reply. This MEP cannot use a fault. When referred to by an operation's pattern attribute, it has the value "http://www.w3.org/ns/wsdl/in-only".
- **Robust In-only**: Identical to *In-only*, except that this type of MEP can trigger a fault. When referred to by an operation's pattern attribute, it has the value "http://www.w3.org/ns/wsdl/robust-in-only".
- **In-Out**: Identical to the *request-response* of WSDL 1.1. A fault here replaces the out message. When referred to by an operation's pattern attribute, it has the value "http://www.w3.org/ns/wsdl/in-out".
- **In-Optional Out**: Similar to *In-Out*, except that the out message is optional. When referred to by an operation's pattern attribute, it has the value "http://www.w3.org/ns/wsdl/in-opt-out".
- **Out-Only**: The service operation produces an out-only message, and cannot trigger a fault. When referred to by an operation's pattern attribute, it has the value "http://www.w3.org/ns/wsdl/out-only".
- **Robust Out-Only**: Similar to *Out-Only*, except that this type of MEP can trigger a fault. When referred to by an operation's pattern attribute, it has the value "http://www.w3.org/ns/wsdl/robust-out-only".
- **Out-In**: Identical to the *solicit-response* of WSDL 1.1. An operation sends a request and then waits for a response. Here the <*output*> would be defined before the <*input*>.When referred to by an operation's pattern attribute, it has the value "http://www.w3.org/ns/wsdl/out-only".
- **Out-Optional In**: The service produces an out message first, which may optionally be followed by an inbound response. When referred to by an operation's pattern attribute, it has the value "http://www.w3.org/ns/wsdl/out-opt-in".

---

[3] *In WSDL, contract features and constraints (message exchange protocol and transport mean) for each service operation interface are described inside the* `<<binding>>` *element of the WSDL document.*

All this variants will be used within the architectural model to fill in the *patternType* value of the *MessageExchangePattern* element from the core metamodel.

3.2.2.2.5 Service role (*WSRole*)

The service role played by a service refers to the performance of choreographic interactions among services using some of the standard languages defined for that aim by the W3C. These standards for choreographies are far from being stable and much discussion remains alive within the Web Service community ([17], [89], [153], [181]). To reach the objective defined in this Thesis; this element is included so as to identify the subset of operations used by a service when participating in a complex choreography.

In contrast, the WSA gives a different point of view of this term, defining it as *the relevant abstract set of tasks provided by a person or organization offering a service.* This definition is at a higher abstraction level than the one given previously since it refers to the tasks offered by a service owner instead of assigning that responsibility to the service itself. For the sake of usability, in this concrete case, the concept of service role adopted will be the one explained in the previous paragraph due to the current trends in choreography languages and in order to maintain the coherence with the PIM architectural modelling.

3.2.2.3 TDM Abstract Syntax: Grid Services

In order to be able to model Grid-based architectures, the Grid Service metamodel will extend the concepts gathered by the Web Service DSL since the foundations of the Grid Computing paradigm are currently based on the Web Service technology.

The actual use of Grid Services as *stateful* Web Services (see section 2.1.8 for more details) and its particularities serves as main reason to propose a concrete extension of the PDM metamodel for service-oriented architectures. Therefore, in the following, the concepts identified within the Architecture of Grid environments (as defined in both the OGSA Architecture [76] and the WSRF specification [91]) based on Grid Services are described in detail. To have a global overview of these concepts and how they relate to the Web Service metamodel defined previously Figure 3-7 shows that extension.

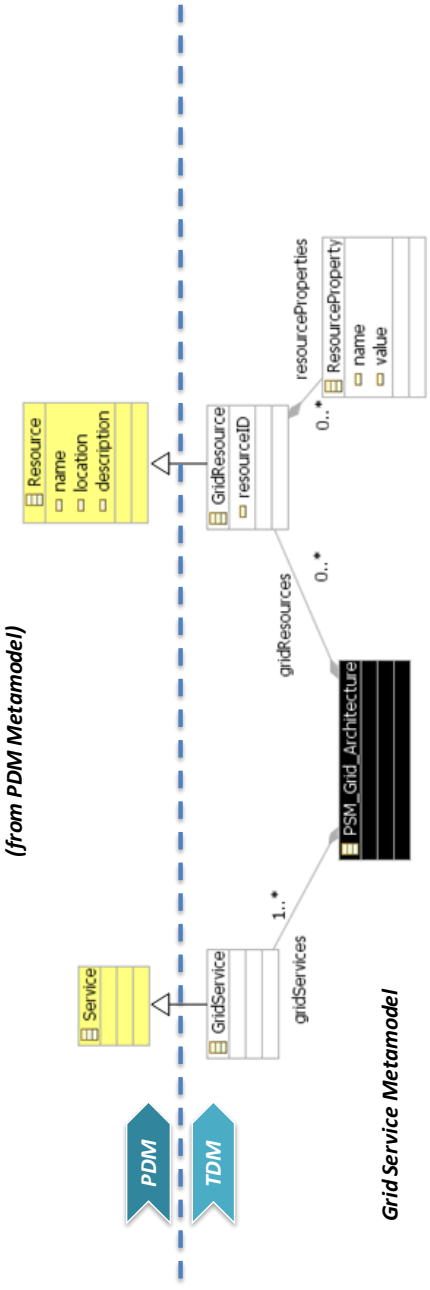

Figure 3-7. Concepts for modelling Grid Services at TDM abstraction level.

3.2.2.3.1 Grid Services

A Grid service is a Web service that works in a Grid environment. Taking into consideration the fundamentals of Grid Computing about enabling virtual organizations based on services, it is appropriate to state that a Grid Service is a Web service that "virtualizes" the use of any element present in the Grid environment and that allows to access to any facility managed in the Grid [76]. That means that a Grid Service *knows* how to access to any resource that is virtualized so it is possible to be used by any client connected to the distributed Grid system. In that sense, the architectural properties of a Grid service are exactly the same that were defined previously for Web Services and so they will the relationships and features.

The only remarkable difference between a standard Web Service and a Grid Service can be found in the moment of communicating with the service to access a concrete resource of the Grid. In order to properly locate a Grid resource, the communication with a Grid service will be include an **endpoint reference**, which is a pair of *address-resource identifier* where the *address* element corresponds to the URI (URL plus URN) of the Web Service and the resource identifier allows to uniquely identify a resource within the Grid and that is controlled by the previous Web Service. This resource identifier, in turn, is described according to the WS-Resource standard and includes a set of port types for querying and modifying the state of that resource. The resource addressed using the WS-Resource may refer to a physical resource (a file or a database) or a more 'ethereal' one, in the sense of maintaining the value of a temporal variable between invocations to the Grid Service. It is compulsory to know the existence of this naming scheme in Grid service environments although it is not explicitly included in the architectural model.

The use of Endpoints within Grid environments is encapsulated inside the messages that Web services in Grid environments exchange so they have no influence on the architectural configuration or the architectural model at this level.

3.2.2.3.2 Grid Resources

A Grid resource is anything that can be virtualized within the scope of a Grid environment and that has some specific properties. Within the *ArchiMeDeS* framework, a Grid resource represents the same concept as a WS-Resource as it is defined by OASIS [91]. The architectural element has been assigned a different name since the Web Service extension already defines the concept of *WSResource*.

3.2.2.4 TDM Abstract Syntax: REST Services

Other important platform that has arisen as an alternative to Web Services can be found in the form of the REST (*Representational State Transfer*) technologies. Though it can be considered as a restriction to the vision of the WSA initiative, it uses many of its principles but relying only on standard operations of the HTTP protocol [70]. It is also related to Grid Computing since both Grid Services and RESTful services aim to give a solution to the management of the state within service-oriented environments.

Analyzing its initial definition given by Roy Fielding in its Thesis dissertation [69], REST can be understood either at the same level as SOAP (as a protocol over the HTTP communication protocol), as a way of formatting the messages and the access means to an existing web service or as an architectural style (as it happens with SOA itself) [127]. The importance gained by REST is that it is entirely focused on the Web and thus its capabilities are constrained to that environment. Web Services, in turn, can be placed in other more local environments not necessarily placed over HTTP.

According to Fielding, the architectural elements that build up a REST architecture are: *Data elements* (resources and representations), *Connectors* (managers of network communications) and *Components* (primarily *service agents*). To fully comprehend the REST initiative it is necessary to fix some premises [185]:

- **Resource identification via URIs**. A RESTful Web Service exposes a set of resources which identify the targets of the interaction with its clients. Resources are identified by *URIs* (which is in accordance to the definition given in the core PSM metamodel of *ArchiMeDeS*)
- **Uniform interfaces.** Resources are manipulated using a fixed set of operations due to the fact of relying on the HTTP protocol: PUT, GET, POST and DELETE. These operations are the only means of interaction with the exposed resources.
- **Self-descriptive messages.** Resources are decoupled from their representation so their content can be accessed in a variety of formats (e.g. HTML, XML, JPG, plain text, PDF, other MIME types, etc.).
- **Stateful interaction via hyperlinks.** Every interaction with a resource is stateless (the HTTP protocol itself is stateless) and, therefore, request messages should be self-contained. That means that, in order to operate with a resource (either to modify it or continue a previous interaction) the invocation must contain all the information needed.

- **Implementation agnosticism.** REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements.

Taking into account the features of the REST initiative, the extension proposed for modelling REST services at the PSM-level of abstraction defines the following elements: **RESTAgent**, **RESTService**, **RESTResource** and **RESTContract**, being all four specializations of the *ServiceAgent, Web Service*, *WSResource* and *ServiceContract* from the core metamodel and the Web service extensions previously specified. These elements are gathered in a metamodel that can be seen in Figure 3-8.

3.2.2.4.1 RESTAgents.

REST Agents are Service Agents whose main purpose is to know what 'piece of code' (i.e. RESTService) should be executed when a request to a concrete URL is made. This URL will be a reference to a specific *RESTResource* under the control of a RESTService (that 'piece of code' mentioned previously).

Architectural properties are the same as the ones defined for Service Agents in the core metamodel. However, an additional element supporting this concept has been included in the current extension for the sake of clarity in architectural models of REST-based configurations. This way it is possible to clearly differentiate those service agents that are capable of executing the corresponding REST service when an operation for a REST resource is received by the REST agent.

3.2.2.4.2 REST Services

REST Services are the computational entities that actually respond to client request that desire to access to a resource. These kinds of services are portions of code (e.g. Java classes) that are executed by a REST Agent and have the right to act over the resource depending on the type of operation requested.

From the architectural point of view, the main difference of REST services with Web Services is that, although both have a **name,** a **location** (an URL) and **can be described**, REST services do not need the existence of an explicit interface exposing the offered operations (they are always the same). As it will be explained later on, the service contract is also standardized requiring only the specification of the message exchange pattern used with each of the predefined operations.

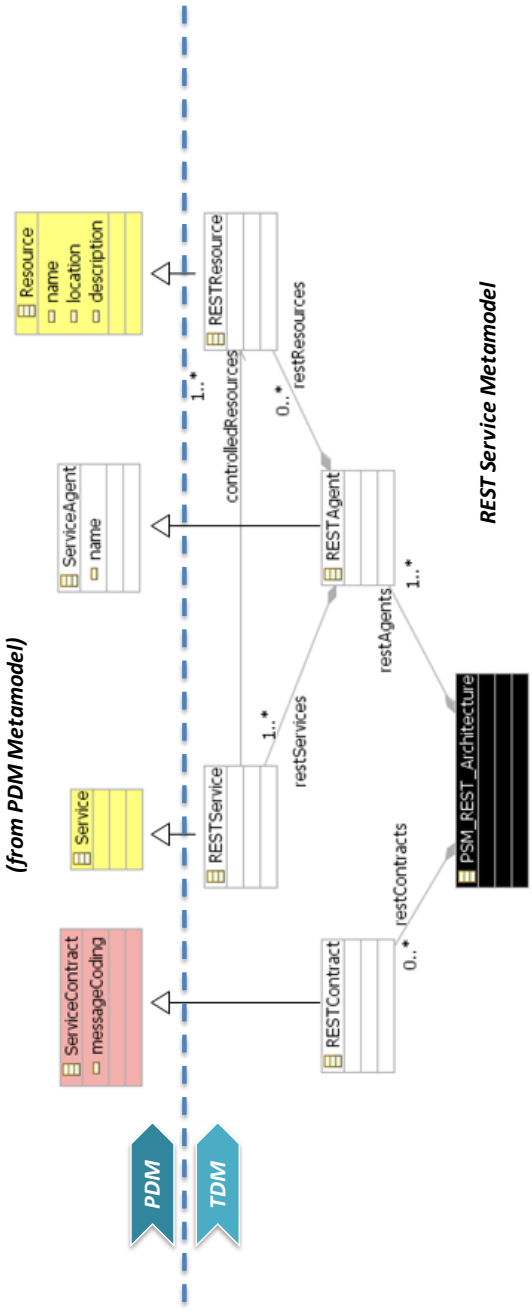

Figure 3-8. Concepts for modelling REST Services at TDM abstraction level

3.2.2.4.3 REST Resources

As it happened with the Grid Computing paradigm, in REST-based environments, resources are understood as any kind of information that can be named (a document or image, a temporary value asked to a service, a collection of other resources, etc.). However, from the REST point of view, a resource is a conceptual mapping to a set of entities (a generic description of the resource, i.e. a *resource representation*, hence the name REST) and not the entity that corresponds to the mapping at any particular point of time (that is, the values or *state* of a resource in a specific moment).

Within the architectural model, a REST resource will be an element clearly identified (as any other resource in service-oriented environments) and will comprise a coherent description of the resource itself. Since no standard is recommended in the REST specification for the description of the resource properties, the extension proposed do not include any explicit reference to that properties as it already happened in the case of extension for Grid-based environments.

3.2.2.4.4 REST contracts

REST services represent a special case when considering the establishment of access conditions from consumer services to providers. REST services are built upon the use of the HTTP protocol as underlying layer for communications and thus it is that protocol the one imposing the conditions under which the contractors may communicate. In this kind of environments, the generally adopted contract is named **uniform contract** [183]**.** It appears as consequence of the standardization of the operations needed to access a service (or, more properly, a resource) and that consists on the description of the URI defining the resource and the standard operation accompanying the resource. In that sense, the behaviour is implicit, since there is no contract that governs the behaviour of each URI-identified resource, hence transferring the constraints of the use and governance issues of the resource to the management of the resource itself.

The influence of this kind of contract is translated into the architectural model by defining a concrete unique element named *RESTContract* establishing the standard conditions of REST contracts. That is, that the operations will be the ones aforementioned (PUT, GET, POST and DELETE) corresponding to the creation, retrieval, updating and erasure of (the state of) a resource.

The options for message exchange patterns are shared with the ones defined for Web Services (i.e. the ones defined by WSDL 2.0) since WSDL 2.0 support the definition of Web Services in the REST way [131].

3.2.2.5 Concrete Syntax

Since the PSM-level of the DSL for Software architectures in *ArchiMeDeS* is divided according to different technological platforms, so it will the UML profile that serves as concrete syntax for that DSL using UML. To illustrate this,

Figure 3-9 shows a diagram with the relationships established among the profiles for this level of abstraction.

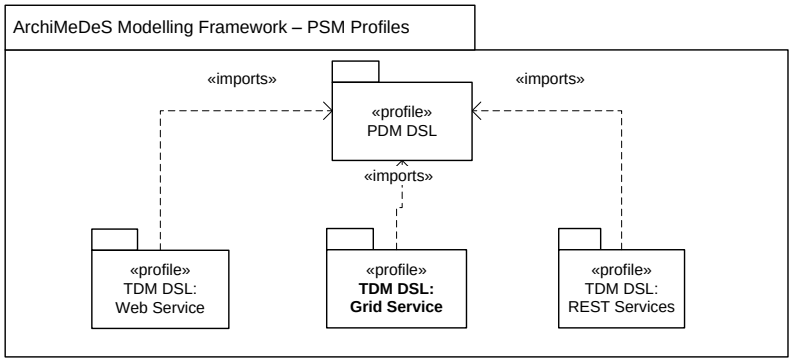

Figure 3-9. Profiles for the PSM DSL in the ArchiMeDeS framework.

The UML diagram chosen to represent the architectural model is, likewise at PIM level, the *Class diagram*. The stereotypes defined for any PSM profile have also associated the closest concept that may be found in the UML metamodel. Following illustrations expose these stereotypes defined for each extension of the PDM metamodel for service-oriented technologies to be used within the TDM abstraction level of the model architecture.

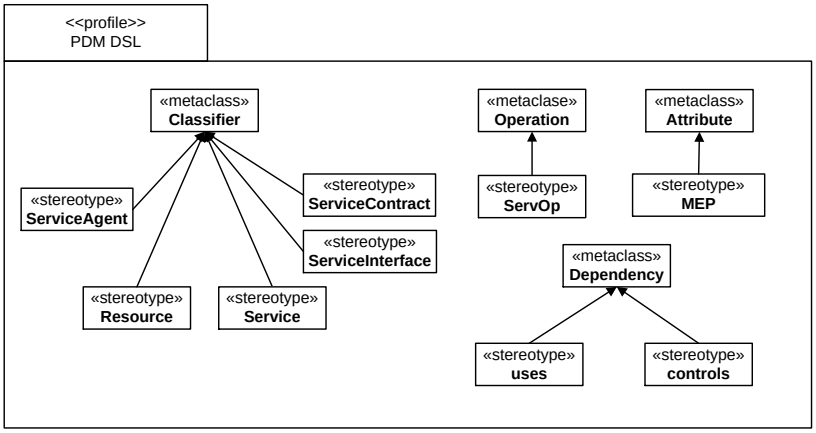

Figure 3-10. UML profile corresponding to the PDM DSL for Service-Oriented platforms.

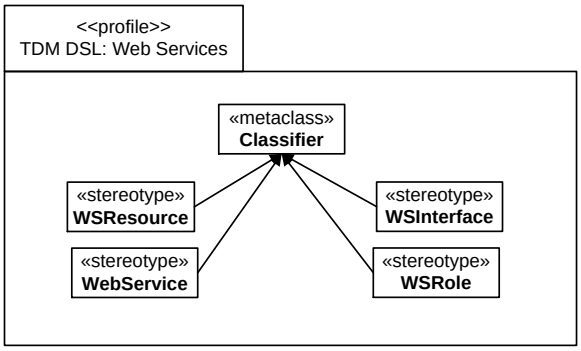

Figure 3-11. UML profile for modelling Web Services.

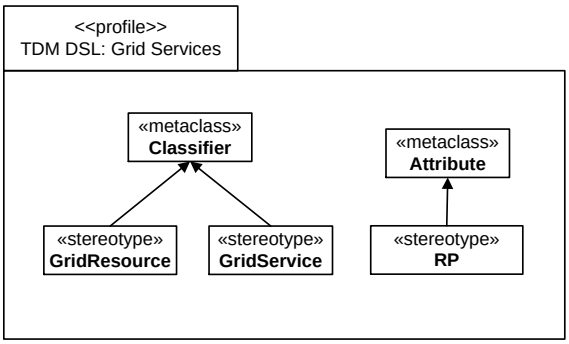

Figure 3-12. UML profile for modelling Grid Services.

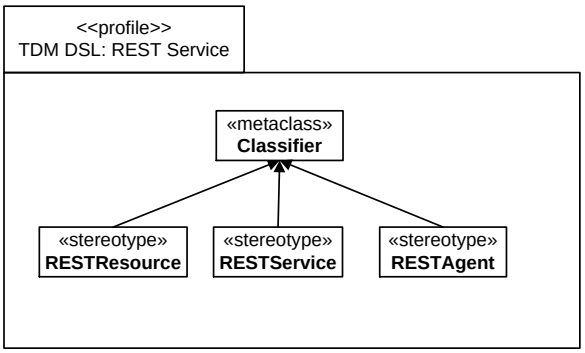

Figure 3-13. UML profile for modelling REST Services.

3.2.2.6 PSM DSLs Summary

This sections aims to present a summary of the concepts and stereotypes defined for each DSL proposed at the PSM level of the MDA architecture. Since this level has been subdivided in two levels, the summary will present a table for each of the DSLs proposed. Each table depicts the concept, the associated semantics and the graphical stereotype used to represent it with UML. In addition, the derived meta-class used and any restriction posed over the concept are depicted. This way, Table 3.3 shows a summary of the terms that build up the DSL created for modelling Software Architectures at the PDM abstraction level and Tables 3.4, 3.5 and 3.6 the corresponding elements of each technology chosen for the TDM abstraction level.

Table 3.3. Concepts and stereotypes of the PDM DSL.

CONCEPT	*SEMANTICS*	*NOTATION*		*RESTRICTIONS*
		BASE UML META-CLASS	*STEREOTYPE*	
Service Agent	Supporting substrate of services and resources	Classifier	<<ServiceAgent>>	---
Resource	Passive element of the architecture	Classifier	<<Resource>>	Must have a location. Description is optional.
Service	Performs part of the system functionality	Classifier	<<Service>>	Must define at least one interface
Service Operation	Atomic functionality provided by a service	Operation	<<ServOp>>	---
Service Interface	Subsets the service functionalities	Classifier	<<ServiceInterface>>	May contain a reference to a service contract
Service Contract	Agreed communicating conditions among service entities	Classifier	<<ServiceContract>>	Must define the message exchange pattern
Message Exchange Pattern	Indicates the way a service operation must be consumed & interaction protocol	Attribute	<<MEP>>	---

Table 3.4. Concepts and stereotypes of the TDM DSL for Web Services.

CONCEPT	SEMANTICS	NOTATION		RESTRICTIONS
		BASE UML META-CLASS	STEREOTYPE	
Web Service	Service defined according to the WSA	Classifier	<<WebService>>	Must define both a URL and a URN
WS Resource	Passive element controlled by a service agent	Classifier	<<WSResource>>	Must be clearly related to a service agent
WS Interface	Interface defined according to WSDL	Classifier	<<WSInterface>>	Must define a service contract
WS Role	Subset of the service functionalities	Classifier	<<WSRole>>	---

Table 3.5. Concepts and stereotypes of the TDM DSL for Grid Services.

CONCEPT	SEMANTICS	NOTATION		RESTRICTIONS
		BASE UML META-CLASS	STEREOTYPE	
Grid Service	Stateful service virtualizing a Grid Resource	Classifier	<<GridService>>	Must be related to at least one resource
Grid Resource	Virtualized element in a Grid environment	Classifier	<<GridResource>>	Must have a set of Resource Properties
Resource Property	Defines part of the attributes of a Grid Resource	Attribute	<<RP>>	---

Table 3.6.Concepts and stereotypes of the TDM DSL for REST Services.

CONCEPT	SEMANTICS	NOTATION		RESTRICTIONS
		BASE UML META-CLASS	STEREOTYPE	
REST Agent	Agent executing a REST service depending on a RESTResource	Classifier	<<RESTAgent>>	---
REST Service	Service that responds to the standard HTTP operations	Classifier	<<RESTService>>	---
REST Resource	Resource that can export its actual state as required by a REST service	Classifier	<<RESTResource>>	---

3.2.3 Modelling DSL Transformations

ArchiMeDeS has been conceived as a model-driven framework and thus it follows an approach based on MDE principles. For that reason, the specification of model transformations allowing the automation of model evolution is highly recommended. This subsection is devoted to explain the model transformations that are included in *ArchiMeDeS* as part of its model-driven strategy for architecture specification. These transformations are defined in order to a) be able to obtain a concrete PSM representation of the architecture from its corresponding PIM model (PIM-to-PSM transformations); and, b) improve the semantic information gathered in the architectural models through the use of architectural styles (PIM-to-PIM transformations).

For that purpose, next subsections define, first, an introduction on model transformations and the kinds of transformations considered regarding the outline of the DSLs defined; next, the set of transformations needed to advance from PIM architectural models to those at the PSM level of a specific service-oriented platform and, finally, the process to include architectural styles information into the PIM architectural model.

3.2.3.1 A Taxonomy of Model Transformations

Just like every program conforms to the grammar of its respective programming language, each model conforms to a metamodel. The metamodel describes the various kinds of elements that can be included in a model and the way they are arranged, related and constrained. A model that is valid according to the corresponding metamodel is said to *conform to* the metamodel, just as a program can be syntactically correct according to the respective programming language.

Nevertheless, working with multiple, interrelated models requires a significant effort to accomplish some tasks related with model management, such as refinement, consistency checking, refactoring, etc. Many of these activities can be performed as automated processes, which take one or more source models as input and produce one or more target models as output, following a set of transformation rules. In the context of MDE this process is known as **model transformation** [206].

Model transformation is defined at the metamodel level, i.e. it maps elements from an input (*source*) to the output (*target*) metamodel. Consequently, it can be used to generate an output model from any set of models conforming to the input metamodel. In other words, the model transformation program works for any model that conforms to the input metamodel. An overview of a generic model transformation process can be seen in Figure 3-14.

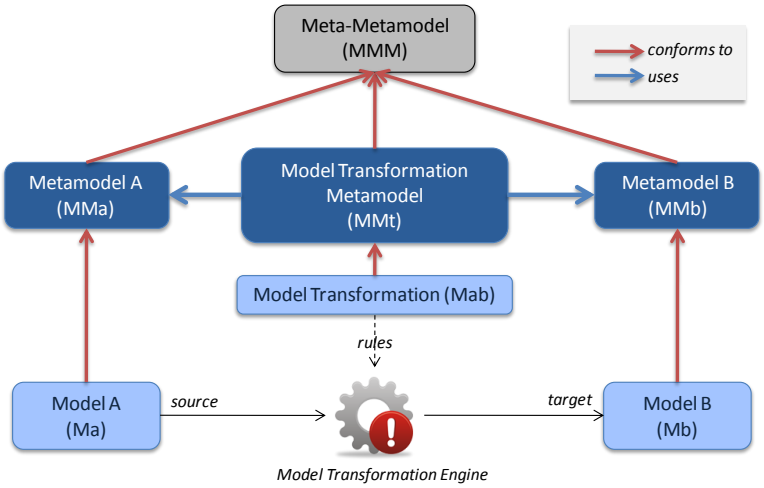

Figure 3-14. Overview of a generic model transformation process.

The root of a transformation process is the *meta-metamodel* (MMM). It provides the architect with a set of basic abstractions that allow defining new metamodels. Next, the source and target metamodels are defined by instantiating the abstractions provided by the meta-metamodel. They are said to conform to the meta-metamodel. Finally, the model transformation engine executes the *Mab* model transformation to map a model *Ma* into another model *Mb*. To do so, *Mab* specifies a set of rules that encodes the relationships between the elements from the *MMa* and *MMb* metamodels. So, it can be used to generate an output model from any set of models conforming to the input metamodel. In other words, the model transformation program works for any model defined according to the input metamodel.

Note that if the set of rules and constraints that drives the construction of a model transformation is defined in a metamodel (MMt), any model transformation will be expressed as a model conforming to that metamodel. Expressing model transformations as models (so-called *transformation models*) makes possible manipulating them by means of other transformations. This provides with several advantages. For instance, any model transformation can be the input or output of another model transformation. In addition, it is possible to compose transformation models as the composition of any other type of models [23], to deploy metamodel evolution and model co-evolution techniques [38], define chains of model transformations [225], reuse exiting model transformations [201], etc.

In order to define what model transformations are relevant in the context of the *ArchiMeDeS* framework, firstly, it is compulsory to have a close look to existing model transformation kinds and analyze which is the one that best suits to the features defined for the presented proposal. To achieve that, this exposition will make use of the model transformation taxonomy defined by Mens et al. [146].

According to that taxonomy, model transformations can be classified according to different criteria. Here, the focus is put on the language influence of the source and target models and their abstraction level in which they are defined:

- **Endogenous transformations (*rephrasing*) vs. exogenous transformations (*translation*).** This criterion refers to the language used for the source model and the target model. In the case the language is the same, the transformation is considered as a rephrasing. In the case of being different, the transformation is said to be a translation.

  Regarding the model transformations considered in *ArchiMeDeS*, those transformations that aim to include architectural style information in PIM models will be understood as endogenous transformations, since the source model (*PIM architectural model*) and the target model (*Enriched PIM architectural model*) are described using the same DSL and, therefore, conform to the same metamodel. On the contrary, PIM-to-PSM transformations (either to PDM or TDM models) will be considered as exogenous transformations since, although using a service-oriented semantic foundation, source and target DSLs are different.
- **Horizontal vs. vertical transformations**. This classification criterion makes reference to the abstraction level in which the transformations take place. This way, it is possible to define *horizontal transformations* when source and target models are comprised within the same abstraction level and *vertical transformations* when they belong to different abstraction levels. In the case of *ArchiMeDeS*, since the proposal follows the MDA approach, PIM-to-PIM transformations will be managed as horizontal transformations and PIM-to-PSM, in turn, will be considered as vertical ones.

In summary, model transformations within *ArchiMeDeS* are clearly defined in two groups: **PIM-to-PIM transformations**, *horizontal* and based on a *rephrasing* of the elements modelled; and **PIM-to-PSM** (either PIM-to-PDM or PIM-to-TDM), *vertical* and with the aim of achieving a *language migration*.

In order to gain a full comprehension of the transformation processes that are defined as part of the *ArchiMeDeS* framework, it is necessary to study the kinds of transformations that may take place in model transformations and their application over service-oriented architectural models. For that aim, Table 3.7 shows a summary of the kinds of transformations considered and an example of their application regarding the transformations modelled and their architectural relevance in that context.

Table 3.7.Architectural relevance of the kinds of transformations considered.

CARDINALITY	MEANING	PIM-TO-PIM TRANSFORMATIONS		PIM-TO-PSM TRANSFORMATIONS	
		ARCH. RELEVANCE	EXAMPLE	ARCH. RELEVANCE	EXAMPLE
1 → 1	Simple mapping	NO	---	YES	*Service* (processing) to *GridService*
1 → N	Decomposition	YES	*Service Type* to *Service* + *Service Type*	YES	*Service* (information) to *Service* + *Resource*
N → 1	Merge	YES	---[4]	YES	*Service* + *ServContract* to *WSInterface*
N → M	Composition	YES	*Service* + *Component* to *Service* + *ServiceRole*	NO	---

Model transformations can be further classified depending on the cardinality of the transformations accomplished, that is, depending on the number of source elements required by the transformation and the target elements obtained as a result. This way, the possible combinations are the ones reflected in Table 3.7.

- **One to One transformations (1 → 1)**. This scenario refers to those situations in which there is a one-to-one correspondence between a meta-class from the source metamodel and another from the target one. When

[4] *This alternative does make sense from an architectural point of view but it will not happen within the proposed framework.*

modelling PIM-to-PSM transformations this is the most common situation and many examples can be found. On the contrary, when performing PIM-to-PIM transformations, this scenario will not occur since, as it will be explained later, the objective is to populate the PIM architectural model with information coming from the architectural style model by means of "annotating" the source model with *Service Roles*, not changing or modifying the original element.

- **One to Many transformations (1 → N)**. In this case, the scenario refers to situations in which the source element transforms into more than one element in the target model. An illustrative example can be found in PIM-to-PSM transformations where, from a conceptual *Service* identified with the *information* variant in the PIM model, a tuple *Service-Resource* is obtained as a result in the PDM model. When superimposing architectural style information to PIM models, the example arises in the moment of having to decompose a *service* instance (or a *ServiceType*) into both a *service instance* and the corresponding *ServiceType*. This occurs because, on the one hand, the *Service Role* will be defined as part of the *Service Type* (since it comprises a subset of the functionality provided by it) and, on the other hand, *Service Roles* are exclusively played by service instances, thus needing the existence of both elements.
- **Many to One (N → 1)**. This scenario takes place when an element in the target model is created from the information contained in more than one source element. That is the case, for instance, of a *WSInterface* at the Web Service TDM model that needs information from both a *Service* and a *Service Contract* of the source PIM model. Although no similar situation can be extracted from the defined PIM-to-PIM model transformations regarding architectural style superimposition, *merge* transformations may occur in the moment of modelling situations in which a new element needs to be created from the information contained in different source elements. This case would be handled as if it were a *Many-to-many* transformation and, accordingly, the transformation strategy will be that explained next.
- **Many to Many (N → M)**. Finally, this scenario occurs when from several elements of the source model need to be transformed into more than one element in the target model. This is the approach that will be mainly followed in PIM-to-PIM transformations. From the information collected in both PIM architectural model and the model with the architectural style information (that is, from more than one source

element) the transformation will result in (actually: *merge*), at least, two elements in the target Enriched PIM architectural model (a *Service* and a *Service role*). The approach for that kind of transformations is to manage them as a composition of simpler situations.

3.2.3.2 PIM-to-PSM Transformations

So far, this dissertation has discussed how to manage abstraction levels separately. Previous sections dealt with modelling the architecture of a software system using a concrete service-oriented point of view, either at the PIM abstraction level (including, or not, information from architectural styles) or a more platform dependent level, whichever the technology of choice used. However, in order to consider *ArchiMeDeS* as a model-driven framework for architecture specification, it is compulsory to provide means for (semi-) automatically transferring the information gathered at conceptual levels into specific technological approaches via model transformations.

Accordingly, the next step towards the completion of *ArchiMeDeS* is the definition of vertical model transformations that will help to translate the models defined with such DSLs. To that end, it is necessary to specify and implement the mapping rules that compose each model transformation. The rules defined for that aim are based on basic transformations (see Section 3.2.3.1), for more complex mappings supplementary information would be required. This additional information would come from the modelling of other development concerns (behaviour, interface, storage, etc.) and from architectural decisions not considered in the scope of this Doctoral Thesis (design rationale, patterns, templates, etc.).

In order to define PIM-to-PSM transformations it is necessary to take into account the separation in two levels that has been defined for modelling architectures at the PSM level. Accordingly, different model transformations have been defined from PIM to PDM and from PIM to each of the TDM DSLs. Reasons for this decision can be summarized in the following:

- PDM and TDM models share the same abstraction level (PSM) with the only difference of minor technological features included as part of the TDM models. In some scenarios and counting with an adequate tool support, it would be enough to have a PDM version of the architecture in order to obtain the source code of a software solution.
- TDM DSLs are considered as “extensions” to the information modelled at PDM level. In fact, TDM models describe the particularities of the architectural models according to the implementation of PDM models depending on the target technology.

- Obviously, all the architectural information modelled in PDM models is reflected in TDM models. In many cases, the information of PDM models is seamlessly modelled at TDM level.
- PSM architectural models can be understood to be comprised by either PDM models exclusively; or as the union of PDM models plus the particularization of some elements according to the chosen TDM.

Regarding how model transformations should be defined, the MDA guide [151] stated that "*the mapping description may be in natural language, an algorithm in an action language, or a model in a mapping language*" (p. 24). This way, model transformations will be described in two parts:

- First, the possible **mappings** between models are defined **using natural language**. They will be structured by collecting them in a set of rules, expressed in natural language. This first part is explained in the following, indicating which are the corresponding elements of each PSM-level DSL that can be obtained from elements specified at the PIM-level DSL.
- Second, the resulting **transformation rules** are implemented **using ATL**. This second part will be explained as part of the tool created to support the *ArchiMeDeS* framework. Since it is considered an implementation issue, this explanation has been delayed to section 4.2.

The target platforms available at PSM depend on the target abstraction level. This way, it will be possible to evolve the architecture to a nonspecific service-oriented platform (via transformation rules for the PDM level) or to the ones corresponding to the technologies defined at TDM level: Web Services, Grid Services and REST. The definition of transformation rules from PDM models to any of the TDM modes have been left for ongoing work.

3.2.3.2.1 Mapping Rules from PIM to PDM

The PIM-level DSL defined previously can be used to obtain different target PDM architectural models. The definition of the translation of the concepts found in the former DSL will be translated, firstly, into the concepts present in the PDM DSL:

- ***Transformation of Service Providers***. Organizational entities modelled at the PIM level (*Service Providers*) will be mapped to *Service Agents* in charge of executing and managing the access to specific services. The *name* property will be transferred from one element to the other.

The identification of *inner* and *outer* providers has a business meaning and does not have a direct influence on the technological elements identified at the lower level of the architecture. In turn, that differentiation will have an influence on other parts of the development methodology: for example, it will determine, to some extent, if services directly depending on them will need a concrete coding task or not.

- ***Transformation of Services and Service Types.*** Services and service types will be transformed in technological resources as part of the PDM architectural model. However, since resources can be specialized according to their activeness (in *resources* or *services*), the mapping of these elements is not trivial.

  The concrete mapping will depend on the value assigned to the *variant* property of the source *service* (or *Service Type*). As it was pointed out in the moment of defining the PIM DSL, services may be classified in *Interaction*, *Information*, *Processing* or *Orchestration* services. Their particular mapping is the following:

  - **Information Services**. Information services were defined as services allowing the access to a specific resource. This way, source services with this variant will be transformed into a pair Service-Resource in which a 'control' association must be created in the target model from the service to the resource.
  - **Interaction Services**. This kind of services, in contrast, needs a resource to act as entry point for its offered capabilities. In that sense, the transformation that will take place will generate a pair Resource-Service in which a 'use' association is included in the model indicating the relationship established between them (from resource to service).
  - **Processing Services**. In this case, the mapping is a one-to-one transformation in the sense that a sole service element can be automatically created as part of the target model.
  - **Orchestration Services**. As in the previous case, the identification of this kind of services will generate a unique service in the target architectural model. As it happened with the identification of inner and outer providers, the specification of an orchestration service aids to fulfil other tasks of the methodological framework (specifically, the behavioural modelling tasks).

In any of the previous cases the *name* property of the newly generated elements will take its value from the corresponding property of the source element. In the cases where the transformation may generate more than one element, the implementation of the transformation includes a modification of the resulting elements by adding a suffix to the source value (such as *"_resource"* or *"_service"* for the created resources and services respectively).

The SERVID property is an artificial element created to uniquely identify a service within a PIM architectural model. In that sense, there is no direct correlation between that property and the *location* or *description* properties that elements at the PDM contain. In the case of *ArchiMeDeS* framework, the option chosen has been to map the value of the SERVID property to the *description* target property so this information is not lost.

- ***Transformation of Service Operations.*** The capabilities that a service provides will be directly mapped to PDM service operations since there is no semantic difference when evolving from the conceptual level to a more technological one. Associated properties and types for each operation element will also remain the same. As it will be later indicated, a more significant divergence will occur when transforming the PIM-level model into a concrete extension of the TDM level of the model architecture.
- ***Transformation of Service Roles.*** Service roles gather the piece of functionality that a conceptual service offers in a concrete moment. In that sense, the equivalent at a platform specific level is the *service interface* element. Consequently, the information put on the *service role* element will be transferred to a *service interface* including its *name* and the set of *operations* associated to it.
- ***Transformation of Service Contracts.*** As it was explained in section 3.2.3.1, service contracts at PDM also represent the conditions under which some consumer may use the operations offered by a service. In that sense, the *service contract* element of the PIM will be seamlessly transformed into a *service contract* at PDM.

  The *Interaction pattern* property that service contracts own will be used to populate the *Message Exchange Pattern* property in the target platform. Consequently, the *patternType* attribute will receive its value from the origin *exchangePattern* attribute.

Some particularities on this transformation depend greatly on the target platform and technology defined for the PSM abstraction level and thus they will be specified in its corresponding subsection of the TDM sublevel. However, apart from the mappings defined for the structural elements and properties that may appear in every PIM architectural model, the associations that they may maintain among them will also be transformed once the target elements are obtained. Table 3.8 shows a summary of all the previous transformations from PIM to PDM.

Table 3.8. Mapping guidelines from PIM to PDM.

SOURCE MODEL: PIM		*TARGET MODEL: PDM*		*OBSERVATIONS*
Element	***Properties***	***Target Element***	***Properties & value assigned***	
Service Provider / **Inner Provider** / **Outer Provider**	name	**Service Agent**	name = name	Business contracts are eluded at technological level
Service Type	Variant = Information	**Service**	name = name + '_service'	New '*controls*' association from *service* to *resource*
		Resource	name = name + '_resource'	
	Variant = Interaction	**Service**	name = name + '_service'	New '*use*' association from *resource* to *service*
		Resource	name = name + '_resource'	
	Variant = Processing	**Service**	name = name	---
	Variant = Orchestration	**Service**	name = name	---
Service	name	**Service**	name = name	Mappings defined for *Service Types* due to variants are also applied to *Services*
	SERVID	**Service**	description = SERVID	
Service Contract	name	**Service Contract**	name = name	Links to service *Consumer* and *Provider* are mapped without variation
Interaction Pattern	interactionKind	**Message Exchange Pattern**	patternType = interactionKind	Other existing patterns will be manually assigned

SOURCE MODEL: PIM		*TARGET MODEL: PDM*		*OBSERVATIONS*
Element	*Properties*	*Target Element*	*Properties & value assigned*	
Service Operation	name	**Service Operation**	name = name	*Parameters* and *Return Values* are also mapped
Synch. Operation	---	**Synch. Operation**	---	
Asynch. Operation	---	**Asynch. Operation**	---	
Service Role	name	**Service Role**	name = name + '_role'	Operations from PIM model are assigned to PDM roles or interfaces
		Service Interface	name = name + '_interface'	

3.2.3.2.2 Mapping Rules from PIM to TDM: Web Services

Since the Web Service Extension defined extends the concepts of the PSM-Core DSL, the transformations that will be explained in the following refer to those elements that need a particular modelling or that modify the already explained standard transformation to the PDM DSL. It is important to note that not all the features supported by the DSL created can be obtained from the information that the PIM architectural model provides. Aspects such as the *URL* or the *ownership* of the service or the concrete *MEP* that will be defined inside a service contract are among them.

- ***Services become WSResources and WebServices.*** Some of the few transformations that can be automated when selecting the Web Service Extension as target DSL, refer to the identification of which elements will become *WSResources* and/or *WebServices* depending on the value assigned to the variant attribute of the source service (as it was mentioned in previous section).
- ***Pattern Types assigned by default.*** To ease the transformation of service contracts, this part of the *ArchiMeDeS* framework assigns, by default, the next correspondences between the values of the kind of interaction defined for each part of the *InteractionPattern* and the *WSDL20_MEP* values: *One-Way* to *InOnly* and *Query-Response* to *InOut.*

  The other two patterns (*Dialogue* and *Choreography*) entail an interaction logic that will be encapsulated in the logic of either consumer and provider services (or both) so there won't be any reflection of these patterns in the TDM architectural model.

- ***Service Roles and Service Interfaces.*** Within the WSA specification, the concepts of *Role* and *Interface* refer to the partial functionalities a service may offer. Accordingly, as it happened with the transformation to the PDM, the *ServiceRole* defined at PIM level will be mapped to two elements: *WSRole* and *WSInterface*. Obviously, the name assigned to these elements will have attached a suffix ("*_role*" or "*_interface*") in order to clearly differentiate them within the architectural model. The corresponding associations with the rest of the elements of the model will be assigned in accordance.

Table 3.9 shows up a summary of the mapping transformations from PIM to the Web Service version of the TDM abstraction level.

Table 3.9. Mapping guidelines from PIM to TDM: Web Services.

SOURCE MODEL: PIM		*TARGET MODEL: TDM WEB SERVICES*		*OBSERVATIONS*
Element	***Properties***	***Target Element***	***Properties & value assigned***	
Service Type	Variant = Information	**WebService**	name = name + '_service'	New '*controls'* association from *service* to *resource*
		WSResource	name = name + '_resource'	
	Variant = Interaction	**WebService**	name = name + '_service'	New '*use'* association from *resource* to *service*
		WSResource	name = name + '_resource'	
	Variant = Processing	**WebService**	name = name	---
	Variant = Orchestration	**WebService**	name = name	---
Service	name	**WebService**	name = name	Mappings defined for *Service Types* due to variants are also applied to *Services*
	SERVID	**WebService**	description = SERVID	
Interaction Pattern	interactionKind = One-Way	**Message Exchange Pattern**	WSDL20_MEP = InOnly	Other existing patterns will be manually assigned
	interactionKind = Req.-Resp.		WSDL20_MEP = InOut	
Service Role	name	**WSRole**	name = name + '_role'	Operations from PIM model are assigned to TDM roles or interfaces
		WSInterface	name = name + '_interface'	

3.2.3.2.3 Mapping Rules from PIM to TDM: Grid Services

As it happened with the mappings from PIM to the TDM Web Service extension, the automation of the transformations to Grid Service TDM models are mainly covered by the transformations defined for the PDM. The main significant differences come in the definition of mapping rules from *Services* and *Service Types* to the matching *GridServices* and *GridResources*. These transformations will take place according to the *variant* attribute of the source service/service type from the PIM model. Modelling of Grid properties cannot be extracted from the information collected in PIM architectural models.

Table 3.10 shows a summary of the mapping transformations from PIM to the Grid Service version of the TDM abstraction level.

Table 3.10. Mapping guidelines from PIM to TDM: Grid Services.

SOURCE MODEL: PIM		*TARGET MODEL: TDM GRID SERVICES*		*OBSERVATIONS*
Element	***Properties***	***Target Element***	***Properties & value assigned***	
Service Type	Variant = Information	**GridService**	name = name + '_service'	New '*controls*' association from *service* to *resource*
		GridResource	name = name + '_resource'	
	Variant = Interaction	**GridService**	name = name + '_service'	New '*use*' association from *resource* to *service*
		GridResource	name = name + '_resource'	
	Variant = Processing	**GridService**	name = name	---
	Variant = Orchestration	**GridService**	name = name	---
Service	name	**GridService**	name = name	Mappings defined for *Service Types* due to variants are also applied to *Services*
	SERVID	**GridService**	description = SERVID	

3.2.3.2.4 Mapping Rules from PIM to TDM: REST Services

In order to provide with a coherent set of mapping rules to evolve from a PIM architectural model to one supporting the features of REST Services it is important to take into account the restrictions imposed by this technological platform. This way, the following model transformations can be defined:

- ***Transformation of Services and Service Types.*** Services and service types will be transformed in *RESTServices* and *RESTResources* (if any) as in the case of previous Extensions: depending on the value of the *variant* attribute.
- ***Transformation of Service Agents.*** As it was explained in section 3.2.3.4, *RESTAgents* in charge of executing and managing *RESTServices* and *RESTResources* are clearly identified in the architectural model at the PSM level. This way, a specific mapping rule allowing the transformation from *ServiceProvider* to *RESTAgent* is included within this part of the *ArchiMeDeS* framework.

Table 3.11. Mapping guidelines from PIM to TDM: REST Services.

SOURCE MODEL: PIM		*TARGET MODEL: TDM REST SERVICES*		*OBSERVATIONS*
Element	***Properties***	***Target Element***	***Properties & value assigned***	
Service Provider **Inner Provider** **Outer Provider**	name	**RESTAgent**	name = name	Business contracts are eluded at technological level
Service Type	Variant = Information	**RESTService**	name = name + '_service'	New '*controls*' association from *service* to *resource*
		RESTResource	name = name + '_resource'	
	Variant = Interaction	**RESTService**	name = name + '_service'	New '*use*' association from *resource* to *service*
		RESTResource	name = name + '_resource'	
	Variant = Processing	**RESTService**	name = name	---
	Variant = Orchestration	**RESTService**	name = name	---
Service	name	**RESTService**	name = name	Mappings defined for *Service Types* due to variants are also applied to *Services*
	SERVID	**RESTService**	description = SERVID	

3.2.3.3 PIM-to-PIM Transformations

Design decisions are an important issue to bear in mind when specifying software architectures, not only due to the need for reusing existing components but also because of restrictions imposed by the business process itself (for example due to organizational restrictions of the company). These particularities have a direct effect on the system architecture since they affect the way architectural elements may relate and communicate to each other. A way of dealing with these issues is by explicitly considering them *inside the architectural model at a conceptual level*, independently from the platform of choice. To do so, it is possible to take advantage of the already defined architectural models at PIM level to collect any potential design decision. If specified in the form of a concrete vocabulary, constraints and rationale, these design features compose what traditionally has been recognized as **architectural style** [208]. The process for including architectural style features in the architecture is known as *superimposition* [105].

Since the *ArchiMeDeS* framework follows a MDE approach, it is proposed to use model-driven techniques to automate the superimposition of architectural style information to the already specified service-oriented PIM models. The increase of automation comes mainly from the fact that the step from the original architectural model to the 'enriched' one is may be partially performed by using **executable model transformations**. More specifically, the solution proposed is based on two concepts from the MDE domain: *model transformation* and *model weaving*. This way, the problem stated previously is translated into a model engineering problem and, therefore, it is possible to benefit from existing tools in the field of MDE to solve it.

3.2.3.3.1 Development Strategy

As it has been referred before, the solution introduced to drive the merging ('*weaving*') process for including the features of architectural styles into service-oriented architectural models is based on the idea of **modelling architectural styles plus the use of weaving models** [126].

When using SOC as architectural paradigm of choice, the adopted strategy relies on the concept of **service role**, understood as *the definition of the subset of functionality that a service may provide in a concrete moment*. Using a sport metaphor, if players could be represented as services, where their functionalities include both attacking and defending capabilities, the role played in each moment (playmaker, blocker, defender, assistant, etc.) would constrain the actions a player might perform at each moment (see Section 5.3 for more details on this example).

As it can be extracted from previous sections, service roles may act as the main behaviour enactors in a service-oriented architecture; however, on the contrary to other paradigms, those roles do not represent architectural elements but the set of attributes or operations a service can offer/use in each moment. By using an approach based on the use of service roles, it is possible to allow the separation of concerns needed to set, on the one hand, the *architectural elements* present in a system configuration (services, service contracts and so on) and, on the other hand, the *architectural vocabulary and restrictions* that have to be fulfilled by means of defining the allowed functionalities provided by services at concrete moments (via the specification of the corresponding service roles).

The architectural model defined at PIM level (see section 3.2.1) is already based in SOA. Therefore, there is no need for defining special processes to include current service-oriented design strategies [59]. As a consequence, the efforts must point at including traditional architectural styles into service-oriented architectural models. This end is introduced in the next subsection.

3.2.3.3.2 Traditional Architectural Style Modelling in *ArchiMeDeS*

Any software artefact found in the context of MDE is considered as a model and the set of constraints directing its definition as a *metamodel* [203]. This means that the architectural style must be a model or part of it. As it was stated at the beginning of this dissertation, the vocabulary used to describe an architectural style can be understood from a component-and-connector point of view. Consequently, an architectural style can be represented as a model conforming to a metamodel specifying the component-and-connector scheme concepts. For that reason, it is possible to state that the definition of a generic component-based metamodel serves to model any kind of **traditional architectural style** following a component and connector scheme. Figure 3-15 shows this metamodel.

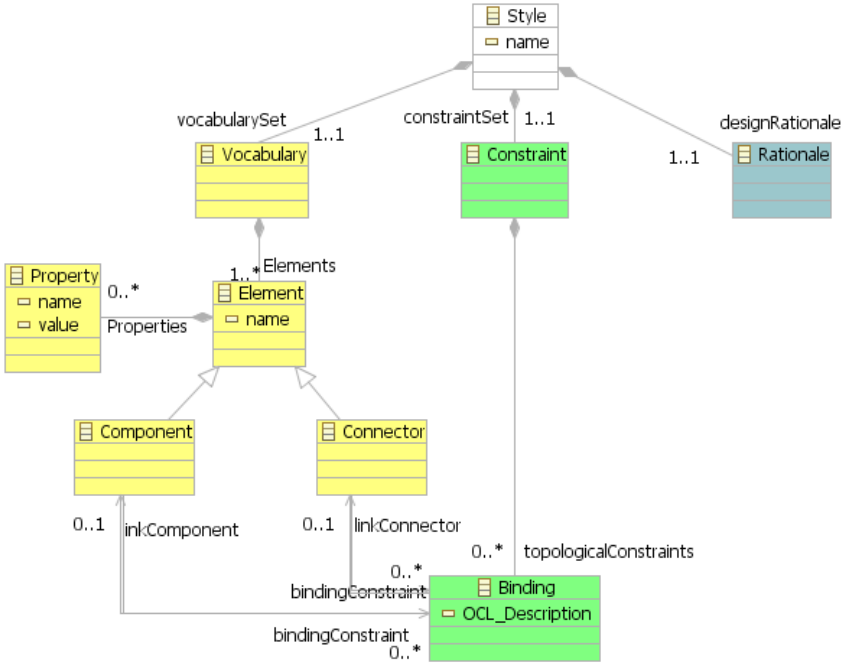

Figure 3-15. Architectural styles metamodel.

3.2.3.3.3 Definition of Architectural Styles within Service-Oriented Architectures

To add the information defined in an architectural style into a service-oriented model of the architecture it is compulsory to define how both its *concrete vocabulary*, *constraints* and *design rationale* fit in the concepts of the SOC paradigm.

- **Vocabulary definition**. As it has been stated previously, services act as independent entities capable of participating simultaneously in multiple service compositions. The functionality associated to a service is embodied in the operations it offers to other entities. In order to assign a precise behaviour while maintaining its independence, elements identified by the architectural style must be placed on a concept associated to a service, but not directly over it. The strategy to follow in order to achieve this is to restrict the available operations of a service by allowing only those that match the behaviour imposed by the architectural style. The only feature of the service which affects to its

behaviour (the portion of functionality it may offer at a given moment) without modifying its semantics (the potential capabilities it has as computational entity) is the concept of **service role** as it was explained before.

The features given to components or connectors defined by an architectural style are, therefore, applied through the definition of roles associated to services within an architectural model. By following this approach the independence of the services is maintained but allowing for the specification of common behaviours and structures in concrete scenarios according to the vocabulary specified in any architectural style. This approach can be used to superimpose traditional architectural styles on the architectural model while preserving the capability of using current SOA design patterns.

Due to the fact that *ArchiMeDeS* follows a MDA approach, the vocabulary of the architectural style can be represented by means of a concrete model that will be merged (weaved) with the architectural model of the system. This merging transformation will produce an architectural model where each service has been assigned a service role according to the vocabulary and restrictions collected in the architectural style model.

It is important to remark here that not all traditional architectural styles can be mapped to a component-and-connector scheme. Therefore, in many cases, connectors will be defined implicitly as a set of functionality derived from the application of a concrete vocabulary to a specific service-oriented model. As an example, if taking into account the *layered* architectural style, the identified components will be the *layers.* The element assimilated to connectors will be the interface provided by all the elements belonging to the same layer. This issue can be modelled either by creating a new modelling element adopting this behaviour or it can be understood as a connector created implicitly by *contextual reflection* [147] deduced from the behaviour portrayed by the individual components.

- **Constraint definition.** Constraints in an architectural style refer not only to which element can be connected to which other, that is, constraints about architectural topology (e.g. in a pipe & filter scheme, filters must connect to each other by means of pipes) but also refer to the protocol that should be used to communicate through the communication channels (e.g. in a event based scheme the architectural components are notified only if they are subscribed to a specific event). So, in the case of service-

oriented architectures we face a double challenge: to constrain the allowed communications among services and to establish a way to enforce the communications following a concrete pattern.

In SOA, communication between services is accomplished through the establishment of different service contracts understood as the agreed conditions (for example, the message exchange pattern for each provided/consumed operation) under which the communication will take place. In order to restrict the allowed interactions, a service-oriented architectural model following a specific architectural style must properly check the established contracts among services, or more specifically, among the roles played by the contracted services.

In order to include the constraints imposed by the architectural style in the architectural model, those restrictions must be placed as **model checking operations** during the merge between both models or in a separate process afterwards.

According to the capabilities assigned to the model transformations, three different approaches can be followed: model checking notifying style infractions, style checking with suggestions for style conformance, and dynamic reconfiguration (modifying the architectural configuration during the transformation process) of the architecture in order to comply with the architectural style restrictions. However, in order to be able to process those constraints, the information about the behaviour of the system is needed in this point. It is important to remark here that this aspect is currently an ongoing work.

- **Design rationale.** This concern is also one of the cornerstones of architectural specification as it refers to modelling the reasons behind making an architectural design decision. Although this aspect is beyond the main objectives marked for the current Thesis, there exist some works in this line [30][188][158] including some of the ones analyzed in the state-of-the-art chapter [160].

3.2.3.3.4 Process for Weaving Architectural Styles into Service-Oriented Architectural Models

Previous subsections have presented the way traditional styles are modelled within *ArchiMeDeS* (using a generic DSL for that aim) and how these concepts will be supported in the service-oriented architectural models (via the use of service roles). Now, it is the time to specify how both elements, together with the use of a PIM architectural model, can result in an **annotated architectural model**

containing the architectural style information. This model is considered as an "enriched" PIM architectural model since it prepares the model to further refinements at lower levels (PSM). In them, the modelling of architectural design strategies may pave the way to concrete implementations with services (such as service *mashups* based on service coordination).

The generic strategy to follow to obtain this "*Enriched PIM architectural model*" can be seen in Figure 3-16. It is remarkable the fact that the resulting model is also defined with the proposed PIM DSL and so it must conform to its metamodel.

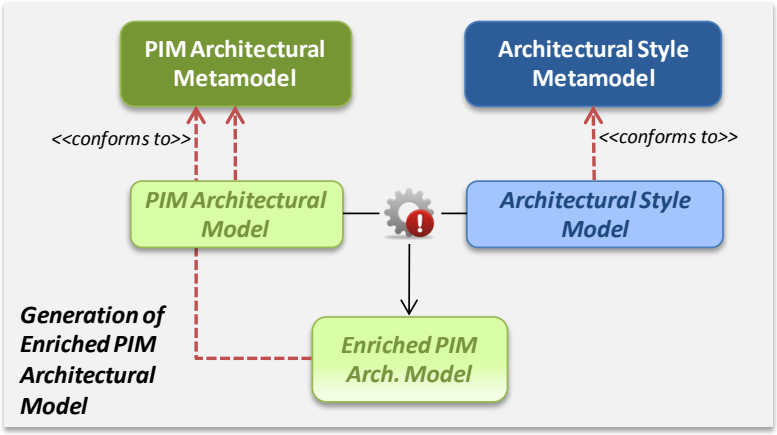

Figure 3-16. Strategy to superimpose architectural styles on PIM models.

In the context of *ArchiMeDeS* it is proposed to use a **weaving model** to capture the architect decisions on how the selected architectural style has to be applied over the service-oriented PIM architectural model. In fact, since a particular architectural style might be applied in different ways over a given PIM model, it is mandatory to indicate the way that architectural style is particularly considered for that architectural model. For example, given the pipe and filters architectural style and given a concrete PIM model, each service might act either as a pipe or as a filter. Thus, when superimposing the pipes and filters style over the PIM model, it is compulsory to state clearly which services will act as pipes and which as filters. Once it has been decided, the next step is to obtain the final '**Enriched PIM model'** from the information gathered in the original PIM model, that from the architectural style model and, finally, that personalized mapping from elements in both models. All this given, the process for including the features of an architectural style in a service-oriented architectural model can be seen in Figure 3-17.

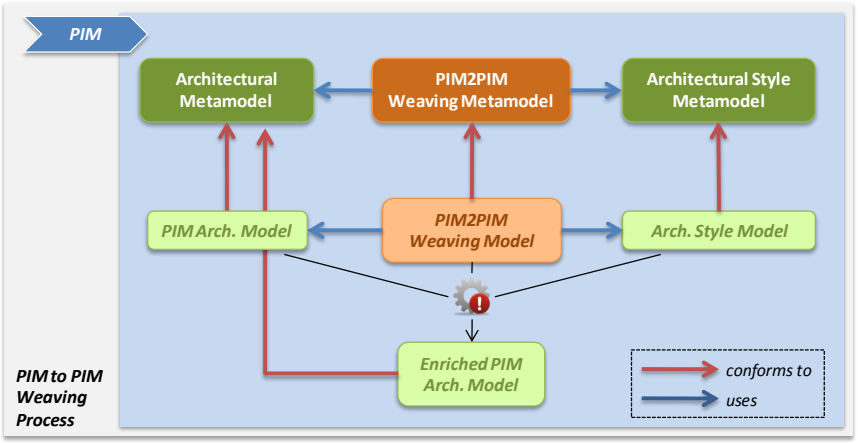

Figure 3-17. Process for Weaving Architectural Styles and Architectural Models.

As it has been pointed out before, the resulting model is an enriched Service-Oriented PIM architectural model ('*Enriched PIM Arch. Model*' in the previous figure) where each service has been assigned a role, according to the chosen architectural style. Since the roles that a *service* instance may play depend on the operations its *service type* defines can play those roles, the resulting enriched PIM model will observe:

a) That *service roles* corresponding to the vocabulary of the architectural style will be necessarily associated to a *service type*. If this service type does not exist, the transformation process will compulsory create them.

b) That *service roles* will be played exclusively by service instances (and not service type elements). If the annotated elements in the weaving model are service types, the transformation process will mandatory create them and associate both the corresponding service role and the generic service type with all the features of the original service instance.

To sum up, in the context of the proposal presented in this dissertation, the option chosen for superimposing architectural styles in architectural models has been to separate the management of the architectural style vocabulary from its style constraints. The former is encoded in the architectural style metamodel. The later, although it is a work still in progress and has not been included as part of the Doctoral Work, should be encoded in the model transformation. Once the '*Enriched PIM architectural model*' has been obtained, one may want to check if that model conforms to such restrictions. Given that here it is proposed to follow and apply model-driven techniques, this checking process would also need to be encoded in a model transformation, whose output will be one of three possibilities:

a) a kind of report about the conformance of the Enriched PIM model obtained to the chosen architectural style; b) a set of suggested element modifications to the output model; or, c) an automatic reconfiguration of the architecture in order to comply with the constraints defined by the architectural style and according to the weaving model.

3.3 ArchiMeDeS as part of an Architecture-Centric Model-Driven methodological framework

One of the main objectives of the current dissertation is to present *ArchiMeDeS* as model-driven framework for the specification of software architectures. However, although the research was initiated as part of a much wider effort for software development (the MIDAS methodological framework [228]), *ArchiMeDeS* also aims to be considered as a complete and independent framework capable to be adapted to any other MDA-based development methodology. To overcome this challenge, and once the semantics and syntax of the DSLs allowing the definition of software architectures have been presented, it is time to explain the role of *ArchiMeDeS* in a concrete MDA-based development context. It will serve to show how the architecture may play a central role within the specific model architecture, thus defining an *architecture-centric* development process.

More specifically, this section is devoted to explain the relationships established between the *ArchiMeDeS* framework and the elements defined by the MIDAS methodological framework for software development. To start, as it was clearly stated in the introduction, MIDAS is a methodological framework whose main features can be summarized in the use of a model-driven approach for software development and whose model architecture is divided into several abstraction levels and concern layers. One of these layers is the one dedicated to the specification of the system architecture and the object of this Doctoral Thesis.

As stated above, the purpose of the current section is to show the dependencies and influences that the constituent elements of the *ArchiMeDeS* framework may have over other parts of MIDAS and vice-versa. In particular, the sources of information that the architectural models may have from upper abstraction levels (i.e. from models at the CIM level) and the needs for behavioural modelling that the architectural models may cover.

In addition, since the architectural layer is considered to be the driving aspect for the whole methodological framework, next subsections will also try to clarify how the elements that appear in the architecture (and its relationships)

helps deciding which models and elements inside models should be created during a system development process within MIDAS.

3.3.1 Information Sources for Architectural Modelling

Throughout the entire dissertation, the architectural models specified have been placed either at a conceptual level of abstraction (PIM level) or at a more technological one (either PDM or TDM) according to the MDA proposal. The analysis of the sources of information for the specified architectural models, in the scope of MIDAS, should be divided, therefore, in two parts: first the origins of the PIM models and, next, the ones that allow completing the PSM level models.

3.3.1.1 Sources for PIM Architectural Modelling

The information that is needed to specify the architectural models from a conceptual point of view, like the one represented at PIM level, comes mainly from the CIM level. According to the MDA proposal, it represents "*a view of a system from the computation independent viewpoint. (…) A CIM is a model of a system that shows the system in the environment in which it will operate, and thus it helps in presenting exactly what the system is expected to do*" [170]. Such model usually includes is sometimes a domain model or a business model that gather the requirements of the system describing the situation in which the system will be used. Within the scope of MIDAS, this abstraction level defines two separated models:

- A **Value Model** that allows the specification of those value objects (goods, money or services) that are created and provided by any of the members that intervene in a business (*business stakeholders*), as well as the business members that are interested in those value objects. The exchange of value objects between each business member is also modelled.
- A **Business Process Model** that is used to understand and describe the business processes related to the environment in which the system will be used. This model identifies the business services that will be offered to end consumers or users of the information system under development.

Apart from these two models, in some contexts, it should be recommendable to define an additional model: a **domain model,** comprising the vocabulary and key concepts of the problem domain. This model may be needed to verify and validate the understanding of the problem domain among various stakeholders. Regarding its influence on the architectural modelling, the concepts depicted in that model may be used to extract the vocabulary used to name every element of the architectural model.

From the analysis of both *Value* and *Business Process* models, it is possible to identify what *Service Providers*, either inner or outer, might be modelled as part of the PIM architectural model, and so the *Business Contracts* that should be established to communicate them. In addition, by knowing the *dependency paths* represented in the *Value model* it is possible to identify the needs of the final user of the system. The idea is that these needs will generate business services for supporting those necessities. Once the business services that the system will support are selected, it is possible to identify the *Services* (or *Service Types*) that will be present in the PIM architectural model. The conditions under which these services communicate (that is, the features of the *Service Contracts* established among them) can be derived from the *Business Process model* according to the business activities and tasks described. Figure 3-16 resumes these influences on the PIM architectural model.

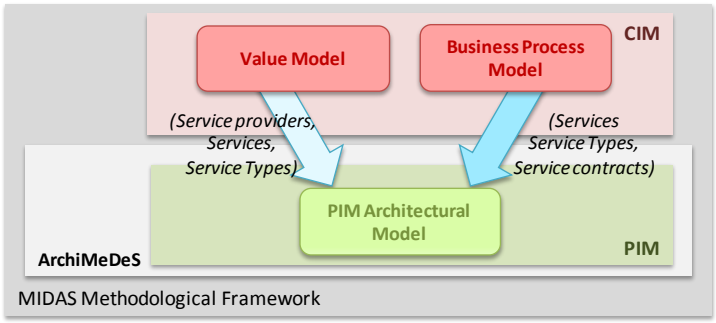

Figure 3-18. Influences on the PIM architectural model.

3.3.1.2 Sources for PSM Architectural Modelling

The main source of information for the PSM is the PIM model of the architecture. In that sense, the execution of the transformations defined for that aim facilitates to obtain a prototype of the PSM architectural model, either at PDM or TDM levels depending on whether the concrete service technology has been chosen or not. However, there are some features that cannot be obtained directly from that model. To overcome these flaws it is possible to look at the behavioural models defined in the MIDAS framework. The process for obtaining these behavioural models is known as SOD-M [45]. Figure 3-17 resumes these influences on the PSM architectural model

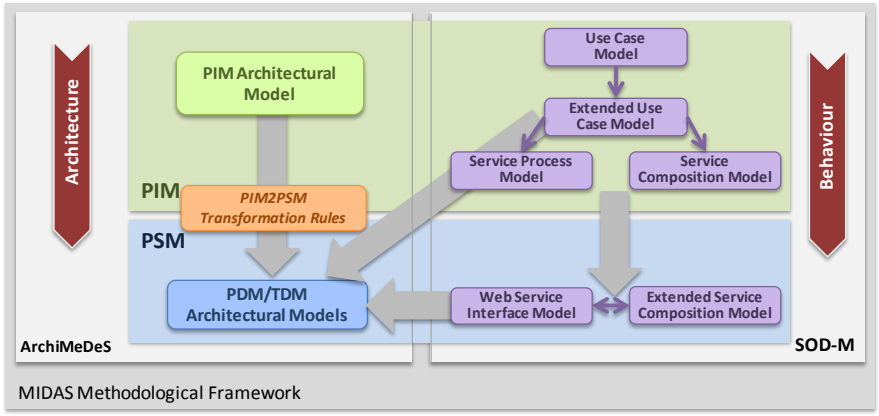

Figure 3-19. Influences on the PDM/TDM architectural models.

3.3.2 Influence of Architectural Modelling over Other Development Concerns

As stated previously, the *ArchiMeDeS* framework also aims to provide a way to convert MDD methodologies into Architecture-Centric MDD methodologies, that is, driving the development process throughout the information gathered in architectural models. Since it would require an in depth analysis of the features and characteristics of any current model-driven methodology, this dissertation will only cover the steps needed to be able to consider MIDAS as an ACMDD methodological framework. In that context, the architecture is said to drive the development process and thus it has a key influence over other tasks and models that take part in MIDAS. Though it is still a work in progress subject of strong research efforts, some clues in that direction will be explained in the following:

- **Influence on the Storage concern**. The content concern in MIDAS refers to models that collect the persistent information of the system under development. For that reason, if the architectural model includes the specification of resources it will almost certainly mean that a **data model** is needed, either as a conceptual model (at PIM level) or as a logical one (PSM).
- **Influence on the Behavioural concern**. The influence of the behavioural concern over the architectural models has been pointed out in the previous subsection. From the other point of view, some decisions that are taken during the SOD-M modelling process require information from the architectural aspect. An example can be seen in the moment of

creating the Extended Service Composition Model that needs to identify the *possibility of communication* between two or more services. This information is collected in the architectural model by means of established service contracts.

- **Influence on the Interface concern**. The modelling of the interface concern in MIDAS includes the specification of *navigational* and *fragments* models [35]. As it happened with the storage concern, the interface concern will need a concrete modelling activity when the architectural model includes interaction services.

It is important to remark that the detection of modelling activities in any of the development concerns of MIDAS is not only a matter of the existence of one or another element within the architectural models. In contrast, it can be seen as a consequence of the consideration of the architecture as a crosscutting aspect during the development process. In fact, in *ArchiMeDeS*, the architecture is a high-level description of the system which guides the rest of the development process and plays the central role in the MIDAS model architecture. In fact, the proposed architecture viewpoint is the "map" to provide the structure of the model architecture, deciding which views ('*concerns'*) are instantiated and which of them are not.

3.4 Concluding Remarks

Previous sections have shown the features and main characteristics of the *ArchiMeDeS* framework. To do so, a set of DSLs for service architectures has been defined. As the framework follows the principles of the MDA approach for the development of software architectures, the corresponding PIM and PSM level metamodels have been described, both at the PIM level of abstraction and the PSM one. For the latter, three different target service platforms have been chosen: Web Services, Grid Services and REST services.

To that end, both the abstract syntax (semantics) and concrete syntax (notation) of the DSLs have been completely specified. Conversely, the transformations from models of one abstraction level to the others have been also specified.

In order to provide the architectural models with design flexibility, the support for the inclusion of architectural styles has been explained, both for traditional architectural styles and for those coming from the current design strategies with the SOA approach.

In addition to the description of the inner elements and features of the *ArchiMeDeS* framework, it has been explained in detail the importance of that

framework within the scope of a broader methodological framework for software development as it is MIDAS. The influence of other models defined within MIDAS has been depicted, taking special care on the information flow that arises between the behavioural models of MIDAS and those of the architecture.

The completeness and validity of the DSL, the correctness of the models, the implementation of the transformations and its feasibility as framework for the specification of software architectures using a service-oriented and model-driven approach will be the main topic of the next chapter.

Chapter 4:
The ArchiMeDeS Toolkit

The development of modelling toolkits plays a prominent role within model-driven engineering approaches [204]. Thus, the implementation of a toolkit associated to the DSLs and model transformations defined in the previous chapter represents an additional feature towards considering *ArchiMeDeS* as complete framework for software architecture specification. Counting with a toolkit for that aim provides, among others, the following features:

- It provides **support for the creation and editing of models** with ease.
- It represents a valid strategy for **verifying the coherence and consistency of elements and relationships modelled**. This is possible due to the fact that metamodels gather the syntactic rules that DSLs define and that models must obey. A common feature of modelling environments is conformance verification.
- It allows defining **a common implementation for the DSLs and adopting it as reference implementation,** granting a degree of interoperability with other proposals in the field of MDE.
- It offers **support for transformation rule implementation and execution.** The tasks of verifying the conformance of the resulting models can be automated but, again, it does not mean that the semantic associated to the transformation is appropriate. This task has to be accomplished by hand as part of the research work.

The following subsections will explain, first, the strategy followed to design the toolkit by focusing on the design decisions taken. Once the architecture of the toolkit is clearly defined, the subsequent section will focus on the development of the individual modules. It will comprise: the creation of a plug-in for managing the abstract syntax of each DSLs separately (including that for architectural style modelling); the implementation of a plug-in supporting their concrete syntax by defining the graphical notation for each DSL; and, the implementation of the model transformations and the creation of a weaving environment for the superimposition of architectural styles on PIM models.

4.1 Toolkit Design Strategy and Architecture

The strategy followed to build the *ArchiMeDeS* toolkit is based on the guidelines defined for M2DAT [227], a technical solution for the model-driven development of Web Information Systems. Accordingly, the development of a toolkit supporting the *ArchiMeDeS* proposal is divided in two steps. The idea is; firstly, to establish the architecture of the toolkit, that is, its *conceptual design*. Afterwards, a more *technical design* is undertaken, were concrete technologies,

languages and platforms are used to implement the former conceptual architecture. In addition, it is worth to mention that the toolkit development process follows an iterative and incremental approach so it has been possible to update the toolkit in parallel with the definition of the DSL proposed by *ArchiMeDeS*.

4.1.1 Conceptual Design

The *ArchiMeDeS* toolkit is comprised of a set of modules, one for each DSL defined by the *ArchiMeDeS* proposal to specify Software Architectures using services. Since each model is defined using a DSL (insights on the motivation behind this decision were given along the previous chapter), the toolkit to develop will be understood as a kind of workbench allowing to work with those DSLs independently. For that purpose, each part of the tool provides with the functionality needed to handle models corresponding to any DSL, like model editing or conformance.

The conceptual architecture of the toolkit (shown in Figure 4-1) has been defined following a traditional layered approach [183]. Within the scope of MDE, separation of concerns allows for distinguishing the presentation of each model from the model itself [117] and so it will be reflected in the architecture of the toolkit. Though it will be later shown how this is semi-automatically provided by EMF [29] (and thus inherently supported in the *ArchiMeDeS* toolkit when introducing its technical design) it is possible at this point to conceptually establish the contents of each tier.

The **presentation** tier includes the editors (whether they are diagrammers or tree-like) to work with each type of model supported by the toolkit. This tier will define, accordingly, the interfaces allowing to edit, create, manage, etc. the models for architecture modelling with services, either at PIM or PSM abstraction level. However, since the *ArchiMeDeS* framework also allows for the definition of an architectural style to be superimposed with a service-oriented architectural model, this feature will be also part of the interface set. In addition, this tier will also comprise a specific interface to execute model transformations, to include the information from the transformation rules defined for either evolving from the PIM abstraction level to the PDM DSL or to any of the TDM DSLs defined.

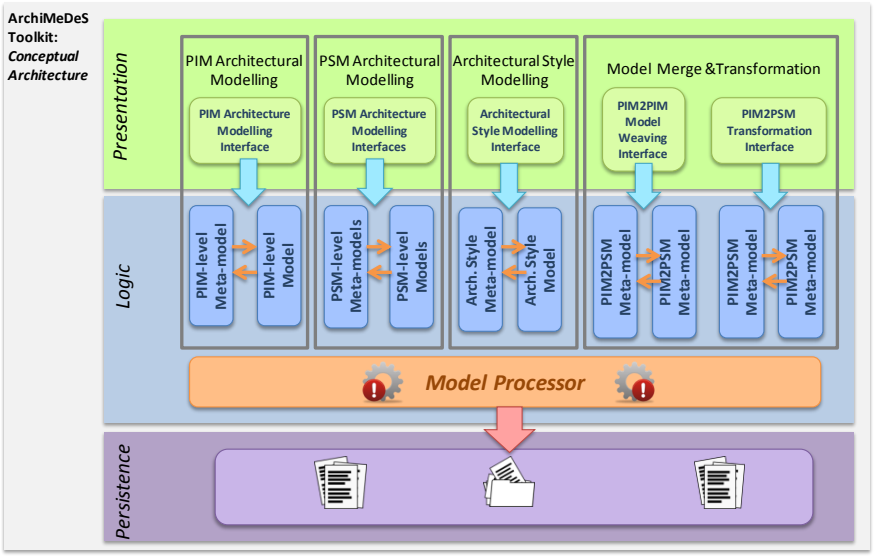

Figure 4-1. ArchiMeDeS toolkit conceptual architecture.

The **logic** tier, in turn, will be responsible of handling both the models and the potential execution of transformations among them. The *ArchiMeDeS* framework defines several related DSLs that should be integrated. To manage the architectural specification in models, this tier will need to include the actual representation of these DSLs in the form of metamodels and be able to handle models that conforms them. Moreover, it needs adding support for, at least, model transformations to connect the different DSLs and, due to possible variations on the architectural design, it is considered to supply with model weaving capabilities. In addition, the capabilities that a DSL workbench should support consist not only in a graphical editor or model generation capabilities. From now onwards, the term *model processing* will be used to refer to all these tasks, following the idea expressed in [227], to refer to all the tasks related with model handling. As a result, the module comprising all these functionalities will be called *model processor(s)* which will be the main element inside the logic tier.

Finally, models should be somehow stored for a later management. The **persistence** tier of the *ArchiMeDeS* toolkit is a file system that incorporates traditional versioning policies. The use of more complex storage systems (such as XML Databases) is eventually discarded since, at the moment, it just brings complexity to the development of the toolkit.

The following section details how this conceptual architecture is mapped into a technical design using concrete languages and technologies.

4.1.2 *Technical Design*

Once the conceptual architecture of the toolkit supporting the *ArchiMeDeS* framework features has been defined, the next logical step is to select the approaches and technologies to be used in order to obtain a complete specification of the toolkit. That means that, for each model operation, it is necessary to select the existing tool or component supporting such task that best suits the *ArchiMeDeS* needs. To provide with a brief overview on this selection of technology, Figure 4-2 shows the main components used to deploy the modules to support DSLs and model transformations. At this point it is important to note that each component and the decision to use it will be described and justified in the following subsections.

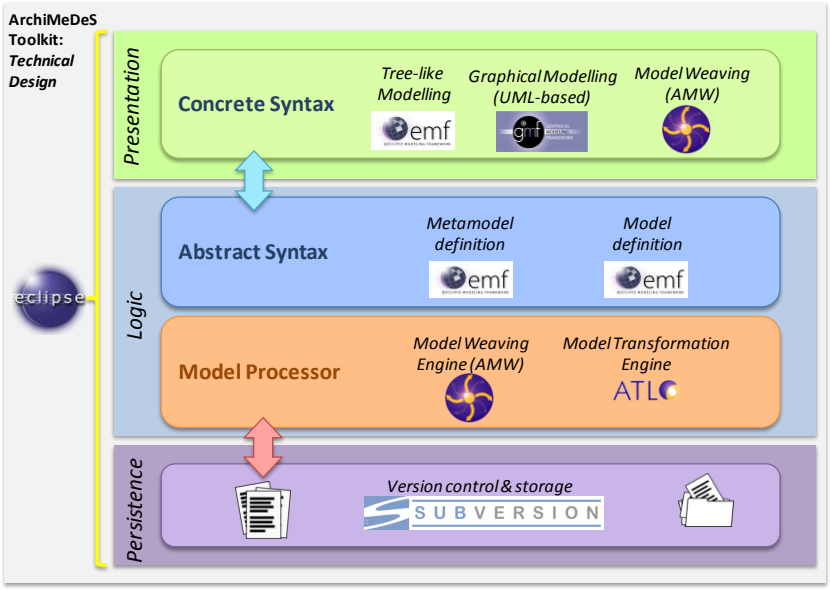

Figure 4-2. ArchiMeDeS toolkit technical design.

As it can be seen, the interfaces that were identified in the conceptual architecture as comprising the presentation tier, will contain the concrete syntax of the DSLs (either using a *tree-like* or *UML-based* working layout with EMF or GMF respectively) and the environment needed to specify model transformations and its personalization (via the ATL/AMW environment).

The idea is that it is at this presentation tier where the **concrete syntax** of the DSLs defined within the *ArchiMeDeS* framework come to play. Though EMF offers a common interface for creating and editing models using a tree-like

structure, to provide with more user-friendly graphical features to work with the architectural models it is necessary to support means for handling nodes, edges, arrows and boxes comprising the UML notation proposed for each DSL. This is accomplished by means of using another widely know extension of the Eclipse platform: GMF, built on top of the EMF infrastructure.

The **abstract syntax** of the DSLs proposed is specified using the features offered also by the EMF, which provides with a programmatic outline to the aforementioned metamodels. The result of the creation of the metamodels using EMF is a set of .ecore files that contain the XML code corresponding to the semantics expressed in the metamodels. The XML follows a structure specified in the EMF core metamodel, that is, the metamodel expressed with EMF acts as a model following the EMF metamodel capabilities. By means of creating an EMF-based representation of the metamodels the Eclipse framework allows to build up a plug-in for the Eclipse platform that permits the edition of architectural models that conform the *ArchiMeDeS* DSL metamodels.

With the EMF framework it is possible to specify, using an XML-based schema, the metamodels that correspond to the PIM DSL, the PSM DSLs and the DSL that allows defining the architectural styles.

This abstract syntax is used as input or output in any model processing task. Thus, the lower level of the toolkit is comprised of the **model processor**. Apart from the kernel of the Eclipse platform, it embeds all the modules allowing to perform several operations related to model management:

- **Model transformations** are developed making use of the ATL language and its corresponding execution plug-in. The rules defined to evolve from PIM architectural models to a target PSM platform are implemented and executed using a specific plug-in for the Eclipse platform.
- **Model annotation** is supported as a way towards introducing design decisions into architectural models, i.e. merging architectural styles with conceptual architecture models, without reducing the level of automation. The module for model weaving support is based on the *ATLAS Model Weaver environment* (AMW) [54]. This way it is possible to define weaving models that are used as annotation models that drive the merging process. Those models plus the weaving models that link them are later processed by ATL model transformations.

Finally, the **persistence tier** is developed using a traditional versioning system. In particular, it uses an application of the *Subversion* [190] system adapted to work as a plug-in for the Eclipse workbench named *Subclipse*.

All in all, the developed toolkit represents a framework that integrates some tools supporting each specific modelling task in order to obtain an efficient toolkit. The modularization that has been accomplished by using the Eclipse platform also favours the inclusion of future refinements of *ArchiMeDeS*.

4.2 Module Implementation

Previous sections defined the architecture of the toolkit either form an abstract point of view or focusing on specific technologies. Accordingly, in order to build the toolkit itself, a number of modules should be implemented following a coherent development process. The process followed is an adaptation of the one presented in [227]. To illustrate it, Figure 4-3 outlines the main steps given, tasks performed, derived artefacts and technologies used in each case.

The process can be divided into four main tasks:

1. **Abstract syntax definition**. The first step is to define the abstract syntax of every DSL that the *ArchiMeDeS* framework proposes. This implies that it will be necessary to implement the metamodels that correspond to the PIM level, the PDM metamodel and the TDM metamodels of each target platforms (Web Services, Grid Services and REST services). At the PIM level, also the generic metamodel used to represent any architectural style model is defined. To get all these artefacts, the EMF plug-in for Eclipse is used in terms of an *.ecore* file for each DSL.
2. **Concrete syntax definition**. For each metamodel developed a concrete syntax is provided. This way, it is possible to work with a graphical interface allowing to manage the models created conforming the previous metamodels. The EMF used for the abstract syntax definition already offers an initial tree-like modelling environment; however, since the final idea is to ease the architecting process with models, the graphical notation (UML-based) that was defined within the *ArchiMeDeS* framework is put into practice. For that aim, the GMF Eclipse extension is used to support the use of the UML profiles defined within the *ArchiMeDeS* proposal. On the contrary, for easing the transformation process, the concrete definition of the weaving models will be defined by means of the capabilities provided by the AMW framework.
3. **Definition of model transformations**. The third step requires the implementation of the transformation rules that link the different metamodels, either at different abstraction levels (PIM to PSM) or at

the same level (to include the architectural style features into the PIM-level architecture model). For each transformation set the tasks to accomplish include:

a. The definition of the transformation rules in natural language (see section 3.3 for this content).
b. The translation of these rules to the ATL language so it is possible to implement them with the corresponding Eclipse plug-in.

The development of each step is explained in detail putting a special emphasis on the artefacts obtained as output of each activity in the following.

4.2.1 Modules for the Definition of the Abstract Syntax

Recalling the methodological decision taken for the development of *ArchiMeDeS*, it was argued in favour of following a hybrid approach, combining the specification of DSLs with a UML-based notation for modelling service-oriented architectures. Accordingly, the technology of choice to implement the proposal should focus on the definition of DSLs but, also, to support UML modelling. That circumstance is one of the first points in favour of using the *Eclipse Modelling Framework* (EMF) environment of the EMP. According to the Eclipse Website[1] EMF is a modelling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. Models can be specified using annotated Java, XML documents, or modelling tools like Rational Rose, then imported into EMF. Most important of all, EMF provides the foundation for interoperability with other EMF-based tools and applications. Other features of EMF can be pointed out:

1. **Combined support for DSL specification and UML modelling**. EMF is a DSL toolkit [29] that supports UML modelling since the EMP includes the UML2 sub-project, an EMF-based implementation of the UML2 standard [175]. The DSLs are developed over the same meta-metamodel that the UML2 sub-project. Since the meta-metamodel is the same, it is possible to bridge the gap that may exist

[1] *Eclipse Website: http://www.eclipse.org/*

between the architectural (meta-)models and those of the UML Standard [ref]. Thus easing the tasks for providing support for having a UML-based modelling process of service-oriented architectures.

2. **Interoperability**. The development of a toolkit as reference implementation of a proposal should not be constraint to a concrete scope. It should allow, at least, model exchange so it is possible to bridge the gap that traditionally has been detected between the standards and theoretical solutions and the actual implementations [24]. The tendency is to agree in a common implementation sufficiently close to the standard and implement it as reference implementation for model exchangeability. Currently, most of the initiatives performing tooling activities develop their prototypes using EMF as metamodelling framework. Accordingly, the selection of EMF for implementing the *ArchiMeDeS* toolkit leverages the level of interoperability so it is possible to import/export other EMF works within the *ArchiMeDeS* framework.
3. **Extensibility**. The *ArchiMeDeS* proposal is not a closed framework and its features have evolved since its initial conception. In the future it is also likely to scale to adopt (and adapt) other architectural concerns. In that sense, having a tool support that can be extended to support new capabilities is a mandatory requirement. In that context, both EMF and the Eclipse platforms also seem to be adequate solutions for that aim. Indeed, Eclipse is conceived as an extensible framework that provides with the basic infrastructure to be extended and was thought to that end. Likewise, EMF itself is also an open framework that is permanently evolving and incorporating emerging technologies.

4.2.1.1 Metamodel Implementation with EMF

Metamodels containing the concepts of each DSLs defined as part of the *ArchiMeDeS* framework has been implemented with EMF. In particular, the metamodels implemented include:

- PIM architectural metamodel.
- PDM architectural metamodel.
- TDM architectural metamodel for Web Services.
- TDM architectural metamodel for Grid Services.
- TDM architectural metamodel for REST Services.
- Architectural style metamodel.

Since the development process of the modules supporting any of the previous metamodels is analogous, from now onwards, the explanation will focus on the implementation of the PIM metamodel module to illustrate that process.

In essence, EMF can be thought as a highly efficient Java implementation of a core subset of the MOF API [172]. EMF provides a kernel structure of a meta-model description, named *Ecore*. *Ecore* is a simplified implementation of EMOF (*Essential MOF*) that generates a file with *.ecore* extension where each of the meta-classes specified in the metamodel is implemented, with the particularity of being self-descriptive. The basic structure in *Ecore* can be seen in Figure 4-3.

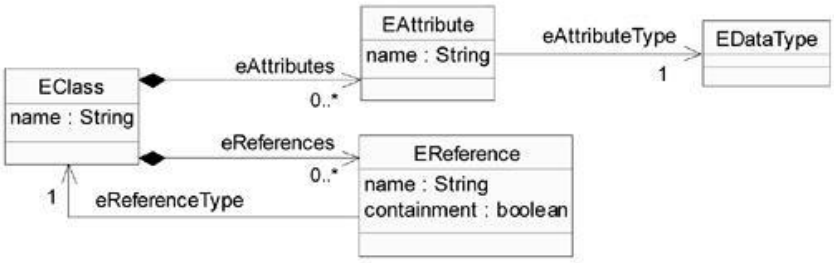

Figure 4-3. Ecore basic structure (metamodel excerpt).

- **EClass** represents a modelled class.
- **EAttribute** represents a modelled attribute in a modelled class.
- **EReference** represents one end of an association between classes.
- **EDataType** represents the type of an attribute.

Using the basic structure of Ecore, it is possible to define any desired structured model. Every metamodel is, therefore, implemented in terms of *Ecore*, the meta-meta-language of EMF. It is important to note here that, due to the XML basis over which *.ecore* files are built, any metamodel defined with Ecore must have a root element. This can be seen in Figures 3-3, 3-5, 3-6, 3-7, 3-8 and 3-15 where their content shows a root element painted in black. In the PIM metamodel, for example, the root element has been named "*PIM_Architecture*".

The *.ecore* file, which represents a metamodel based on EMF (i.e., the abstract syntax of the DSL defined), will be used as source file by the EMF environment for the generation of a simple tree-like model editor. This editor represents the starting point for the creation of models conforming to the aforementioned metamodel. However, in order to provide with a proper model editor, it is necessary to define several additional models codifying the relationships between the meta-concepts of the metamodel and the elements they

represent. The most important among them is the *genmodel model*, known as the *generating model.* This model is compulsory for generating the code corresponding to the desired model editor. To illustrate the differences among the *.ecore* and the *.genmodel* files, Figure 4-4 shows the relationships among the information stored by each file. This separation of the *genmodel* from the base model has the advantage of maintaining the Ecore metamodel unaltered and independent from any relevant information needed for the code generation of the model editor.

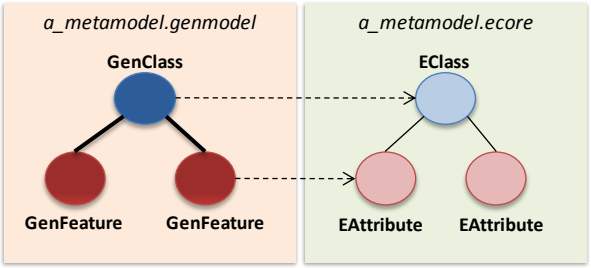

Figure 4-4. Relationship between .genmodel and .ecore models.

Figure 4-5 illustrates all the relationships established among all the elements generated as part of the generation of a model editor. The objective of each generated code can be thought as follows: the *model code* file allows to access to the metamodel (*Ecore*), to create models conforming to the metamodel and serializing/deserializing tasks. This *model code* is used by both the *edit code* files and the *editor code* files for relating the aforementioned functionalities with a graphical interface. These last elements provide with a default interface (tree-like) for model editing and management conforming to the metamodel initially defined.

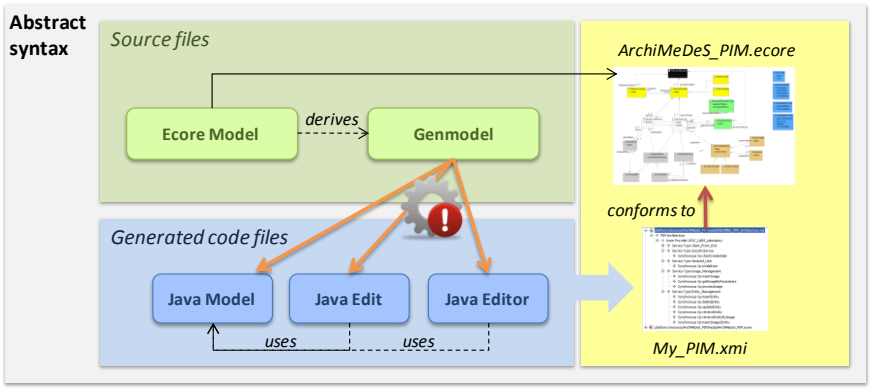

Figure 4-5. Overview of the generation process of EMF-based editors.

In summary, the procedure of code generation is: the *Ecore model* is mapped to a *Genmodel*; after that, the *Genmodel* will be used to generate code resulting in three different files (*model*, *edit* and *editor*). The final outcome is a tree-like Java-based editor, enough simple and complete to specify models conforming to a given metamodel. This way, it is possible to count with basic editors for each DSL defined in this Doctoral Thesis.

4.2.2 Modules for the Definition of the Concrete Syntax

As it has been stated previously, once the abstract syntax of the DSLs has been implemented using Eclipse, it is recommended (but not compulsory) to provide the tool with the support for using a graphical notation, thus easing the creation of the models by means of a more intuitive interface better than the tree-like option. For that aim, and similarly to the development work accomplished for the abstract syntax of the DSL defined, the concrete syntax of the DSLs created within the *ArchiMeDeS* framework will take advantage of another extension for the Eclipse platform: in that case, the *Graphical Modelling Framework* (*GMF) Project* [221] and its corresponding plug-in for the Eclipse workbench. This plug-in allows the creation of an upper-level abstraction layer for the metamodels defined in EMF. In the case of the *ArchiMeDeS* framework, a concrete GMF extension is created to give support to the UML profile stereotypes for both the PIM level and the PSM levels of the DSL language for service architectures.

GMF provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF (*Graphical Editing Framework*, [156]. Figure 4-6 shows the dependencies between those Eclipse components.

Any GMF editor depends on the GMF runtime and uses the EMF, GEF and Eclipse platform. Before the advent of GMF, an Eclipse model editor was developed by binding the EMF model with the GEF view by hand-coding. GMF undertakes this task replacing the coding by modelling to provide an easier way to develop graphical editors using GEF and an underlying EMF model [161].

The underlying idea is that a set of models serve to define the concrete visual syntax of the DSL and collect the correspondences between the EMF model (the abstract syntax) and the graphical elements. From such models, GMF generate the code that implements the graphical editor in the form of an Eclipse plug-in.

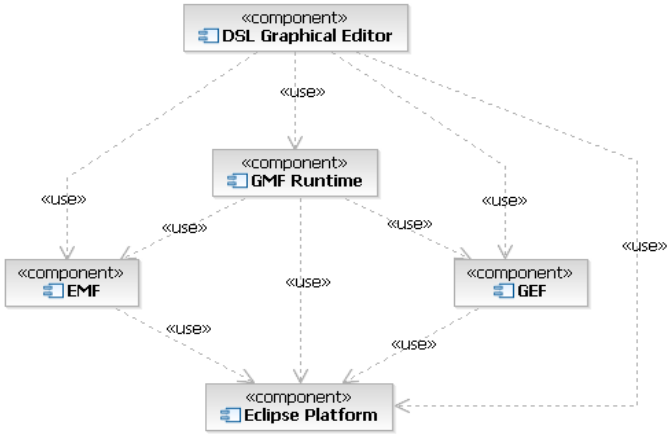

Figure 4-6. Dependencies between a generated graphical editor, the GMF Runtime, EMF, GEF, and the Eclipse Platform[2].

4.2.2.1 Graphical Support with GMF

The development process for granting a graphical interface for the *ArchiMeDeS* toolkit is depicted on Figure 4-7, detailing the different models that you should define to build a GMF editor.

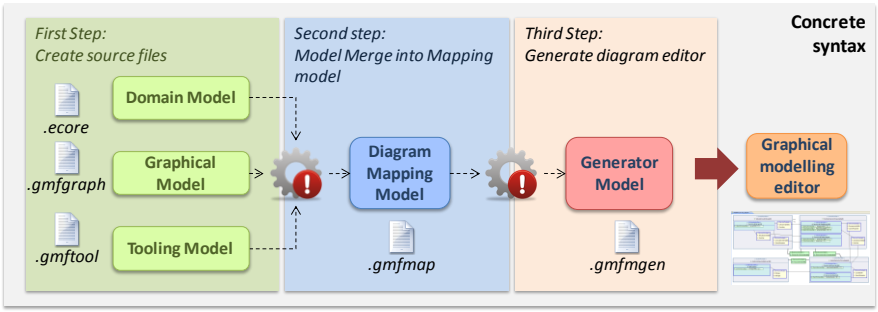

Figure 4-7.Overview of the GMF development process.

[2] *Retrieved from* http://www.eclipse.org/articles/Article-Introducing-GMF/ article.html

- **The domain model**: this is the Ecore metamodel used to define the abstract syntax of the given DSL. It defines the non-graphical information managed by the editor.
- **The graphical definition model**: this model defines the graphical elements to be displayed in the editor.
- **The tooling definition model**: it establishes the widgets that compose the user interface of the editor. In essence, it defines the tool palette.
- **The mapping model**: finally, this model links the previous models together. Graphical and tooling elements are linked with their corresponding elements form the domain model. In other works, it bridges the abstract syntax of the DSL with the concrete (visual) syntax plus the widgets to add each different modelling element to the diagram.

GMF tries to simplify the tasks of defining these models by providing with wizards that drive the user on the process to define each one. In addition, a tentative mapping model is automatically generated. It is a first attempt to match domain, graphical and tooling models. From that initial mapping, the user has the right to modify the mappings identified as needed.

Once the previous models have been defined, GMF generates a new model, known as *generator model*. This model encodes implementation details that will drive the generation of the final plug-in that implements the graphical editor.

This way, the main features of GMF are reutilization of the graphical definition for different domains and applications and automatic generation of the graphical editor: on the one hand, since the only connection between the domain concepts and its graphical representation is the mapping model, the only thing to do is to modify the mapping model to reuse the graphical abstractions already defined for any other domain. On the other hand, GMF applies MDE techniques. The diagrammer is automatically generated from a set of models applying model transformations. Actually, until recently, they were not proper model transformations since JET was used to generate the diagrammer code. As a result, GMF can be considered a perfect example of MDSD.

Finally, if the default capabilities of a GMF editor satisfy the user, it is not necessary to touch a single line of code since the whole process is automatic. However, it is still possible to modify the generated code to obtain a different *look and feel* for the editor or to add/modify the capabilities provided by GMF.

This entire process has been applied to the creation of a graphical interface for each DSL as part of the *ArchiMeDeS* toolkit.

4.2.3 Modules for Model Transformation

The advantage of using a model-driven process for the specification of service architectures relies also on the provision of support for automatic model conversion. It the context of *ArchiMeDeS* it is achieved from one abstraction level to another trough the execution of both predefined model transformations and personalized changes to the models. In order to achieve this feature, the transformation rules specified in Section 3.3 are defined in ATL and included into a concrete module as part of the toolkit.

At this point, the implementation of the *ArchiMeDeS* toolkit faces two different challenges: a) to prepare the toolkit for the support of automatic transformation PIM models into their PSM counterpart depending on the concrete target platform; and, b) to have the ability of weaving architectural style models into consistent PIM models.

4.2.3.1 Implementation of PIM-to-PSM transformations with ATL

As it was pointed out previously, the implementation of model transformation is accomplished by making use of the facilities provided by the ATL (*ATLAS Transformation Language*) [55] and its implementation over the Eclipse platform [54].

ATL is a model transformation language and toolkit that provides ways to produce a set of target models from a set of source models. Developed within the Eclipse platform, the *ATL Integrated Environment* comprises a number of standard development facilities (syntax highlighting, debugger, editor, etc.) that eases the development of ATL transformations. It is mainly based on the OCL standard and it supports both the declarative and imperative approach, although the declarative one is the recommended.

Mappings are implemented in ATL by defining a set of rules: each rule specifies a source pattern and a target pattern, both of them at metamodel level. Once the ATL transformation is executed, the ATL engine establishes matching between the source pattern and the source model. Then for each matching, the target pattern is instantiated in the target model, replacing the matching found in the source model.

In contrast with most existing languages, ATL allows for rule inheritance and provides both implicit and explicit scheduling. The implicit scheduling is supported by the imperative constructions of ATL. When the transformation starts, the algorithm starts with calling a rule that is designated as an entry point and may call further rules. After completing this first phase, the transformation engine automatically checks for matches on the source patterns and executes the corresponding rules. Finally, it executes a designated exit point. Explicit

scheduling is supported by the ability to call a rule from within the imperative block of another rule. ATL transformation descriptions are transformed to instructions for the *ATL Virtual Machine*, which executes the transformations. This is analogous to Java and the Java Virtual Machine.

4.2.3.1.1 Implementation of ATL rules using the MeTAGeM framework

In order to create the ATL code corresponding to the PIM to PSM transformation rules, the MeTAGeM [26] framework for model transformation specification has been used. For that purpose, MeTAGeM describes an ATL generation process based on a MDA approach for each transformation rule. To illustrate this process, the transformation rule that will be used for that aim is the one that allows transforming a *Service* (with the attribute *variant* set to '*information'*) defined as part of the PIM architectural model into two elements of the PDM level, a *Resource* and a *Service* elements joined by a '*controls'* association. The illustrative example is applicable to any kind of transformation defined within *ArchiMeDeS*. The development process is comprised of four steps:

- **Definition of the transformation at PIM level**. The first step is to conceptually define the mapping rule by identifying the cardinality of the rule, i.e., selecting the kind of transformation to create. For that aim, MeTAGeM obliges to select both the source metamodel and the target metamodels. Afterwards, it is necessary to specify the kind of transformation. In the selected example it will be '*one to many*'. Once it is done, it is the moment of indicating the source and target elements involved in the transformation. Figure 4-8 shows the result of this step.

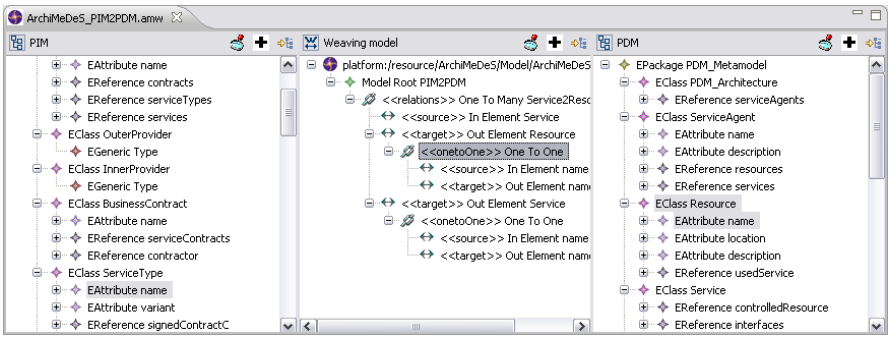

Figure 4-8.PIM model associated to a transformation rule.

- **Definition of a PSM transformation model**. The second step entails the modelling of the desired transformation using a specific hybrid language as previous step for the code generation in a concrete transformation language. Figure 4-9 shows the modelling of the selected transformation rule using MeTAGeM. Please note that this model is independent of the transformation language and that, if it were the case, it could be possible to select a transformation language different from ATL.

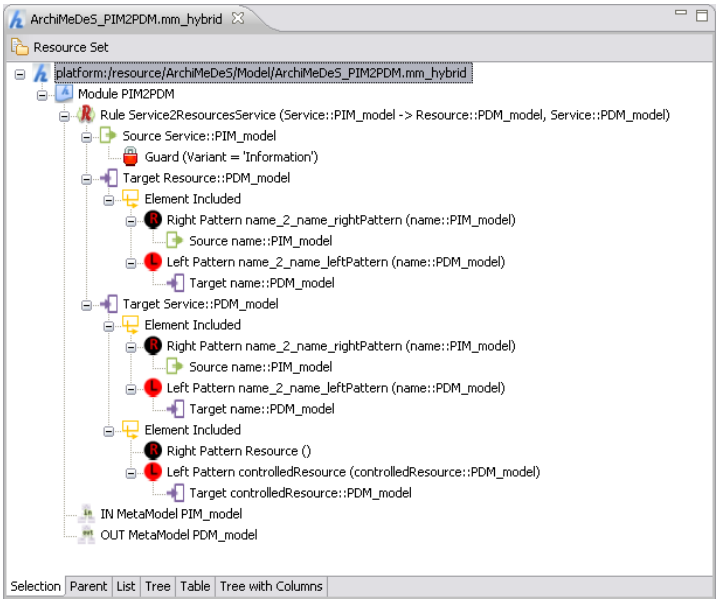

Figure 4-9.PSM model associated to a transformation rule.

- **Definition of the ATL model**. The third step is automatically generated from the previous PSM model. The MeTAGeM toolkit allows for the application of the concrete features of the ATL language to the previous hybrid model. The result is an *.ecore* model corresponding to the ATL model for the selected transformation rule.
- **Generation and edition of the ATL code**. The final step is also automatically generated. The MeTAGeM toolkit allows the generation of the ATL code associated to the selected transformation rule. Since there are particularities not solved with this automatic generation process, the tool also allows for the manual edition of the rule. Figure 4-10 shows the piece of ATL code that corresponds to the final coding associated to the selected transformation rule.

```
-- @atlcompiler atl2006
module PIM2PDM;
create PDM_model : PDM from PIM_model : PIM;

-- Comments -> This is a MatchedRule: Service2ResourcesService ->
rule Service2ResourcesService {
 from
  service_in : PIM!Service (service_in.Variant = 'Information')
 to
  resource_out : PDM!Resource (
   name <- service_in.name + '_resource'
  ),
  service_out : PDM!Service (
   name <- service_in.name +'_service',
   controlledResource <- resource_out
  )
 -- ActionBlock:
 do {}
}
```

Figure 4-10. ATL code associated to a transformation rule.

As it can be seen, the MeTAGeM framework provides with an intuitive development process for the (semi-)automatic generation of ATL code corresponding to the transformation rules defined as part of the *ArchiMeDeS* framework. Accordingly, this process has been applied to implement all the transformation rules defined to evolve from a view of the system architecture at PIM level to a concrete PSM architectural model depending on the service target platform of choice (see Tables 3-7, 3-8, 3-9 and 3-10 for a complete listing of these rules.

4.2.3.2 Implementation of PIM-to-PIM transformations in AMW

The superimposition of architectural styles in PIM models cannot be accomplished through an automatic transformation process since the description of the role played of each architectural element regarding a concrete architectural style is a design decision that uniquely depends on the architect's criterion. Therefore, PIM architectural models need to be annotated by hand with the features of the architectural styles and, as a result, a personalized transformation process must be accomplished based on a combination of the following:

- A **manual mapping** of the concepts defined as part of the architectural style vocabulary to services and service types existing in a PIM architectural model. This will be done by creating an *annotation model*

that links (*annotates*) the services with the desired concept of the architectural style.

- An **automatic generation** of the '*Enriched PIM architectural model*'. A set of transformation rules will be defined taking as input models the PIM architectural model, the architectural style model and the previous annotation model. The execution of these rules will result in a *service role* assignation as part of the final 'Enriched PIM architectural model'.

Figure 4-11 depicts the role of these annotation models in the scope of the weaving process defined. This figure extends the theoretical exposition of the weaving process outlined in section 3.2.3.3.4 taking into account the technologies used to implement the associated toolkit.

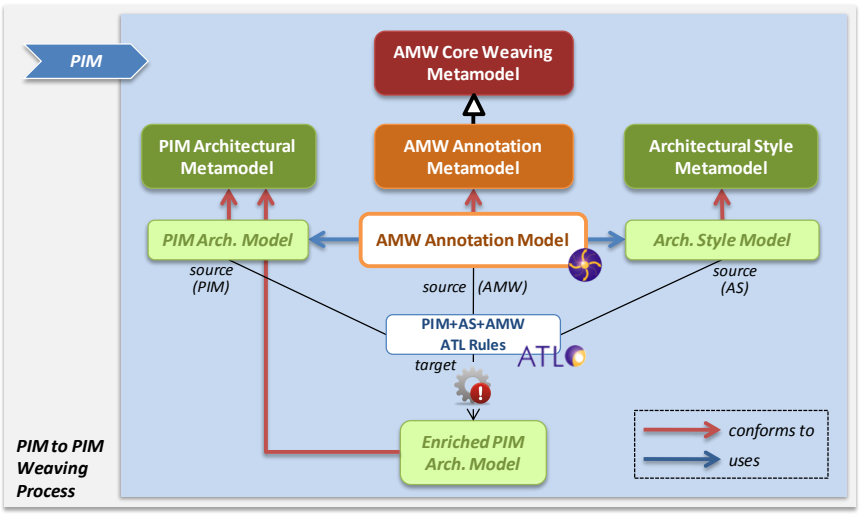

Figure 4-11. Implementation of architectural styles superimposition using weaving models and ATL transformations.

First step: manual annotation of PIM models

The first step of the superimposition process is to define the associations between the source PIM architectural model and the model with the concrete architectural style to be used. For that aim, AMW weaving models can be used as annotation models.

For every PIM architectural model and every Architectural style model ('*PIM Arch. Model*' and '*Arch. Style Model*' in Figure 4-11) a weaving model ('*AMW Annotation Model*') is defined conforming to an annotation metamodel. Such weaving model contains a set of annotations understood as links between the elements of both source models.

The creation and handling of weaving models relies on the *ATLAS Model Weaver* (AMW). The model weaver workbench provides a set of standard facilities for the management of weaving models and metamodels [53]. Moreover, it supports an extension mechanism based on a *Core Weaving Metamodel* [55] that contains a set of abstract classes to represent information about links between model elements (see the upper part of Figure 4-12).

Normally, the classes from the *Core Weaving Metamodel* are extended to define new weaving metamodels for specific contexts. This is due to the fact that all the classes defined in that metamodel are abstract classes and thus cannot be directly instantiated. The extension used to implement the support for architectural style superimposition within *ArchiMeDeS* is shown in Figure 4-12. Note that the Core Weaving Metamodel is depicted on the upper part of the figure whereas the extension is depicted at the bottom.

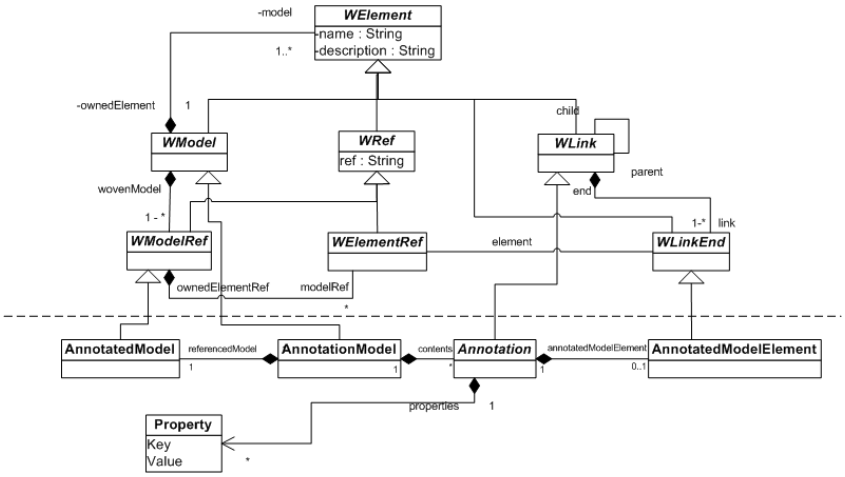

Figure 4-12. AMW Annotation Metamodel.

According to that metamodel, an annotation model includes a single-valued reference to the *AnnotatedModel* plus a set of annotation objects. Each annotation contains a single-valued reference to the model element plus a list of properties. The properties have an identification key and the corresponding value. The

AnnotatedModelElement class acts as the proxy for the linked/annotated elements. That is, each record is merely a set of key-value pairs.

Once it is possible to define personalized annotation metamodels (see Section 5.2.2.5 for some examples on the creation of these models), next step is using it to define the links between the PIM architectural model and the Architectural style model thus creating a personalized superimposition of the architectural style over the modelled architecture.

Second step: obtaining the Enriched PIM architectural model automatically

The second step is to combine the information of three models: PIM architectural model, Architectural style model, and Model weaving model. They represent the information needed to execute an automatic transformation that appends to the target PIM model the corresponding *service role* elements. Consequently, the target model ('*Enriched PIM Architectural Model*') is generated from both the source models and the weaving model. Figure 4-14 shows and excerpt of the *definition* of this ATL transformations rules. An example of the result of the execution of these rules is given in Section 5.2.2.5. Moreover, the described superimposition process allows obtaining different target models from the very same source PIM architectural model just by modifying the annotation/weaving model and/or the architectural style model.

The final model of the two-step process described above only takes into account the vocabulary of the architectural style modelled, thus creating an element organization (service instance annotation with roles) depending on the chosen architectural style. At this moment, the version of the toolkit presented in this dissertation only considers that element organization; however, a second version is in progress in which the architectural elements that currently are considered implicitly (see explanation in Section) will be explicitly created in this second step of architectural style superimposition. This extent will need a modification of the architectural style metamodel in order to specialize the constraints that are considered as part of the architectural style model (in that case it should be needed the addition of another element with the features of the potential element to be included as part of the resulting architectural model).

```
module PIMEnricher; -- Module Template
create OUT : ArchiMeDeS_PIM refining PIM : ArchiMeDeS_PIM, style : Architectural_Style,
amw : AMW;

-- Given an object from the woven model, it returns the Link referring to that object in
-- the weaving model
helper context OclAny def: getLink() : AMW!WLink =
        AMW!WLinkEnd.allInstances()->asSequence()->select(aux | aux.element.ref =
         self.__xmiID__) ->first().refImmediateComposite();

helper context OclAny def: hasLink() : Boolean = (not self.getLink().oclIsUndefined());
helper def: Connectors: Sequence(Architectural_Style!Connector) =
        Architectural_Style!Connector.allInstancesFrom('style');
helper def: Components: Sequence(Architectural_Style!Component) =
        Architectural_Style!Component.allInstancesFrom('style');
(...)
rule ServiceType2ServiceType {
  from
        st : ArchiMeDeS_PIM!ServiceType(st.hasLink().debug(st.name + ' has link:' ))
  to st_out : ArchiMeDeS_PIM!Service ( instances <- s ),
        s : ArchiMeDeS_PIM!Service (
            name <- st.name + '_instance',
            playedRole <- role ),
        role : ArchiMeDeS_PIM!ServiceRole ( name <- st.getLink().getRole())
}
```

Figure 4-13. Excerpt of ATL code allowing to obtain 'Enriched PIM architectural models'.

In addition, in order to offer a full support for architectural style modelling, superimposition and validation it would be necessary to provide with means to evaluate the resulting PIM model. To do so, it would be necessary to perform a validation process in which the modelled constraints could be used to derive some kind of output regarding the correctness of the modelled architecture. As it was pointed out in Section 3.2.3.3, this is an issue left as open research line to explore in the near future.

However, at this point of the dissertation, it is possible to assert that the moment in which implement a tool support for that validation process will be immediately after obtaining the '*Enriched PIM Model*'. This way, it could be possible to separate the annotation process from the validation one, thus, without affecting to that functionality. This checking process would be also encoded in model transformation, whose output will be any of the possibilities mentioned in Section 3.2.3.3.

4.3 Concluding Remarks

This Chapter has shown the development of a toolkit supporting the modelling tasks that associated to the *ArchiMeDeS* framework. Accordingly, this toolkit allows for the edition of architectural models at both PIM and PSM levels, the (semi-)automatic transformation of PIM models into PSM models and the obtaining of enriched PIM models by means of the superimposition of architectural style features on them. The resulting toolkit has been developed over the Eclipse platform and using some of the most currently used extensions to cover the modelling needs (EMF, GMF, AMW). Because of that, the toolkit can be easily extended with new modules and features derived from the updating of the *ArchiMeDeS* framework.

Although the toolkit created covers all the modelling needs defined in the *ArchiMeDeS* framework, it could be the subject of many improvements regarding interface refinements, definition of wizards for easing the modelling tasks or upgrading of the base functionality with new modules supporting architectural style validation or integrating the modelling of other concerns from the software development process.

Chapter 5:
Validation

Previous chapters described the inner features of the *ArchiMeDeS* framework, portraying its contents in the form of several DSLs and transformations among them, and the development of a supporting modelling toolkit. This Chapter is dedicated to validate both the proposal and the toolkit developed as part of the *ArchiMeDeS* framework.

As it was mentioned in the introduction of this dissertation, the investigation process follows an approach founded on the use of case studies [241]. Accordingly, the validation of the *ArchiMeDeS* framework (and the toolkit supporting its principles), is **accomplished by using it to specify the architecture of several case studies**. The content of each case study and its purpose is listed next:

- ***The GESiMED system***. This system represents a real-world case study based on the development of an information system for the management of digital medical images [96]. This case study is used to check the feasibility of using *ArchiMeDeS* to architect a full case study at either PIM or PSM levels of the proposal and using any of the target service-oriented implementation platforms (Web Services, REST Services and Grid Services). This case study will be also used to show an example of the model transformations defined within *ArchiMeDeS* and the corresponding modules of the modelling toolkit. The use of both the *ArchiMeDeS* framework and its associated toolkit for that case study is comprised in Section 5.2.
- The ***Pick & Roll*** strategy as a ***simulation of a basketball game setting***. The features of the GESiMED system do not allow the validation of the *ArchiMeDeS* framework for modelling service choreographies. In order to fill in this gap, the simulation of a basketball game setting (selecting the *Pick & Roll* strategy in particular [167]) is used. This way it is possible to show an example on the usage of the *ArchiMeDeS* capabilities to model this kind of service coordination schemes. The architectural modelling of this case study is explained in Section 5.3.
- ***A SMPP gateway***. This case study is based on the architecting of a gateway for massively sending SMS texts using the SMPP (*Short Message Peer-to-peer Protocol*) protocol. For that aim, the architecture of the SMPP gateway will be, first, modelled using the *ArchiMeDeS* PIM DSL for that aim, and, second, represented using the π-ADL architectural description language [180]. This is made with the purpose of showing that the presented DSL is defined at the right abstraction level as required for an adequate architectural description language. By doing so, it is

possible to check that the initial PIM description (made using the corresponding *ArchiMeDeS* DSL) is comparable to a standard architectural description using a formal ADL. The development of this case study with ArchiMeDeS and π-ADL is comprised in Section 5.4.

The *ArchiMeDeS* framework brings together three different concerns from Software Engineering: it uses a *Model-Driven approach* as strategy to follow for architectural specification; it establishes *Service-Orientation* as main semantic source for the elements specified in architectural models and it considers *Software Architecture* as the driving aspect for the development processes. The purpose of this chapter is to validate all these features and their synergies by means of the aforementioned case studies.

5.1 Using ArchiMeDeS for Architecting the GESiMED System

The main case study used to validate (and refine) *ArchiMeDeS* is the architecture specification of the *GESiMED* system. Initially conceived and implemented as a Web Information System [96] it was lately transformed into a service-oriented solution to cover new needs and broaden its usage to new scopes of research. The *ArchiMeDeS* framework was used to help in the development of the service-oriented version of *GESiMED*.

5.1.1 Background of the GESiMED system

Developed in conjunction with the GTEBIM (***G****rupo de* ***T****ecnología* ***E****lectrónica,* ***B****ioingeniería e* ***I****magen* ***M****édica*), a research group from the Rey Juan Carlos University, *GESiMED* allows the creation and maintenance of scientific studies for the research in neuroscience [3]. It was designed to give support to the investigation activities of neurologists, neuroradiologists and other professionals related to that area of the medicine. Additionally, the system is also used by researchers administering the *GESiMED* system as part of the LAIM (***L****aboratorio de* ***A****nálisis de* ***I****mágenes* ***M****édicas*) laboratory of the URJC, where it has been physically deployed.

The initial objective of the system was to offer to these neuroscientists a database for the storage of medical images accessible through the web, over which it would be possible to launch several types of queries and normalized processes. This would permit these researchers to perform the processing and analysis of the stored images and whose results would be also stored in the same database so that they could be consulted in ongoing and future studies. A general overview of this scenario can be seen in Figure 5-1.

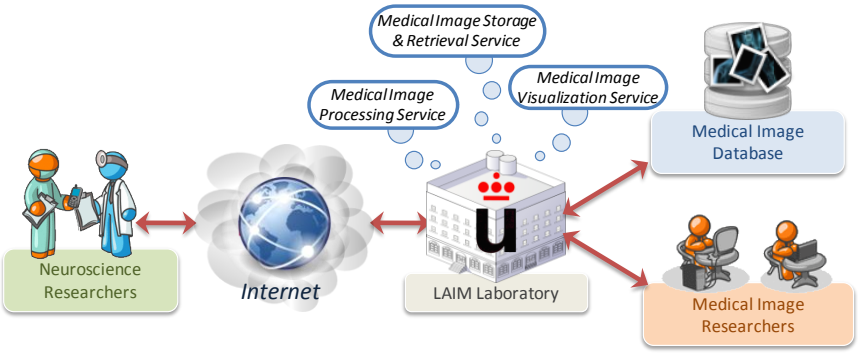

Figure 5-1. Overview of the GESiMED working environment.

To gain a complete understanding about the *GESiMED* system a proper description of the business processes it supports is given next. It has to be taken into account that, in that context, business processes are initially defined as a set of functionalities that are offered to the users of the system:

- **Storage and Retrieval of Digital Medical Images**. The main asset of *GESiMED* is the efficient management of digital medical images. The database of the system is built upon a huge set of medical images and the results of their processing using specific algorithms. The interest on these images does not rely on them as isolated image entities but on several other pieces of information associated to them. This information includes the research study which is being performed, the individuals that participate in that study, their grouping rationale and the concrete image retrieval tasks that are performed over them.
- **Processing and Result management of Digital Medical Images**. Through the *GESiMED* system, researchers in neuroscience may perform different types of image processing, such as analysis or segmentation. In order to accomplish the task required, the researcher may want to select a concrete set of images, launch the desired algorithm over the images and, finally, store the results of the image processing.
- **Medical Image Visualization**. Associated to the retrieval of the medical images, researchers may perform visualization processes over the stored data. These visualization processes may consist on tasks such as image reconstruction (transformation from 2D to 3D visualization) or multimodal image visualization that includes the obtaining of the functional view of the image set or the anatomical view of that same image set.

In addition to these business services provided by *GESiMED*, the features of the system require that other external services, also belonging to the business domain of the system, exist in order to give a complete support to the functionalities needed as part of the research activities of neuroscientists. These extra business services include the **payment of an established fee** from the external neuroscientists to the LAIM laboratory for the use of *GESiMED* and **obtaining the images from specialized equipment** operated by image experts (e.g. radiologists) installed in the clinical centres in which individuals participating in the studies are subject to the image acquisition tasks.

In order to provide a more formal view of the *GESiMED* capabilities, and according to the specification done in [45], it is possible to define all the high-level services offered by *GESiMED* using two models: a value model and a business process model. This way it is possible to obtain a visual outlook of the business processes supported by the system taking into account the values offered to each of the actors involved in the scope of the system (*value model*) and the business processes consequently associated to them in which the system participates (*business model*). The definition of a **domain model**, comprising the vocabulary and key concepts of the problem domain has been deliberately omitted since none of the terms needed to understand the scope of *GESiMED* are novel or difficult to understand.

The **value model** corresponding to the *GESiMED* system can be seen in Figure 5-2. The notation used is *e3value* [86]. In that model, the LAIM Laboratory is identified as an actor and both the medical centres and the neuroscientists as segments of the market, being the researchers in neuroscience considered as the final consumer of the business. Together with that information, the model shows, as value activities, the *services* offered by the LAIM Laboratory since it expects to get revenue from the performance of the business processes it offers. In accordance with the description given at the beginning of this subsection the value activities are: the "*Medical Image Storage and Retrieval Service*" (***MIS&RServ*** for short), the "*Medical Image Processing Service*" (***MIPServ*** for short) and the "*Medical Image Visualization Service*" (***MIVServ*** for short). The value objects exchanged between the different actors participating can be summed up in "*images*", "*image processing result*", the "*access and query of images*", the "*result of visualizing the images*" and the "*fee*".

Another aspect gathered in the value model is the identification of *dependency paths*, marked in the illustration as (a), (b), (c) and (d). These paths show the different working needs of neuroscientists and represent the starting points of the several dependency paths. These paths can be understood as follows:

first, neuroscience researchers need to obtain the images provided by the clinical centres (path (a)). Once images have been acquired, neuroscientists ask the LAIM laboratory for the processing or visualization of the images. So, when a process is required (path (b)), researchers provide the images and pay the corresponding fee, obtaining, in return, the result of the process selected. Images provided by the neuroscientist are delivered to the MIS&RServ service from where the MIPServ service recovers them in exchange of the processing result. Similarly to that path, there exists another one related to the image visualization (path (c)). Finally, neuroscientists may perform any kind of query over the database. The MIS&RServ is the service in charge of this responsibility which, in return, receives the payment of the service performed (path (d)).

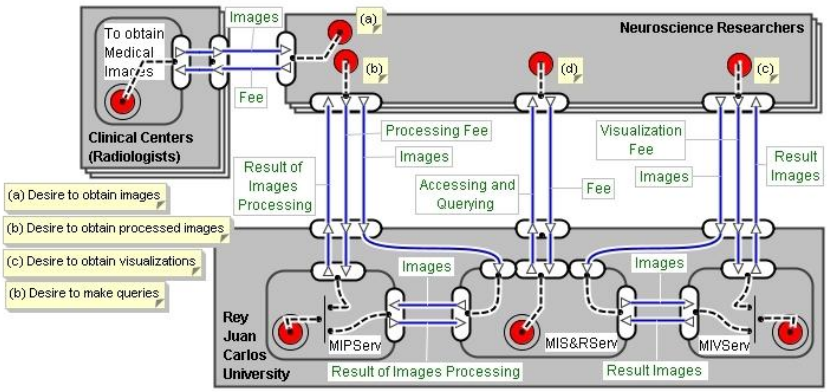

Figure 5-2. GESiMED value model.

Figure 5-3 shows, in turn, the business process model associated to the *GESiMED* system. This model shows a set of activities that may perform any of the neuroscientists as part of their research work. In that sense, researchers must obtain, prior to anything, the images either asking for its acquisition to the medical centres or querying the system's database. Once the images are retrieved, researchers may ask for its processing to several centres, such as the LAIM laboratory.

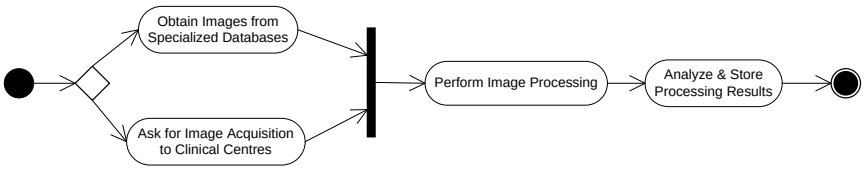

Figure 5-3. GESiMED business process model.

5.1.2 GESiMED PIM Architecture

The knowledge about the domain of *GESiMED* and the business processes in which the system is framed serve as inputs to specify the PIM architecture of the *GESiMED* system. The creation of this model and its conformance validation has been done using the *ArchiMeDeS* toolkit. Figure 5-4 shows a snapshot of the toolkit in the moment of creating the PIM architectural model of the *GESiMED* system.

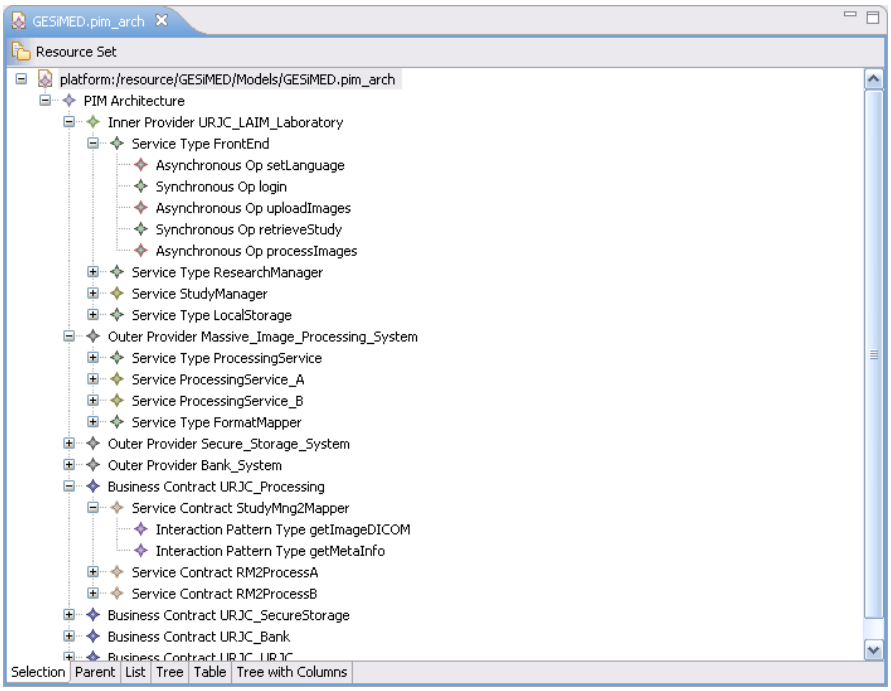

Figure 5-4. GESiMED PIM architectural modelling with the ArchiMeDeS toolkit (partial model).

The whole PIM model in UML can be seen in Figure 5-5. For the sake of clarity some elements has been omitted (service providers, business contracts and some interaction patterns in service contracts). To better exemplify the use of *ArchiMeDeS* for modelling software architectures, the explanation on the modelling of the case study will discuss each type of element separately.

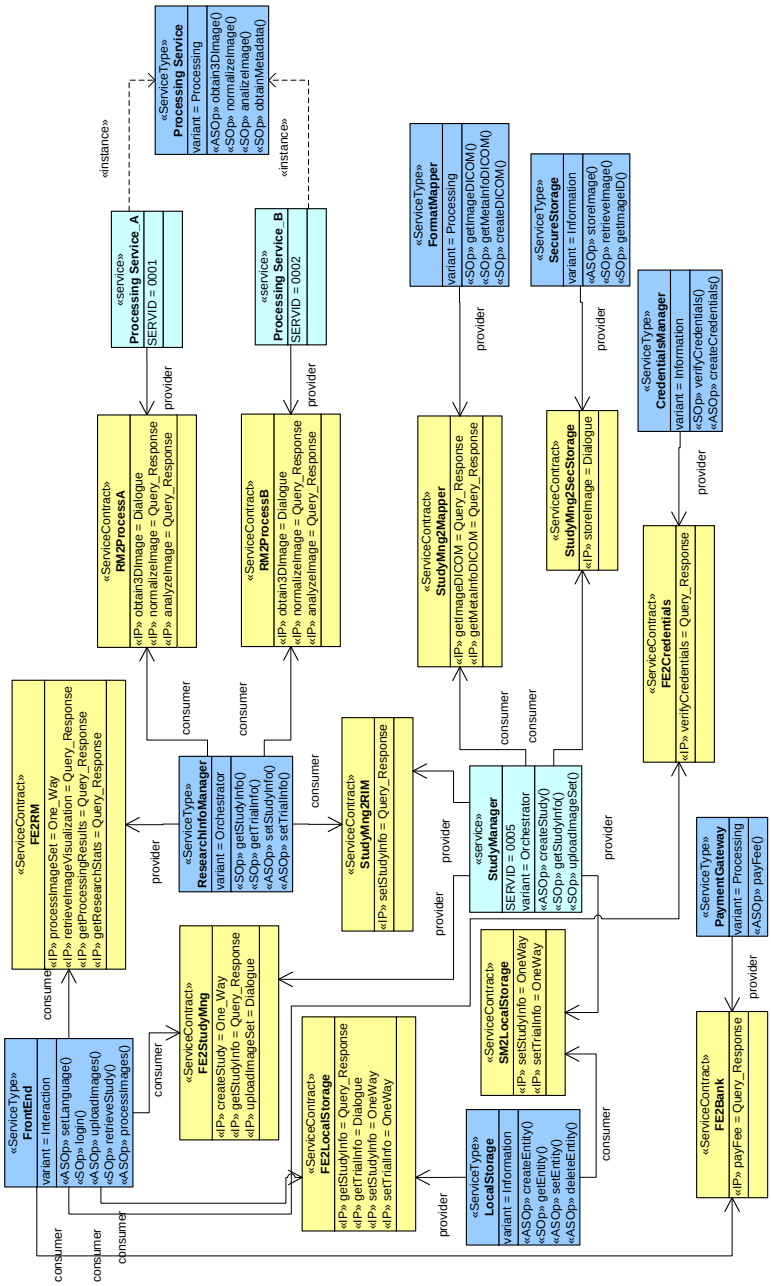

Figure 5-5. GESiMED PIM architectural model.

5.1.2.1 Modelling Service Providers

To illustrate how these organization elements are positioned in reality, Figure 5-6 depicts the main business entities as service providers that can be identified in the context of the *GESiMED* system. In it, it is possible to recognize, as outer service providers, the *Bank System* provider, the *Secure Storage* provider and the *Massive Image Processing* provider. Because of the fact that the *GESiMED* system acts as the unique business operator managing the system under development, the only inner provider will be that representing the institution (*URJC LAIM Laboratory* provider) in charge of the services implemented and deployed.

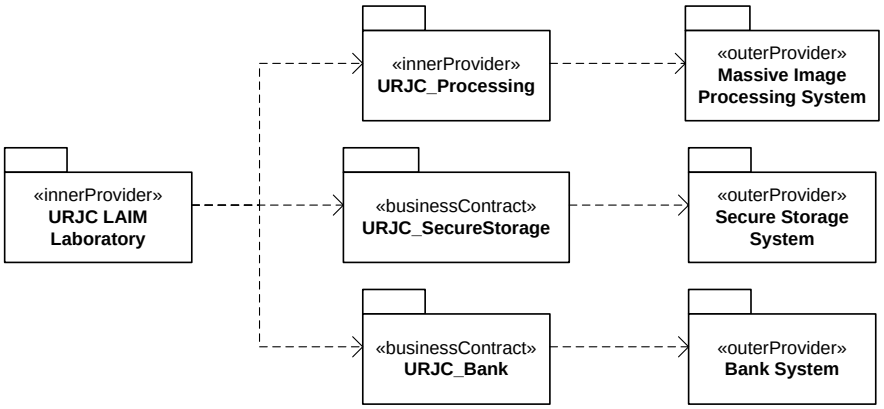

Figure 5-6. Service providers involved in the GESiMED case study.

5.1.2.2 Modelling Services and Service properties

Once the business entities of *GESiMED* have been identified, it is possible to explode the contents of each provider by identifying their constituent services. This way, it is possible to model not only the service types that may be identifiable in the context of *GESiMED*, but also, the concrete service instances that must be present in order to correctly achieve the behaviour expected for the system. To give an example, Figure 5-7 depicts some services that belong to the *Massive Image Processing* outer provider. This figure shows two instances of the services that have the ability to process the digital medical images and provide operations that allow executing several algorithms on them, either individually or over a set of them to extract a concrete result. Note how the model reflects a design decision of including two identified instances that will compulsory take part in the enactment of the functionalities expected from the system.

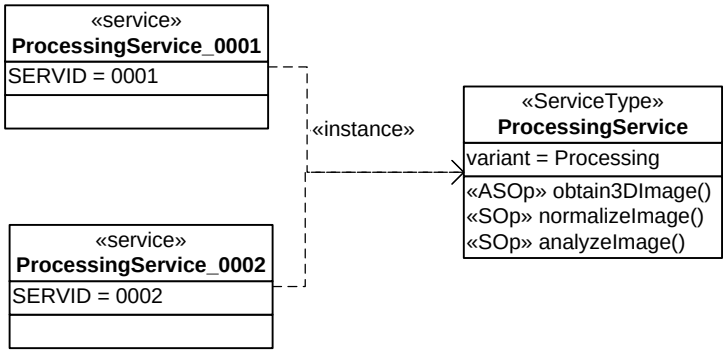

Figure 5-7. Modelling service types, services and operations.

5.1.2.3 Modelling Service Contracts

Figure 5-8 shows a contract established between two services of the *GESiMED* system. In that case, it depicts the relationship established among a general service type named *FrontEnd* and another one in charge of administering the information related to information of the clinical studies managed internally to the *GESiMED* system, named *StudyManager*. The model does not reflect, as a consequence of a design decision, whether they are sole instances or, on the contrary, various elements in the form of different service instances. The contract established shows the interacting rules that must be observed when a potential *FrontEnd* service instance wants to take advantage of the functionalities provided by a *StudyManager* service. Note how the definition of an operation as synchronous (e.g. *uploadImageSet*) does not entail that the interaction pattern must be of the *Query-Response* type as it would be thinkable in a first moment. According to the information gathered from the business scope of the system, to store the images related to a medical trial it is necessary to define a set of dependent variables that obliges to carry out a complex interaction process (not specified here for the sake of simplicity).

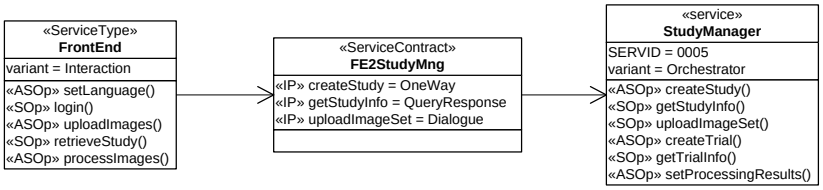

Figure 5-8. *Part of the GESiMED model architecture showing a service contract.*

5.1.2.4 Modelling Service Composition: Orchestration

In the context of *GESiMED*, the processes associated to its functionality only require the specification of service compositions based on an orchestration scheme (choreographies will be exemplified within the modelling of a different proof of concept later on this chapter). To illustrate this kind of composition, it is possible to observe how the *GESiMED* system deals with the storage of the information that is initially uploaded by neuroscience researchers. These researchers normally obtain medical digital images in files coded in the DICOM format [162] which has to be processed to extract both the graphical information taken from a patient (the image itself) and a compendium of data associated to the acquired image (using a *FormatMapper* service). The image storage workflow follows with the insertion of all this data in a secure database (external), via a *SecureStorage* service. Apart from this, it is necessary to update the information related to the clinical trial or study to which the image belongs (using the *LocalStorage* service). In that case, this information is stored internally but keeping a reference to the images included in that clinical trial. Due to this differentiation in the storage of the information, several services must participate in the task of storing the medical images sent by the researchers. To complete this task a special service (*StudyManager* acting as orchestrator) is in charge of executing the workflow defined previously. A simplified overview of the model for this scenario can be seen in Figure 5-9.

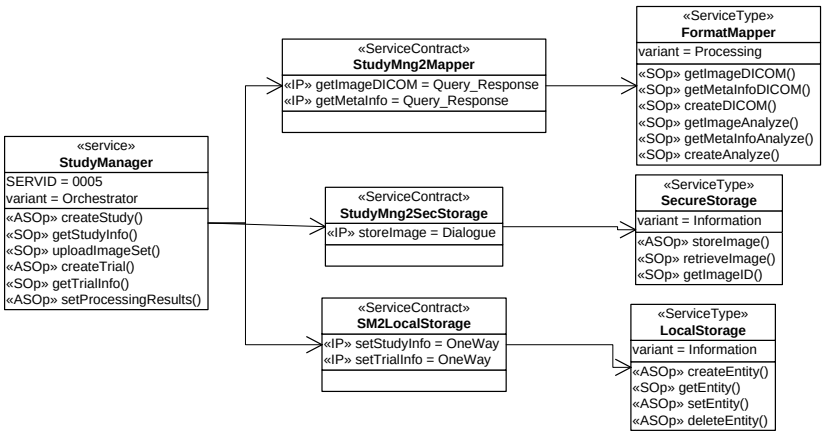

Figure 5-9. Service Composition Modelling in GESiMED: Orchestration.

5.1.2.5 Modelling Architectural Style Superimposition

This subsection aims to provide an illustrative example on how the *ArchiMeDeS* framework and its associated toolkit can be used to superimpose architectural style information over PIM architectural models by means of *service roles*. To do so, a partial model of the *GESiMED* architecture will be used as example for the superimposition of two different architectural styles. In particular, the *Pipe & Filter* and the *Layered* architectural styles have been selected to be superimposed, using the module created for the *ArchiMeDeS* toolkit, over the *GESiMED* PIM architectural model.

The process for architectural style superimposition starts with the definition of the model of the architectural styles to be used. Figure 5-10 shows both Pipe&Filter and Layered style models. It is important to note here that the support for modelling the restrictions associated to the styles have been left for future work. As it can be seen in the Figure, models used define these restrictions in natural language. Accordingly, the only elements that will be used in the superimposition process and subsequent model transformation will be those defining the vocabulary of the architectural style. As it was explained previously, the weaving process described only aims at obtaining an enriched PIM architectural model in which the services modelled are annotated with service roles according to the architectural style selected.

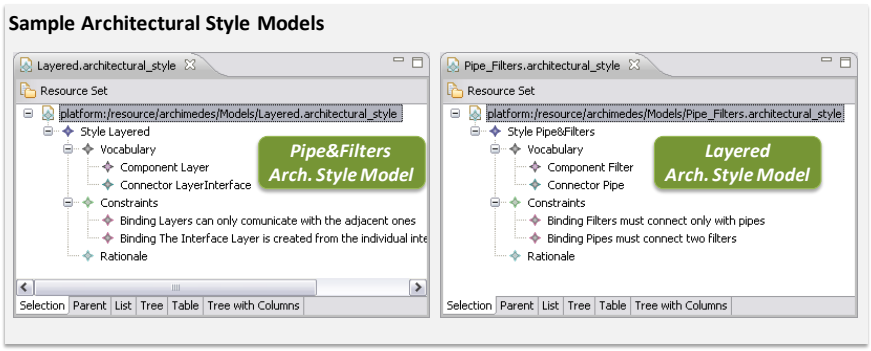

Figure 5-10. Sample architectural style models.

Next step is the creation of the weaving model, in which services and service types in the PIM architectural model are *linked* with the desired element described by the architectural style model of choice. To illustrate this, Figure 5-11 shows the *weaving model* created for the superimposition of the *Pipe & Filters* style over the GESiMED PIM architectural model.

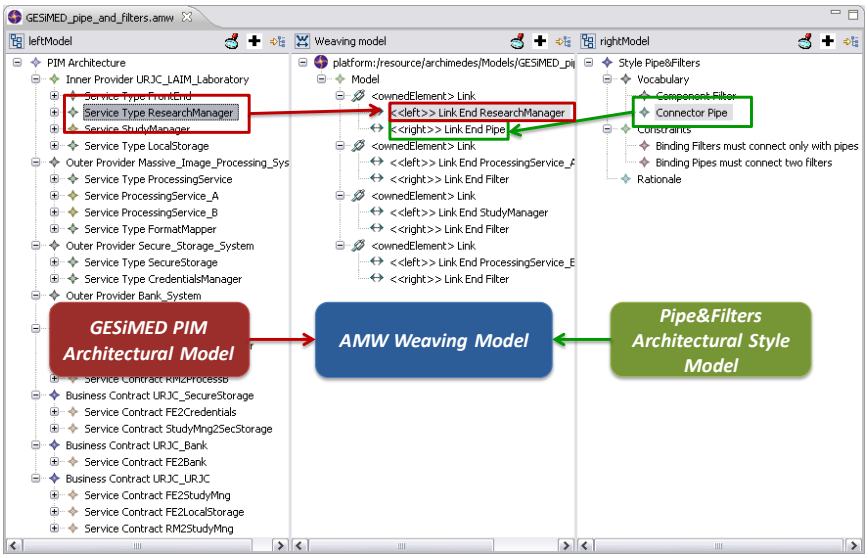

Figure 5-11. Definition of the AMW annotation model.

For the sake of clarity, model shown in Figure 5-11 focuses exclusively on the annotation of the services related to the activity of processing a set of digital medical images (via any of the instances of the *ProcessingService* service) and the subsequent storage of the results in the local storage system (via the *LocalStorage* service). Each of these services will be identified as *filters* performing some kind of computational functionality. In order to communicate them and exchange the information they need to process, it is necessary to define which element will act as *pipe*. In the case of the selected scenario, the service in charge of this task will be the *ResearhManager* service. For that reason, this service will be annotated as a *pipe*.

Once this model has been created, the next step is to automatically process the three models previously mentioned (the *GESiMED PIM architectural model*, the *Pipe & Filters architectural style model* and the *AMW annotation model*) to obtain the '*Enriched PIM Architectural Model*' for the *GESiMED* system annotated with service roles and according to the style chosen. To do so, it is necessary to *execute* the ATL transformations allowing to perform this process automatically (depicted in Section 4.2.3.2).

Depending on the input models the result obviously varies. To illustrate this issue, Figure 5-12 shows the resulting models after superimposing both the Pipe & Filter and Layered architectural styles over the *GESiMED* PIM architectural model. To apply the layered style, the design decision taken has been

to define two different layers, one for the *ResearchManager* service and other for the processing and storage Services. The assignment of values to the properties related to each role (layer numbering and positioning for example) requires a subsequent refinement of the models which is not possible to fulfil automatically.

However, it is noteworthy to observe how the information from the input PIM architectural model has not been altered in any way, thus preserving the original architectural configuration. This resulting architectural model allows for additional reasoning on the behaviour the system may perform, according to the specific roles given but independently from the inner configuration of the services comprising the architecture.

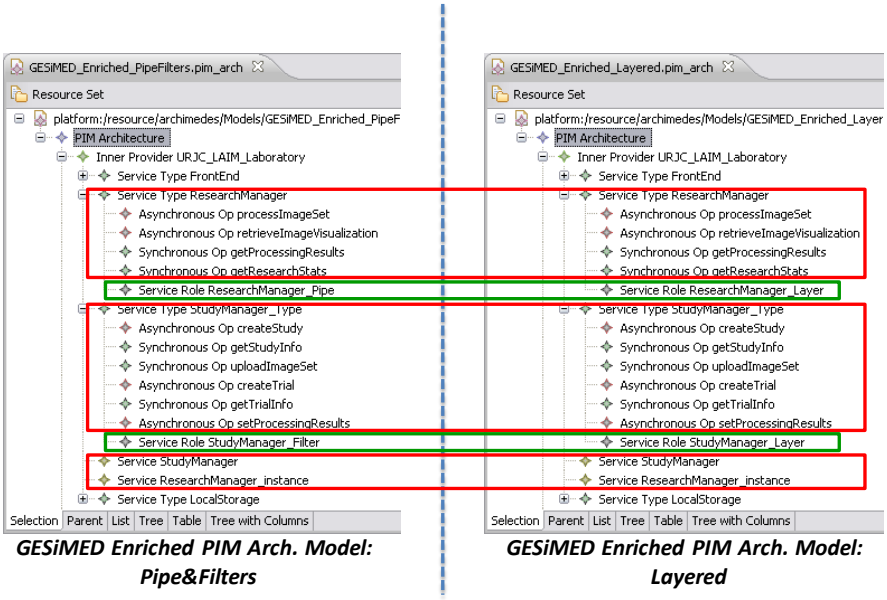

Figure 5-12. Differences between resulting 'Enriched' models.

Figure 5-13 shows a graphical comparison between a set of elements from the original 'PIM architectural model' and the elements automatically obtained after the execution of the ATL transformations. As it can be deduced from that figure, the set of operations assigned to either the service roles or the service instances are the same that the original *service type* or *service* element had. In order to consider coherent the architectural configuration, a later processing of this model deleting or including the necessary operations would be required.

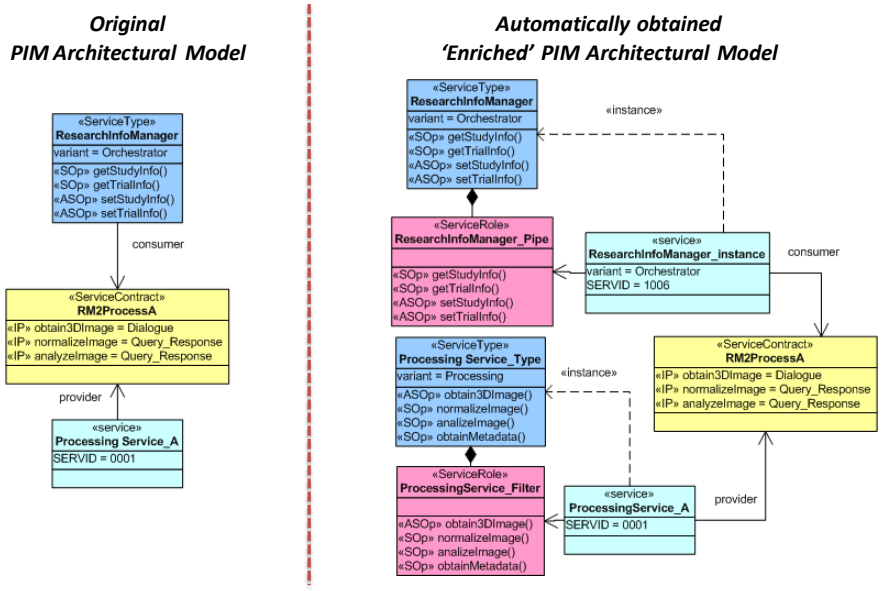

Figure 5-13. Sample transformation of an original PIM architectural model into an enriched PIM architectural model.

5.1.3 GESiMED PSM Architecture

The PSM level of the architecture will depend on the target platform chosen to implement the system under development. However, for the PSM modelling of the *GESiMED* system using the *ArchiMeDeS* framework it is highly recommended to model, first, the PDM model. This model will comprise all the commonalities that the *GESiMED* system will have independently of the service-oriented platform finally chosen to deploy the system. Afterwards, this model will be available to be refined with the particularities coming from the target platform of choice, thus creating the corresponding TDM models for Web Services, Grid Services or REST services.

At this point it is important to remark that the development of *GESiMED* includes the creation of several other models as indicated by the SOD-M method [49] and the MIDAS framework [35]. Due to space limitations, the concrete models have been left out of the article. The reader might refer to the bibliography to check how the hypertext of the system [48], the concrete behaviour [45] and the storage [227] at either PIM or PSM level should be modelled.

5.1.3.1 *GESiMED* PDM Architecture

Figure 5-14 shows the model corresponding to the PDM level of the *GESiMED* architecture. In that figure, it is possible to see how services have been transformed depending on the value assigned to the *variant* property in the PIM model. For example, the *LocalStorage* Service has been transformed into a pair Service-Resource. In that case, the objective is to reflect in the architectural model that there will be a unique service granting the access to a persistent resource, either a database or a different storage capability. The decision on what kind of resource or the inner structure of the data stored has not been included as part of the architectural model. The reason behind this is the separation of concerns fostered by the methodological framework in which *ArchiMeDeS* is framed, where a specific model gathers all the storage issues. The same happens with resources and services obtained from *interaction* services. This Thesis encourages the necessity of creating explicit interface models for that kind of resources. Regarding orchestrations, the transformation from the PIM model ends up in creating a set of services related to each other depending on the service contracts established at that conceptual level.

A first version of the architectural model at this level can be obtained by executing the implemented PIM-to-PDM ATL transformations. However, it is necessary to refine that initial model in order to consider the PDM model complete and include features and architectural decisions that may depend on either the architect's decision or the information gathered in other models of the model architecture. For example, service types that were identified at PIM level could become a number of concrete service instances depending on the definition of the concrete activities that need to be performed to implement a specific behaviour.

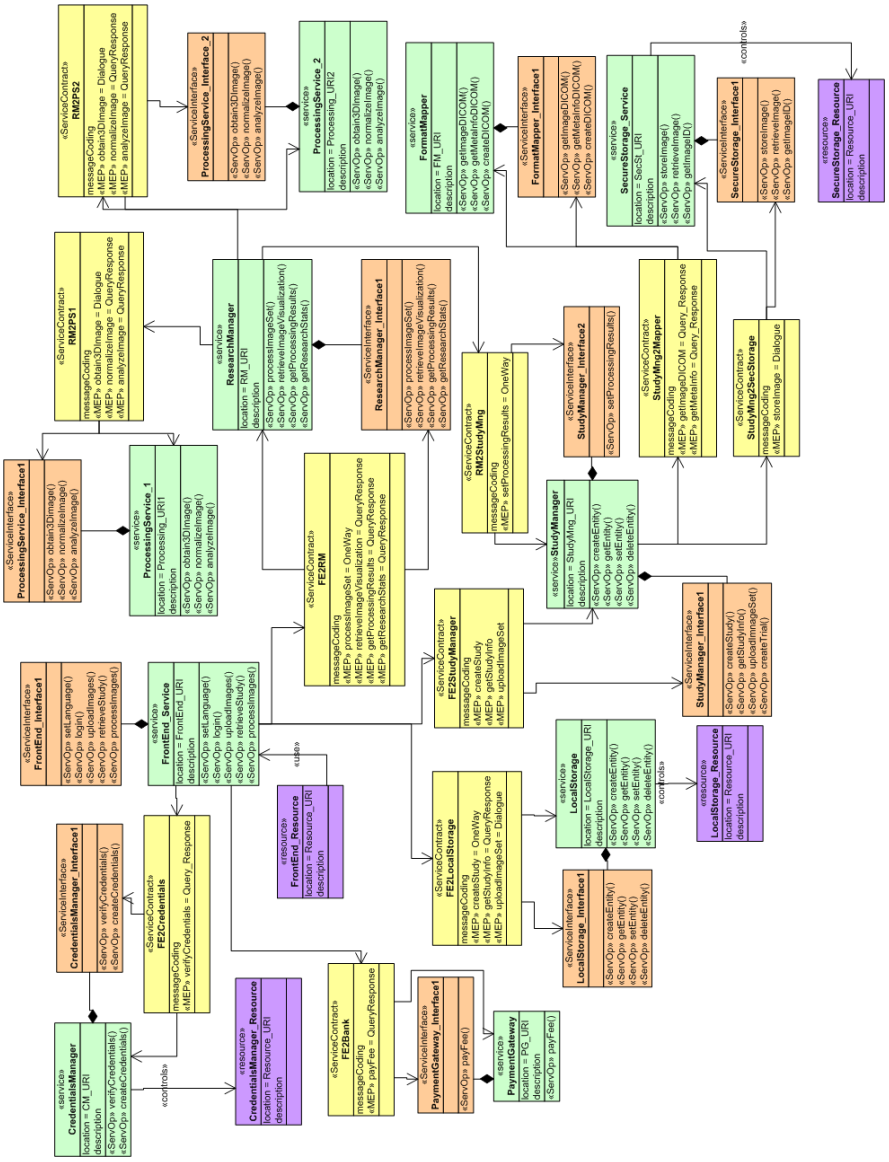

Figure 5-14. PDM modelling of the GESiMED system.

The relevance of creating a PDM model separately from the implementation technology can be clearly illustrated by having a closer look at the *Processing* service. This service is one of the most important ones within the scope in which the *GESiMED* system is being used. From the point of view of the implementation technology, this service may be deployed as a simple Web service in charge of image normalization and segmentation or it may take advantage of the distributed computing capabilities that a Grid Computing environment may provide.

The migration to either Web Services or Grid Services will only require the modification of the TDM models without affecting to neither the PIM architectural model nor the PDM, as well as its transformation to the concrete code. Other interesting service is the *bankService* one, since it allows us to work in aspects related with non-functional aspects such as security or policy protocols although these aspects are not the subject of this Thesis.

The contracts that were specified in PIM level architectural model have been seamlessly transformed into service contracts among the different services identified at the PDM. This is due to the fact that the way in which the communication among two elements is implemented will depend on the technology selected, and on the type of contract established among them. In most of the cases, the communication control will simply be embedded in the implementation of the elements connected (a Web Service invocation from the Web interface will be coded within the code of the server Web page). This element will implement the message exchange pattern and control the communication as the involved services shift turns.

Finally, as services usually are the only way to access a resource, the specification of the concrete features of that resource will be modelled outside the architecture model. The database, for instance, will have its own Schema Model (see [228] for more details).

Although it is possible to set different service interfaces depending on the context in which the *GESiMED* services are used. For the case of the implementation done, the service interfaces created will comprise all the operations that a service may perform. For subsequent refinements, or in case of taking advantage of the potential service roles identified at PIM level due to architectural style restrictions, this PDM model allows for the inclusion of new interfaces without affecting the nature of the services themselves. This issue provides evidence on the relevance of using the SOC paradigm for modelling software architecture as it was pointed out previously.

It is important to remark here that the elements that are shown in Figure 5-14 contain more information than the one modelled at PIM level. As it was mentioned in Section 4.3.3, the architectural model that is obtained from executing the PIM-to-PDM transformations is far from being complete and needs for the inclusion of data collected in other concerns modelled as part of a system development process (for example, from behavioural models). In addition, properties like location or description will have to be assigned by hand by the architect since they are attributes that will depend greatly on implementation and deployment decisions.

5.1.3.2 *GESiMED* TDM Architecture: Web Services

In order to get the TDM model of the *GESiMED* system architecture, it is possible to follow two different approaches: on the one hand it is possible to start from the PDM model of the architecture and apply or modify those services that will be implemented by means of the Web Service technologies and standards; or, on the other hand, it is possible to execute the ATL model transformations defined to step from the PIM modelling of the architecture directly into the Web Service model at TDM level.

The TDM architectural model for *GESiMED* using Web Service technologies and standards resembles significantly to the PDM model with some minor changes. Figure 5-15, for example, shows the modelling of the FrontEnd service and the resource it allows accessing it. In addition, it shows the service agent in charge of its execution. From that information it is possible to derive the necessity of having a Web page resource allowing to access a Web Service that performs certain functionality. Note that this is the only information that can be extracted from the architectural model, a skeleton of the system source code or, more properly, a *protoarchitecture* of the system [45]. In order to implement either the Web page or the Web Service itself, additional information is needed. In the case of the Web page, for instance, the modelling of the interface concern is encouraged. At the moment of writing the present dissertation, code generation must be done manually.

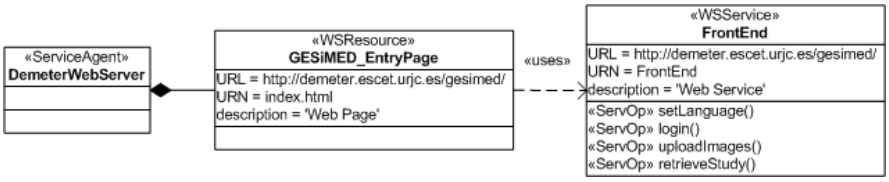

Figure 5-15. TDM modelling of GESiMED: Web Services, Resources and Service Agents.

Another example of the architectural modelling of the *GESiMED* system can be found in Figure 5-16. In it, a Web service contract is depicted showing the information gathered according to the DSL described in *ArchiMeDeS* for Web Service technologies. It is remarkable, for example, how the interaction patterns that were defined at PDM have been transformed into concrete values of the WSDL 2.0 standard language. As it was noted previously, from this partial model it could be possible to generate a skeleton of the interface offered by the *ResearchManager* Service, resulting in a partially complete .wsdl document. The rest of the missing information in that document should be filled with data from other modelling concerns (e.g. storage for *message datatypes*)

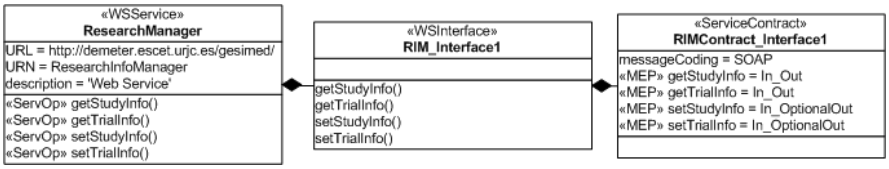

Figure 5-16. TDM modelling of GESiMED: Web Service contracts and interfaces.

5.1.3.3 *GESiMED* TDM Architecture: Grid Services

As it happened with the Web Service TDM model, the Grid service architectural counterpart can be also obtained mainly from the PDM model. The part of the *GESiMED* that benefits from the capabilities offered by the use of a Grid Computing platform is centred in the *ProcessingService* and the related resource needed to compute tasks related to image processing using specialized algorithms. Figure 5-17 shows a partial model of the *GESiMED* architecture centred in the modelling of the *ProcessingService* using the TDM DSL for Grid Services defined as part of the *ArchiMeDeS* proposal.

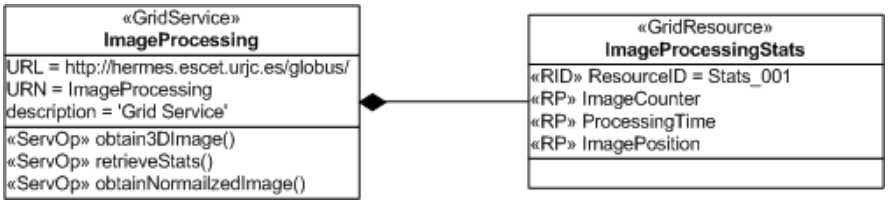

Figure 5-17. TDM modelling of GESiMED: Grid Service and Grid Resource.

Although the partial model shown above can be considered as very simple, it contains enough information to populate the attributes and variables needed to describe both the Grid Service (associated *.wsdl* file) and the Grid Resource

needed to be able to perform, for example, the generation of a 3D image from a set of individual 2D images. In order to achieve that behaviour it is mandatory that the Grid Service maintain the state between invocations (since the images are retrieved one by one by the service). This is the main objective of modelling the Grid Resource that will store the persistent information between invocations. The information modelled for that aim conforms to the WSRF specification and, as it happened with Web Services, following and adequate transformation process it could be possible to generate all the files needed to build and deploy a Grid Service over a Grid Computing platform such as the *Globus Toolkit 4* (that was the one used in this case).

5.1.3.4 *GESiMED* TDM Architecture: REST Services

To illustrate the impact of using REST services on the system architecture, this section will provide with a partial modelling of the *GESiMED* architecture using the stereotypes defined for the DSL for REST services at the TDM abstraction level. In particular, the focus will be put on the *LocalStorage* service that allows the management of entities within the scope of the *GESiMED* system. In that context, entities refer to studies, trials, individuals (subjects from which the digital images are obtained), groups of individuals kinds, etc.

As it can be seen in Figure 5-18, the model is very similar to that of the Web Service DSL. The main reason for that correspondence can be found in the fact that REST services are currently considered as a particular extension to the Web Service technology. In fact, the support for REST service description is, at the moment of writing, part of the WSDL 2.0 standard language for services [235].

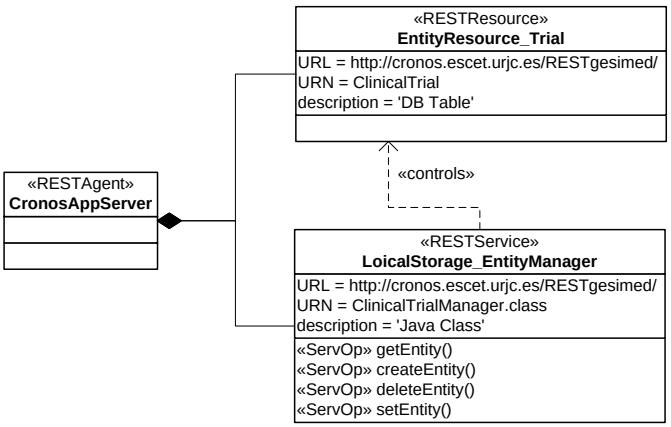

Figure 5-18. TDM modelling of GESiMED: REST Service, Agent and Resource.

It is important to highlight the fact that the *LocalStorage_EntityManager* service conforms to the REST principles, by offering a standard set of operations available to access a concrete entity (in that case, a clinical trial): *getEntity* (GET operation), *setEntity* (POST), *createEntity* (PUT) and *deleteEntity* (DELETE).

5.2 Using ArchiMeDeS for Architecting a Basketball Game Setting

As it has been shown in previous sections, at the architectural level, one of the distinguishable aspects of the use of the Service-oriented paradigm in this context is the way architectural elements are composed. When dealing with composition and services the key aspects relies in the ability to represent both orchestrations and choreographies as they are the way service compositions are accomplished. In order to verify that the *ArchiMeDeS* is suitable to represent different service composition strategies at PIM, the *GESiMED* case study served to illustrate how service orchestration may be modelled at PIM level. However, as it was pointed out, that case study did not resemble any possible choreography. To overcome that insufficiency this section describes how the proposed framework is applied over a partial case study as proof of concept for modelling service choreographies. To do so, the example selected is based on a sport metaphor in which a scenario is specially created to represent it using a service-oriented approach. In that sense, the description of tactical systems in team sports is considered as valid analogy.

With that aim, the description of existing tactical models for basketball as a metaphor is used to develop service-oriented architectures in which coordination is required. Sport analogy is appropriate in this case since this environment provides independent entities (players represented by services) that communicate with each other to obtain a common objective [194]. Thus, the problem of coordinating services is reduced to service choreography modelling, allowing to focus on checking the feasibility of the approach.

To check both the viability of the modelling approach to define service choreographies and the development method used, here it is used a concrete tactical basketball play as coordination strategy to be modelled: *pick and roll (P&R)* [167] during the course of a basketball game. First of all a series of several factors, needed to understand this basketball offensive action, are defined.

5.2.1 Background on the simulated Basketball Game Setting

In a *two-versus-two* players game setting (2vs2) using P&R is one of the different attack options included in an offensive system. P&R involves four players (two attackers and their defenders), where the attackers interact to free each of their defenders using a legal obstruction movement. Roles derived from this tactical situation are four: Playmaker (*attacker with ball*, AB), Team mate (*attacker without ball*, AWB), Defender (*ball player defender*, DB) and Assistant (*player without ball defender*, DWB). Each of these roles has associated a series of specific actions [194]. Table 5.1 shows some of these actions.

Table 5.1. Actions associated to each player in Pick and Roll.

Tactical offensive actions		***Tactical defensive actions***	
PLAYMAKER	**BLOCKER**	**DEFENDER**	**ASSISTANT**
Stop & Shot	Block	Deny Screen	Soft Flash
Drive	Roll	Over Screen	Switch
Clear	Pop	Below Screen	Hard Flash
Rolling Pass	Repick	2vs1	2vs1

The application of P&R during an attack phase is not the result of the direct intervention of one of the participants acting as main coordinator (for example: a coach giving instructions to be carried out by players). On the contrary, the emerging behaviour comes out from the relation of the different participants involved. Each player rarely performs the same action in different plays; their actions are determined by the conducts, decisions or movements of the rest of the team members. For example, in the same play, playmaker may choose: to pass the ball to his team mate if is clearly unmarked after blocking, shoot if his defender has been perfectly blocked, etc. [194].

5.2.2 Modelling Service Composition: Choreographies

In order to be able to model this situation from an architectural point of view and using services, the modelling of choreographies is encouraged. Figure 5-19 shows the PIM model that correspond to a P&R situation using the DSL defined within the *ArchiMeDeS* proposal.

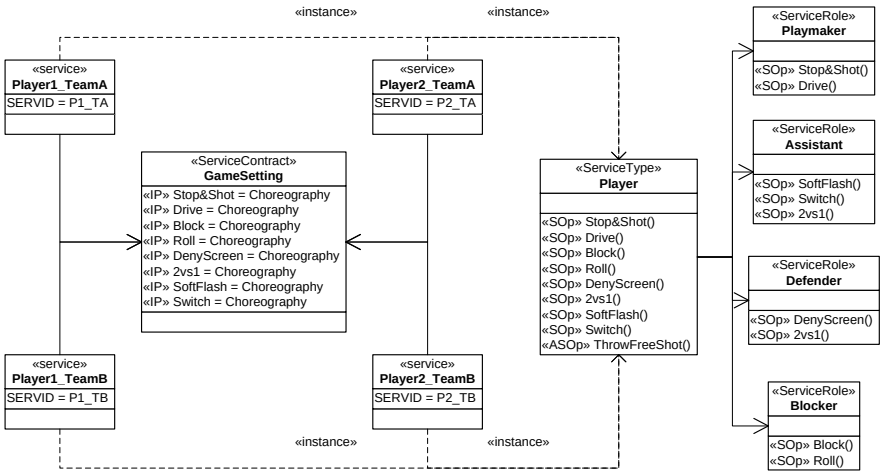

Figure 5-19. PIM composition example: choreography.

In that model it is possible to see how choreographies are modelled based on the interaction of four basket players. These players are represented via service instances having the same Service Type (‘player’) but being able to act following different roles (blocker, playmaker, assistant, defender). That is, all services can perform a certain set of operations, but they are only able to execute some of them according to the assigned role in a certain moment. This role identification leads to different operations available for each participant in the choreography. The decision of which roles are playing each service should be defined at run-time.

On modelling service choreographies at run-time

It is important to remark the fact that the defined service model only represents static information and no reference to how the system architecture evolves has been included. Although this issue has been left for future works, when a service participates in a choreography it acts according to a certain role. To show a representation of role assignment in a concrete run-time configuration, UML2 can be used. In UML2 roles are represented as connectable elements [175]. At this point, it is worth to note how the service definition (service type instances representing each player) is separated from the roles played in a concrete game setting. As a result, choreographies were defined as *the message exchange produced as a result of the interaction among the roles played by each service instance*. These messages come from the invocation of the operations defined on each role and the actions they perform as a consequence.

In UML2 relations between each connectable element are defined with *collaborations*, represented through collaboration diagrams. Following the sport metaphor, each of these collaboration diagrams represents a concrete tactical situation in which each player has assigned a certain role (see Figure 5-20). The collection of all the collaboration diagrams is considered as a complete tactical system. Each tactical situation has associated a set of variants for the game evolution. Those variants are represented with sequence diagrams including each participating role together with the operation invocations. These two independent descriptions are related as explained on the UML2 specification, i.e., through dependencies defined on a *CollaborationUse* element.

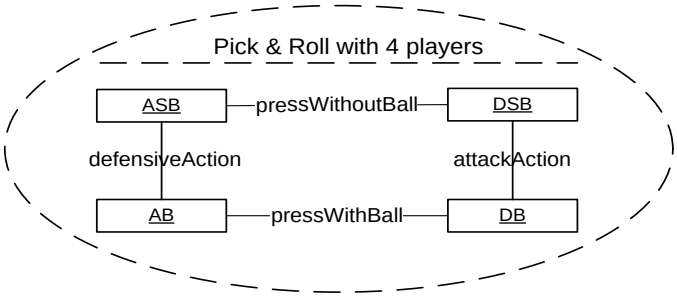

Figure 5-20. Roles modelled as collaboration in a Pick&Roll setting.

The number of possible tactical combinations for the same play is quite high, so represent all of them using low-level languages is far from being trivial or simple. By using the sport metaphor it is possible to detect how model-driven development may help to develop software solutions in which the implementation of coordination strategies based on choreographies is needed.

5.3 Using ArchiMeDeS for Architecting a SMPP Gateway

This subsection introduces another case study based on the architecting of a *SMPP gateway* for massively sending SMS texts. This example will be developed in two parts: first, the architecture of this case study will be modelled using the *ArchiMeDeS* framework. This modelling task will be done at PIM level. Later, using this PIM description as a template, the whole SMPP gateway architecture will be represented using the π-ADL architectural description language.

By doing so, it is possible to check that the initial PIM description (modelled using the corresponding *ArchiMeDeS* DSL) is comparable to a standard architectural description using a formal ADL. This is made with the purpose to

show that the presented DSL is defined at the right abstraction level as required for an adequate architectural description language.

Note that this does not intend to be a formalization of the whole DSL. Obviously, by translating every element in the case study, this section essentially provides the template for the translation for every element in the PIM metamodel. However, to provide a full formalization was never included as an objective of the Doctoral Work. The only purpose of this section is to check the abstraction level of the PIM DSL, and to be able to assert that it is comparable to that of any other ADL.

5.3.1 Background on the SMPP gateway system

The case study selected has as main objective the emulation of the functionality of a SMPP (*Short Message Peer-to-peer Protocol*, [211]) gateway system by means of services. SMPP is a telecommunications industry protocol for exchanging SMS messages between SMS peer entities known as *Short Message Service Centres*. Through a service interface (Web Service, REST compliant or RPC-like) a user is able to send SMS text messages to multiple addressees. Figure 5-21 shows the PIM modelling using the *ArchiMeDeS* proposal. Although the system is made up by many other elements, here the focus is set only on the following building blocks and functionalities:

- **Reception Subsystem**: its main purpose is to receive delivery requests directly from the user. It contains a single service (*ReceptionService*) offering several operations such as 'sendSMS', which allows the user to submit a SMS send request to a set of previously stored recipients.
- **Storage Subsystem**: this provider stores information related to clients and SMS messages. It acts as a repository and source of information for the other two subsystems. It comprises two services:
    - **SecureDataService**: performs operations requiring a secure connection or the encryption of data. The operations supported include '*authenticate'* (to verify a client's credentials) and '*updateCredit'* (to update the client's financial information depending on the action performed over the gateway).
    - **SMSManagerService**: in charge of managing all the information related to SMS messages (such as status, SMS text) and creating listings and reports associated with a specific client.

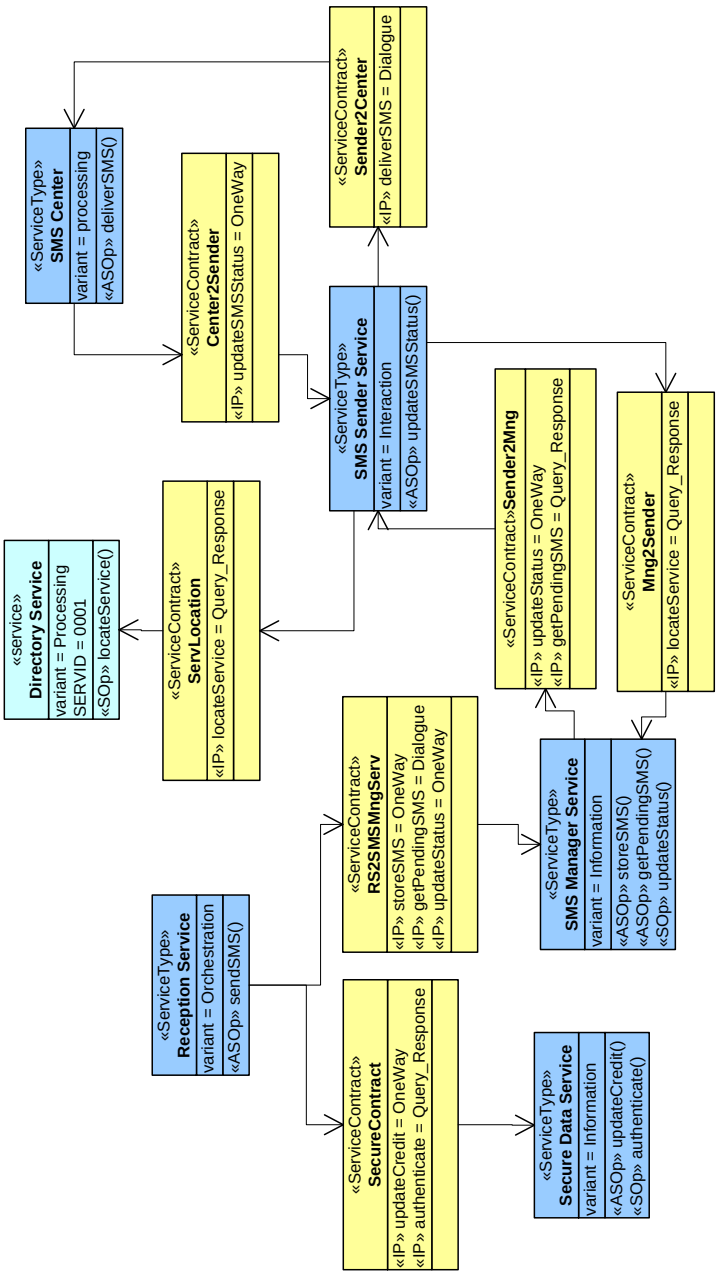

Figure 5-21. PIM architectural model of the SMPP case study.

- **SMS Processing Subsystem**: this subsystem is in charge of retrieving, processing and sending the SMS texts and related information to the specialized SMS server. It is made up of two different services:
    - **SMSSenderService**: Retrieves SMS texts from the *Storage Subsystem* and sends them to *Short Message Service Centres*.
    - **DirectoryService**: The main task performed by this service is to return the service identifier of the *SMSCenterService* which has to be used in order to send a SMS to a specific recipient.
- **SMSC (*Short Message Service Centre*)**: this provider represents specialized SMS servers capable to send the same SMS text to a predetermined number of recipients. Its functionality is enacted by one service instance which receives the SMS message and the list of recipients (*SMSCenterService*).

Once the SMPP case study has been modelled according to the *ArchiMeDeS* proposal, the next step is to represent the architectural model specification emphasizing the aspects of π-ADL that provide an adequate solution for our system as well as explaining how the structures and principles of π-ADL are adapted to our vision of PIM-level service architecture:

5.3.2 Representation of Services and Service Operations in π-ADL

Services represent computing entities performing a specific behaviour within the system architecture and thus they are specified by means of π-ADL abstractions (see Figure 5-22 for the specification of the *ReceptionService*).

```
value ReceptionService is abstraction () {
    outConn:Connection[view [operation: String, data: any]];
    output:view [operation: String, data: any];
    inConn:Connection[view [operation: String, data: any]];
    input:view [operation: String, data: any];
    if (input::operation == "sendSMS") do {
        via SendSMS send input::data
            where {resultConn renames resultConn};
        via resultConn receive result;
        compose {
             via outConn send result;
        and
             done;
    }}}
```

Figure 5-22. Specification of a Service with π-ADL.

Every service abstraction defines its own communication channels through input and output connections. The data acquired and sent by these connections comprises a description of the operation and the data associated to that message.

Depending on the operation requested, the service abstraction will transfer the control of the execution to the corresponding operation. The only behaviour associated with the service abstraction is, therefore, that of redirecting the functionality request to the corresponding operation and sending back returning values if any.

Operations, in turn, are also specified by means of abstractions as they encapsulate part of the functionality offered by services. Like any other abstraction used in the description of the service architecture, operation abstractions will receive the information tokens through connections, sending back an answer when applicable.

In π-ADL communication through the connections is performed synchronously. This means that communication with operations is synchronous. Therefore, the semantics associated with the asynchronous operations are lost since the abstraction will be blocked in a send operation until any other abstraction in the architecture perform a receive operation over that channel. In order to model asynchronous operations, the specification can be placed in one of the sub-blocks of a compose block, with the second sub-block returning immediately with the `done` keyword.

5.3.3 Representation of Service Contracts in π-ADL

As stated previously, services relate and communicate through contracts. Within the architecture these contracts are active connectors in charge of enabling the message exchange between services according to a specific pattern, represented by means of the programmatic specification of a state machine. Similarly, connectors in π-ADL are represented by means of abstractions.

In a static service environment, in which contracts between services are established at design time, all the information needed by a contract to correctly fulfil its behaviour (message exchange pattern and contractors) is defined and initialized internally within the contract abstraction when the system starts. In dynamic environments however, this is normally accomplished by transferring all the information through the channel opened simultaneously when the abstraction is executed. In both cases, the contract is able to perform the behaviour needed to transfer data requests and results from one service to another from that information.

Figure 5-23 depicts part of the analysis of a state of the message pattern execution. In it, it is shown how, in order to send anything to one of the services connected through the `Shipping` contract, a compose structure should be used:

first to send the data through the connection and second to execute the abstraction and unify the connections.

```
...
// Message Exchange Pattern: Query/Response
MessagePattern(0)::state_id = 0;
MessagePattern(0)::via_SERVID = "C";
MessagePattern(0)::op = "receive";
MessagePattern(0)::numNextStates = 1;
MessagePattern(0)::next(0)::criteria = "sendSMS";
MessagePattern(0)::next(0)::newState = 1;
...
currentState = 0;
while (currentState > -1) do {
 countState = 0;
 while (countState < maxcountState) do {
   state = MessagePattern(countState);
   countState = countState + 1;
   if (currentState == state::state_id) do {
      if (state::op == "receive") do {
       if (state::via_SERVID == "S") do
        {via inConnectionS receive inData;}
       else do {via inConnectionC receive inData;}
       } else do {
        if (state::via_SERVID == "S") do{
         compose { via outConnectionS send inData;
         and via dynamic(S) send Void where
             {outConnectionS renames inConn,
              inConnectionS renames outConn};  }
      }else do{ via outConnectionC send inData;}}
      countNewState = 0;
      maxCountNewState = state::numNextStates;
      while (countNewState < maxCountNewState) do {
       nextState = state::next(countNewState);
       countNewState = countNewState + 1;
       if (nextState::criteria == inData::operation) do
       {       via out send nextState::newState;
               currentState = nextState::newState;
       }}}}}
```

Figure 5-23. Partial specification of a Service Contract with π-ADL.

When executing the specification of a service architecture with π-ADL any behaviour defined is carried out as an independent thread of execution. However, in order to be able to perform a coordinated and joint execution, the different abstractions must be linked. Because of the dynamic nature of the service architectures, contract abstractions can be reused as the instances of the services

they communicate can vary during the lifecycle of the system. In order to achieve this behaviour, contracts (or more appropriately abstractions performing the contract role) must be able to dynamically instantiate the channel that they have to use to send or receive the data transferred in each moment. To deal with this issue π-ADL defines the `dynamic(<connection_name>)` operator. This operator represents one of the main advantages for dynamic architecture specification since π-ADL allows the transference of connections through connections (see Figure 5-24 for an illustrative example).

```
...
if (state::via_SERVID == "S") do{
 compose       {
   via outConnectionS send inData;
 and
   via dynamic(input::ServConnGroup(0)::SERVID)
        send Void
        where {outConnectionS renames inConn,
        inConnectionS renames outConn};
 }
}else do{   via outConnectionC send inData;
}
...
```

Figure 5-24. Example of a dynamic connector with π-ADL.

5.3.4 Representation of Service Composition in π-ADL

Coordination among services can be achieved by defining choreographies or orchestrations. Choreographies can be formalized with π-ADL by means of shared connections. Orchestrations, in turn, depend mostly on the code specified inside a unique abstraction belonging to a service playing the role of coordinator of the composition.

In our case study a service taking the orchestrator role is the *ReceptionService* Service which coordinates the access to the services managed by the Storage Subsystem. The *ReceptionService* consumes functionalities provided by the *SecureDataService* service for the authentication of the SMPP gateway and the storage of the incoming SMS text by means of invoking the *SMSManagerService* (*storeSMS* operation) to send the SMS texts. In π-ADL this orchestration is accomplished by taking advantage of the abstractions that represents the service contracts connecting every service involved (*SecureContract* and *RS2SMSMngServ*). In that sense, and from the point of view of the architectural representation of the case study with π-ADL, there is no special construction in the language for implementing element composition.

Figure 5-25 shows an piece of the code of the *ReceptionService* abstraction in which the consecutive invocations is reflected.

```
...
compose {
  via SecureConnTo send dataTokenSecure;
and
  via SecureContract send Void
      where {SecureConnTo renames inConnectionC,
             SecureConnFrom renames outConnectionC};
  via SecureConnFrom receive dataTokenResult;
  select dataTokenResult::data {
      case Boolean do resultVal = dataTokenResult::data;  }
  if (resultVal == false) do  {
      result = "Invalid credentials";
      compose { via resultConn send result;  and done; }  }
  if (statusVal == true) do {
  compose {
     select input::data {
       case view[phoneNumber : String,
                 recipients : String,
                 SMSText : String] do
       via RS2SSMSMngServ send input::data
            where {storeConn renames SMSConn}; } done;}
...
```

Figure 5-25. Sample service composition with π-ADL.

5.4 Concluding Remarks

Previous subsections have shown different ways of using the *ArchiMeDeS* framework so it is possible to check its suitability for the specification of software architectures using the concept of service as base for the architecture.

To perform the verification of the syntax and semantic associated to the DSL created at both the PIM and PSM levels of the MDA proposal several case studies have been used. These case studies have been either implemented completely (*GESiMED* as complete example of use with two target platforms) or used as proof of concept to check the feasibility of the DSL and designs proposed (SMPP to check the consistency of the PIM DSL and a sport metaphor to test if it is possible to represent choreographies with *ArchiMeDeS*).

Chapter 6:
Conclusions and Future Works

Once the features of the proposal have been presented (Chapter 3) together with the associated toolkit (Chapter 4) and shown its use over real case studies (Chapter 5), it is the time to draw some conclusions, make an analysis of the work accomplished and to outline the works that may be the subject of forthcoming research work. To do so, this Chapter presents: first, an analysis of achievements reached according to the objectives set out at the beginning of the dissertation; next, the main contributions of the Doctoral Thesis are explained together with the publication of the scientific results; and, finally, several open research lines are depicted as future works at the end of the Chapter.

6.1 Analysis of Achievements

At the beginning of this dissertation, Section 1.2 stated that the main objective of this Thesis was *the specification of a framework for modelling software architectures in which the specification of the architecture is obtained following a model-driven process (based on the MDA proposal) and where the concepts of the service-oriented paradigm act as foundation for the elements present in the architecture*. 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 the Thesis

To fulfil this objective, Chapter 2 provided a study of current proposals for the development of service-oriented software solutions, putting a special interest in understanding their strategy for dealing with the architectural viewpoint. Along with this study, current proposals in the scope of the application of MDD techniques to the development of service-oriented solutions were also identified.

The features of relevant proposals were gathered in order to position the current Doctoral Thesis within its research scope. The main result of this study exposed the need for the specification of a framework different from those found in the bibliography. That novel framework (*ArchiMeDeS*) would cope with the development of software architectures combining some of the techniques and principles that nowadays dominate the Software Engineering research field. The model-driven approach appeared to be a solution widely used among development proposals dealing with service-orientation. Structurally speaking, when trying to define architectures that use the service-oriented paradigm as base, some aspects were recurrent within the literature reviewed: the idea of service composition, the definition of service contracts, the awareness of having a rich interface for

services, etc. These elements were included in the moment of deciding the inner characteristics that the *ArchiMeDeS* framework should include.

On the architectural side, the analysis of the literature detected some deficiencies when giving a proper support for the specification of Software Architectures with services (such as the incorporation of technological issues into theoretically conceptual architecture solutions). In addition, the study of model-driven techniques for the specification of Software Architectures showed that there was an increasing need for creating integrated development solutions in which the structure of the architecture could be clearly and easily specified. In that sense, previous works also confirmed that counting with modelling and meta-modelling toolkits to aid in automating that process is highly desirable.

Obj.2 - Definition of a DSL for the specification of Service Architectures at PIM level

To give support for the definition of software architectures based on services and using a model-driven approach, the strategy was founded on the definition of DSLs for service architectures at each abstraction level of the MDA proposal. Accordingly, Section 3.2.1 gave a complete description of the DSL defined for the PIM abstraction level, describing both its abstract syntax (i.e. semantics defined in the form of a metamodel) and its concrete syntax (by defining a UML profile with the corresponding stereotypes for creating UML models).

As an additional and important outcome for the modelling at this abstraction level, the architectural modelling was enriched with the possibility of defining **architectural style** models. The use of architectural styles during the development of the Architecture grants a sort of flexibility by providing a way to include design decisions within architectural models. These models were defined in such a way that it is possible to merge the information of the architectural model of the system with that of the architectural style. As a direct consequence of the use of service-orientation as underlying architectural paradigm, the support for service design strategies is naturally included.

Obj.3 - Definition of DSLs for the specification of Service Architectures at PSM level

The aforementioned PIM DSL allows for the modelling of software architectures at a conceptual level using services. However, in order to take into account implementation or technological issues as part of the architectural

configuration of a software system, the PSM DSL counterpart is needed. This was the main concern of Section 3.2.3.

Analogously to the work accomplished at PIM level, at PSM level models of the architecture were specified using a hybrid approach (a DSL with UML notation). In order to precise a complete version of an architectural DSL at PSM, this task started selecting the potential target execution platforms. This aspect constituted one of the most important milestones during the research effort given that the technology of choice plays a prominent role when trying to model the system architecture at this level. Since the number of execution platforms and running environments differ greatly, this Thesis centred its attention in giving support to Service-Oriented platforms.

Having solved that problem, and knowing the existing service standards and constitutive elements of SOA implementations (thanks to the analysis accomplished in Section 2.2.2 of the state of the art), the development of a DSL at this level should be enough general to allow a possible platform migration but, in turn, enough detailed to comprise the technological particularities of concrete service technologies. For that aim, the initial consideration was that it would be comprised of a sole PSM DSL. However, after working with the technologies and platforms chosen, the definition of the PSM level of *ArchiMeDeS* ended up in considering the separation of that level in two different levels: PDM and TDM. Because of that, section 3.2.2 was dedicated to present two complementary levels of a DSL proposal for the PSM level: on the one hand, a PDM metamodel with all the common elements that any service-oriented execution platform may share (at least the three platforms chosen); and, on the other hand, concrete TDM metamodels for three well-known service-based platforms: Web services, Grid Services and REST services. Similar to the case of the PIM DSL, every DSL at PSM level was given a concrete syntax based on UML by providing the description of several UML profiles.

Obj.4 - Specification of model transformation rules

The DSLs defined for the representation of software architectures at any level of the MDA approach did not offer any groundbreaking advantage for the either architecture specification or software development process different from other modelling initiatives. However, one of the main assets of the *ArchiMeDeS* framework relied on the possibility to (semi-)automatically obtain one model from the information of the other one. Accordingly, in conjunction with the definition of the DSLs, a set of rules was defined to overcome this issue (fully explained in

section 3.2.3), either PIM-to-PIM (inclusion of architectural style features) or PIM-to-PSM (both PIM-to-PDM and PIM-to-TDM).

Transformation rules defined for that aim allowed performing two different kinds of transformations: model merge and abstraction level change. The former one corresponds to a merging process between the information gathered in an architectural style model with that of the PIM metamodel. To do so, a weaving process based on the use of the *service role* concept was depicted in Section 3.2.3.1. The latter, in turn, refers to a set of correspondences between the concepts of the PIM metamodel and the PDM/TDM models of the architecture. All these transformations were, firstly, defined in natural language indicating a set of correspondences; and, secondly, implemented using the ATL language, largely considered as one of the best options for model transformations due to its completeness and tool support as could be derived from the study of works in this subject (Section 2.2.1).

Obj.5 - Creation of a toolkit supporting architectural modelling

To reach that goal, the tools derived from the EMP (EMF, GMF, AMW, etc) where chosen to build a toolkit for modelling the DSLs defined within *ArchiMeDeS*. Relying on the Eclipse platform, the first step was the definition of the metamodels in EMF. As a result, editors supporting the definition of the models conforming each DSL where obtained. This way it was possible to count with an initial tree-like editor for model edition. The next step was to provide with a graphical support in order to provide with a UML support to the toolkit. In that case, the GMF extension for Eclipse was used.

Support for model transformation was also implemented within this toolkit. To do so, the transformation rules were implemented in ATL. Moreover, the weaving process for the superimposition of architectural style features in PIM models was implemented using the AMW plug-in for Eclipse. The building process of this toolkit was fully explained in Chapter 4.

The developed toolkit was thought not only as requirement to be fulfilled in order to consider the architectural framework complete but also as instrument for the automatic model syntax verification according to the metamodels specified. In addition, the tool aids in the automation of software architecture specification since it supports the execution of model transformations.

Obj.6 - Validation of the architectural framework by means of its application to several case studies

As stated previously, the definition and implementation of a modelling environment was used with the purpose of checking the correctness of the proposed DSLs and the accuracy and appropriateness of model transformations defined within the *ArchiMeDeS* framework. However, in order to check the relevance and feasibility of the proposal, some case studies were developed.

To check the correctness of the DSLs, the validation efforts were divided according to abstraction levels. On the one hand, for the **PIM DSLs**, the strategy followed was the representation of a concrete case study (based on the architectural representation of a SMPP system) using a specific formal ADL (π-ADL) and making a comparison between the representation of the case study using the formal structures of π-ADL and the corresponding PIM model represented using *ArchiMeDeS*. On the other hand, at **PSM level**, the proposed DSLs were validated by comparing them with their counterpart representation using the standard languages of the technological platform of choice. Accordingly, the resulting PSM models were used as a proto-architecture for the creation of the source code regarding both a Web Service platform (via W3C standards) and a Grid Service platform (using *Globus Toolkit*) as target implementation technologies.

The case study used to check the *ArchiMeDeS* framework as a whole is based on the architecting of a system for the management of medical digital images named *GESiMED*. The architecture of this system was modelled at both PIM and PSM levels. The final TDM models were used to manually create the source code corresponding to the implementation of *GESiMED* as it was reflected in Chapter 5. In order to cover the validation of other aspects not considered when using the *GESiMED* system, the architectural modelling of other case studies were used as partial examples. For instance, a sport metaphor (*Pick and Roll* situation in a basketball game) was used to prove the suitability of the proposal for the specification of service choreographies; and, as it has been mentioned before, an emulation of a gateway for sending SMS messages was used to provide with a comparable representation of the architectural models with π-ADL.

6.2 Main Contributions

This Thesis has resulted in a number of contributions, regarding not only the main asset of this research (the *ArchiMeDeS* framework) 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.

Specification of a framework for model-driven development of Software Architectures

The main contribution of this Thesis has been the specification of *ArchiMeDeS*, a full-fledged framework that allows the specification of software architectures and with a distinguishable feature: the use of the SOC paradigm to build up the Architecture. Few of the proposals analyzed in this dissertation take advantage of the benefits that the use of services may offer to the specification of software architectures. An MDA-based model architecture for architectural specification has been defined to improve the software architecture design by focusing on its capabilities to bridge the gap between high-level conceptions of modern business organizations and their potential implementations.

The features of *ArchiMeDeS* contribute to establish a novel form of tackling the definition of system architectures as well as its relationship with other aspects of system development. The use of models for that purpose also empowers the understanding of the system-to-be and the steps and elements that may be defined as part of that development process. This is accomplished through the specification of a concrete grammar for architectures in the form of DSLs at different abstraction levels.

As it has been largely stated during this dissertation, the use of MDA as foundational approach for architectural specification, with its PIM/PSM separation, brings the opportunity to diversify the implementation target and allows easing possible future migrations of the system, or even a change in the supporting platform or implementation technology of choice. The inverse procedure (to obtain a PIM representation from different PSM models), although not considered as part of this Thesis, would also be possible.

Besides, the framework includes the possibility to define architectural decisions and constraints in the form of architectural styles. The novelty from other proposals is that, having into account that the architecture is built upon services, it is possible not only to give support to traditional architectural styles but also to design patterns that have arisen with the use of services within the enterprise scope, leading to some well-known architectural strategies based on services. In addition, as it was pointed out in the moment of dealing with architectural model transformations (Section 3.2.3), *ArchiMeDeS* represents a framework, general enough, for the support of other kinds of architectural

transformations, such as element merge or decomposition, into more complex service-based processes.

Support for semi-automatic generation of platform specific service architectures using a graphical environment

Initially conceived as a way to check model conformance, the creation of a graphical toolkit for the definition of models and metamodels required to represent software architecture configurations, surpassed its expectations and ended up in a powerful toolkit that supports not only the drawing of the models (either using a tree-like or an UML-based interface) but also a way to test the correctness of the models, the syntactic coherence of the metamodels and, eventually, the validation of the models obtained as a result of executing model-to-model transformations.

The platform used as basis for the toolkit, Eclipse, proved to be an adequate choice to reach the objectives marked for this Thesis. It demonstrated to be a valuable option for both modelling and meta-modelling (using Eclipse extensions such as EMF, GMF, etc) but, and maybe more importantly, it allowed to incorporate the definition of model transformation rules (expressed in ATL language) inside the same framework, so it was possible to semi-automate the desired transformation processes. This was possible thanks to its capability to be easily extended with new functionalities throughout the inclusion of specialized independent modules.

Model transformation has been accomplished through transformation processes that ranged from a simple model-to-model transformation (in the case of the obtaining of the PSM model of the architecture from its corresponding PIM model) to the allowance of defining a full superimposition process to enrich PIM models with information gathered in models representing architectural styles.

Definition of a state-of-the-art evaluation in the scope of three engineering disciplines of increasing importance and demonstrated synergy

The task of bringing together the features of the three main topics considered in the dissertation has required focusing on several distinct premises: on the one hand, the selected **criteria** needed to have enough relevance in the context of software specification; for example, by referring to the role given to the Architecture within a development process (in the scope of Model-Driven Engineering) or the base paradigm of the Architecture (in the context of Service-Orientation). On the other hand, the criteria used needed to be significant enough to detect the different approaches that research initiatives propose; for instance, the abstraction level at which the software architecture specification is considered

or the modelling approach followed by model-driven initiatives. To help in this criteria selection, the research accomplished during the making of this Thesis inherits the questions and solutions derived from the research scope in which the Thesis is framed. Previous research works resulted in the identification of issues that needed to be solved regarding MDE, SOA or architectural specification. The previous experience in these topics also conditioned the selection of one factor or another.

The result is, first, a short but representative criteria set that allows for the identification of the different approaches that are somehow related to the abovementioned topics; and, second, the performance of an initial outline of what features include the most relevant initiatives in that field.

All in all, the accomplishment of the state of the art represented one of the first steps to define what should be the features of the *ArchiMeDeS* framework. This state of the art allowed focusing on the need for incorporating together novel engineering approaches (such as MDA), key software concerns as guiding development artefacts (such as architectural specification) and increasingly spread computing paradigms (such as SOC) towards a new way of achieving success in Software Engineering.

Implementation of a fully functional case study derived from the research in neuromedicine

The last remarkable contribution refers to the main case study used to validate the proposal. The *GESiMED* system represents a real example in which the foundational aspects of this dissertation can be used. This way, the *ArchiMeDeS* framework is used to provide with a technological answer to the business needs detected in a concrete domain such as that of research with neuroimages, as some publications confirmed ([96]).

Using a model-driven approach to specify the architecture of GESiMED provides a way of easing the platform migration of this system. This issue can be achieved thanks to the possibility of defining the system architecture at a conceptual level (PIM) and then selecting a concrete target platform (TDM). In addition, the capabilities provided by *ArchiMeDeS* for the superimposition of architectural styles on architectural models facilitates the adaptation of the system to specific business constraints found in the context of the research in neuroscience, for example, due to processing load requirements.

6.3 Scientific Results

Some of the results of this Thesis have been published in different forums, both national and international. In the following, those publications are grouped according to the type of publication.

- ✓ **Articles in International Journals**
  - M. López-Sanz, J. M. Vara, E. Marcos, C. E. Cuesta. A Model-Driven Approach to Weave Architectural Styles into Service-Oriented Architectures. *International Journal of Cooperative Information Systems. **To be published in June 2011***. **Impact Factor JCR: 0.528 (ISI JCR 2009)**
  - M. López-Sanz, C. E. Cuesta, E. Marcos, J. Domínguez. Developing Coordination Strategies using a Service-Oriented Model-Driven Approach. *International Journal of Web Services Practices*. Vol.:3 (3-4), pp: 115-121. Ed.: Web Services Research Foundation. Eds.: Sang Yong Han. ISSN: 1738-6535. Seoul (South Korea). November, 2008.
  - M. López-Sanz, C. Acuña, C. E. Cuesta, E. Marcos. Modelling of Service-Oriented Architectures with UML. *Electronic Notes in Theoretical Computer Science (ENTCS).* Vol.: 194 (4), pp 23-37. Ed.: Elsevier. ISSN: 1571-0661 DOI: 10.1016/j.entcs.2008.03.097. Amsterdam (The Netherlands). April, 2008
  - J.A. Hernández, C. Acuña, V. de Castro, E. Marcos, M. López-Sanz, N. Malpica. WEB-PACS for Multicenter Clinical Trials. *IEEE Transactions on Information Technology in Biomedicine*. Vol.: 11, Nº 1. pp: 87-93. Ed.: IEEE Computer Society, IEEE Engineering in Medicine and Biology Society. ISSN: 1089-7771. New York (USA). April, 2008. **Impact Factor JCR: 1.436 (ISI JCR 2007)**
  - V. De Castro, E. Marcos, M. López-Sanz. A Model Driven Method for Service Composition Modelling: A Case Study. *International Journal on Web Engineering and Technology*. Vol.: 2 (4), pp: 335-353. Ed.: Inderscience Enterprise Ltd. ISSN: 1476-1289. Switzerland. July, 2006.
- ✓ **Articles in Iberoamerican Journals**
  - C. J. Acuña, E. Marcos, V. de Castro, J. A. Hernández, M. López-Sanz. Gestión de imágenes médicas a través de la Web. *Revista colombiana de computación (RCC).* Vol.: 8 (1). pp. 1-11. ISSN: 1657- 2831. Eds.: E. Carrillo Zambrano, A. Fedossova. Bucaramanga (Colombia). June, 2007

- ✓ **Articles in International Conferences**
  - M. López-Sanz, C. E. Cuesta, E. Marcos. Formalizing High-Level Service-Oriented Architectural Models Using a Dynamic ADL. *Proceedings of the Workshop on Adaptation in serVice EcosYsTerms and ArchiTectures. On the Move to Meaningful Internet Systems: OTM 2010 Workshops (AVYTAT'10),* **LNCS 6428-067**, pp. 57-66. Eds.: R. Meersman, P. Herrero, T. Dillon. ISBN: 978-3-642-16960-1. Ed.: Springer-Verlag, Berlin, Heidelberg (Germany), 2010.
  - M. López Sanz, C. J. Acuña, V. de Castro, E. Marcos, C. E. Cuesta: Using an Architecture-Centric Model-Driven Approach for Developing Service-Oriented Solutions: A Case Study. *Proceedings of the International Workshop on System/Software Architectures (IWSSA'09). On the Move to Meaningful Internet Systems: OTM 2009 Workshops*, **LNCS 5872**, pp. 350-359. Eds.: R. Meersman, P. Herrero, T. Dillon. ISBN 978-3-642-05289-7. Ed.: Springer-Verlag, Heidelberg (Germany), 2009. (Acceptance ratio: 41%)
  - M. López-Sanz, J. M. Vara, E. Marcos, C. E. Cuesta. A Model-Driven Approach to Weave Architectural Styles into Service-Oriented Architectures. *Proceedings of the First International Workshop on Model-Driven Service Engineering and Data Quality and Security (MOSE+DQS'09),* Hong Kong, China. Eds. D. Cheung, Il-Yeong Song, W. Chu, X. Hu, J. Lin, J.Li and Z. Peng. ISBN: 978-1-60558-816-2. (Acceptance ratio: 60%)
  - M. López-Sanz, C. E. Cuesta, E. Marcos, J. Dominguez. Developing Coordination Strategies using a Service-Oriented Model-Driven Approach. *Proceedings of the Fourth International Conference on Next Generation Web Services Practices (NWESP'08).* pp. 198-203. Editorial: **IEEE Computer Society**. Eds.: A. Abraham, S. Yong Han. ISBN: 978-07695-3455-8.
  - M. López-Sanz, Z. Qayyum, C. E. Cuesta, E. Marcos, F. Oquendo. Representing Service-Oriented Architectural Models using π-ADL. Emerging Research Paper. *Proceedings of the 2nd European Conference on Software Architecture (**ECSA'08**),* **LNCS 5292**, pp.273-280. Eds. R. Morrison, D. Balasubramaniam, K. Falkner, 2008. ISBN: 978-3-540-88029-5. ISSN: 0302-9743. (Emerging research papers acceptance ratio: **14.5%**).

- M. López-Sanz, C. J. Acuña, C. E. Cuesta, E. Marcos. Defining Service-Oriented Software Architecture Models for a MDA-based Development Process at the PIM level. Working Session Paper. *Proceedings of the 7th IEEE/IFIP Working Conference on Software Architecture (**WICSA'08**)*, pp. 309-312. Editorial: **IEEE Computer Society**. Eds. P. Kruchten, D. Garlan, E. Woods, 2008. Lugar de publicación: New York (USA). ISBN: 0-7695-3092-3. (**CORE A Conference,** acceptance ratio: **28.75%**).
- M. López-Sanz, C. J. Acuña, C. E. Cuesta, E. Marcos. UML Profile for the Platform Independent Modelling of Service-Oriented Architectures. Poster. *Proceedings of the 1st European Conference on Software Architecture (**ECSA'07**)*, **LNCS 4758**, pp. 304-307. Eds. F. Oquendo, 2007. ISBN: 978-3-540-75131-1. ISSN: 0302-9743. (Acceptance ratio: **25.78%**)
- M. López-Sanz, C. J. Acuña, C. E. Cuesta, E. Marcos. Modelling of Service-Oriented Architectures with UML. *Proceedings of the 6th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA'07)*. Pp.: 21-36. Eds. C. Canal, P. Poizat, M. Virola.
- M. López Sanz, V. de Castro, E. Marcos, J. L. Bosque. A Comparative Study between Web Service and Grid Service Developments in a MDA Framework. *Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS'07).* Vol. ISAS. Eds. J. Cardoso, J. Cordeiro, J. Felipe, 2007, pp. 114-121. ISBN: 978-972-8865-90-0. (Acceptance ratio: **12%**)
- V. de Castro, M. López-Sanz, E. Marcos. Business Process Development based on Web Services: A Web Information System for Medical Images Management and Processing. *Proceedings of IEEE International Conference on Web Services (ICWS'06)*. **IEEE Computer Society**. Ed.: F. Leymann, L.J. Zhang, 2006, pp. 807-814. ISBN: 0-7695-2669-1 (**CORE A Conference,** Acceptance ratio: **33%**)
- M. López-Sanz, E. Marcos, J. L. Bosque. A Proposal of Grid Middleware Architecture for Medical Image Management. *Proceedings of the IADIS International Conference–Applied Computing 2006*. IADIS Press. Eds. N. Guimarães, P. Isaias, A. Goikoetxea. ISBN: 972-8924-09-7. (Acceptance ratio: **23%**)

- ✓ **Articles in Iberoamerican Conferences**
  - M. López-Sanz, C. J. Acuña, C. E. Cuesta, E. Marcos. MDA para Arquitecturas Orientadas a Servicios: Un perfil UML a nivel PIM. *8th Argentinean Symposium on Software Engineering (ASSE'07).* Art. nº 10, págs. 1-13. Eds. Roberto Giordano Lerena, Isabel Passoni, Pablo Montini. ISSN: 1850-2792 (ASSE), 1850-2776 (JAIIO)
  - V. de Castro, E. Marcos, M. López-Sanz. Service Composition Modeling: A Case Study. *Proceedings of 7th Mexican International Conference on Computer Sciences (ICCS'06)*. IEEE Computer Society. Ed.: Sergio Rajsbaum, 2006, pp. 101-108. ISBN: 0-7695-2666-7. ISSN: 1550-4069. (Acceptance ratio: 26%)
  - V. de Castro, M. López-Sanz, E. Marcos. Modelado de Procesos de Negocios Basados en Servicios Web. Poster. *Actas del 9º Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes Software (IDEAS'06)*. Eds.: Jaelson Castro, Luca Cernuzzi, Silvia Gordillo. ISBN: 950-34-0360-X.
- ✓ **Articles in National Conferences**
  - M. López-Sanz, C. E. Cuesta, E. Marcos. Modelos Arquitectónicos Orientados a Servicios: Definición y Desarrollo Formal sobre un Caso de Estudio. *Proceedings of the Third International Workshop on Autonomic and Self-Adaptive Systems (WASELF'10)*, pp. 11-20. Eds: J. Cámara, C. E. Cuesta, M. A. Pérez-Toledano. Ed.: SISTEDES 2010. ISSN: 1988-345
  - M. López-Sanz, C. E. Cuesta, E. Marcos. Modelado de Coreografías de Servicios con UML 2.1. *Actas de las IV Jornadas Científico-Técnicas en Servicios Web y SOA (JSWEB'08)*, pp. 1-14. Eds.: J. M. López Cono, A. Vallecillo y A. Ruiz-Cortés. ISBN-13: 978-84-691-6710-6
  - M. López-Sanz, C. E. Cuesta, E. Marcos, J. Domínguez. Modelado Arquitectónico Orientado a Servicios de Estrategias de Coordinación Inspiradas en Tácticas Deportivas. *Actas del Taller de Trabajo Integración de Aplicaciones Web (ZOCO'08)*. Editores: J. L. Álvarez, J. L. Arjona, R. Corchuelo, D. Ruiz.
  - M. López Sanz, V. de Castro, J. L. Bosque, E. Marcos. Estudio comparativo del desarrollo con Servicios Web y Servicios Grid en un marco basado en MDA: aplicación a un caso de estudio. *Actas de las II Jornadas Científico-Técnicas en Servicios Web (JSWEB 2006).* Eds.: M. Lama, F. Curbera, J. C. del Arco, E. Sánchez, 2006, pp. 23-29. ISBN: 84-690-2398-5.

- G. López, M. López-Sanz, V. de Castro, E. Marcos. Diseño de Procesos de Negocios que requieren la intervención del Usuario: Un estudio sobre las alternativas de implementación. *Actas del Taller Desarrollo y Mantenimiento Ágil de Aplicaciones basadas en Servicios Web (ZOCO'06)*. Eds.: Rafael Corchuelo, David Ruiz y José Luís Arjona, 2006, pp. 39-49. ISBN: 978-84-690-5792-6
- V. de Castro, M. López-Sanz. Modelado de Composición de Servicios: Un Caso de Estudio. *I Jornadas Científicos-Técnicas en Servicios Web – (JSWEB'05)*. Eds.: E. Marcos, J. M. Alonso, V. de Castro, J. C. del Arco. ISBN: 84-9732-455-2. (Acceptance ratio 49%)

6.4 Future Works and Open Research Lines

Despite the contributions made on this Thesis, as the research tasks progressed, several directions to further work were detected. Some of them were just not considered as objectives of this Thesis (support for non-functional requirements for example) while others have emerged during the development of this work (extending the PSM DSLs to support other target executing platforms). Next subsections summarize some of them.

Definition of a complete model-driven development process in which *ArchiMeDeS* architectural models play a guiding role

The *ArchiMeDeS* framework allows the specification of software architectures at different abstraction levels. However, considered within the methodological context in which the Thesis is framed (MIDAS) the integration with the model-driven development of other concerns in MIDAS should be completed. This way, and although some of the issues needed were mentioned in Section 3.3.2, one of the future works that is currently under development is the integration of *ArchiMeDeS* and *SOD-M* [49] a service-oriented method for the development of information systems centred in the behavioural concern. In addition, and also within MIDAS, the definition of the interface concern through an architecture-centric model-driven process is also a subject of intense research efforts and the main focus of a Thesis already in progress.

In summary, it is quite clear that the information modelled in the architecture can be seen as the driving aspect of wider model-driven methodological frameworks, and for MIDAS in particular. In that sense, the complete definition of MIDAS as architecture-centric model-driven development methodology is highly encouraged. Moreover, since model-driven processes defined as part of MIDAS are also being supported by tools (and *meta-tools*)

implemented over the Eclipse platform, another line of research being currently considered is the integration of all the *ArchiMeDeS* toolkit modules with the tools supporting the other MIDAS' development concerns.

Support of additional platforms, either service-based or not

The most used and widespread technologies for the implementation of service architectures are those related to Web Services (either REST-compliant or not) and Grid Services. An architectural support for these platforms has been clearly defined as part of the current Thesis. However, many standards have proliferated to cope with adjacent issues to the use of these technologies. The proposed DSLs cannot be considered fully complete without accepting a lack of support for some of the existing standards and languages. This dissertation has centred its attention in technologies that have a direct influence on the specification of any feature related to the structural view of a system, that is, within its architectural view (interface definition, resource specification, contract establishment, message exchange, and so on).

The rest of the aspects that may have influence over the architectural configuration (such as policies, monitoring or discovery issues) and its related standardization initiatives open a field of research that may end in obtaining a full view of the architecture with additional information. However, and bearing in mind that the MIDAS methodological framework fosters a multiple concern view of the development process, to support all the additional aspects it would be highly recommended to create a separated model set for that aim.

On the support of additional platforms different to those based on service technologies, investigation in this line would probably require a modification of the DSL created at the PDM level, and probably the TDM, so it is possible to create an extension for the support of concrete features of other platforms and technologies. Upcoming technologies, on which novel computing paradigms are based (such as *Cloud Computing*), could be selected as target platforms of interest.

Formal representation of the DSLs syntax

Though the use of models for the specification of Architectures represents a step forward in the ease of the development of Software solutions, having a formal representation of architectural elements and relationships between them presents several supplementary benefits. Counting with an executable version of the architectural configuration at a conceptual level, the possibility of applying a mathematical base for verifying and validating the correctness of the architecture

or the capability of incorporating a formal way of supporting dynamism in models are among those benefits.

All these research lines step inevitably on the selection of a target ADL to which port the concepts gathered in the DSLs created to define Software Architectures with Services. This aspect was initiated with the use of π-ADL to check the reliability and correctness of the modelling proposal for the PIM level of abstraction (see Section 5.3). However, and with the aim of consolidating a model-driven approach for the whole framework, the transition from the presented DSLs to a concrete ADL should be done by defining model transformations and mappings between the grammars of both languages. Once the relation is established (bidirectionally) it would be possible to seamlessly automate the transition from one approach to another. Afterwards, it could be possible to operate the formal representation of the architecture to, for instance, reason about the architectural model to check certain properties about the system configuration or support model updating through architectural reconfiguration [43].

Full support for architectural rationale modelling

Apart from the identification and inclusion of architectural styles as a way of personalizing the architecture according to specific designs or constraints, there are other several aspects that have an important influence over the system architecture. On the one hand, the architectural decisions themselves that lead to opt for one or another architectural design; and, on the other hand, the existence of low-level design patterns highly dependent on the technology of choice may require to modify or change the architectural configuration of a system.

Efforts in this line would need to address the creation of models and metamodels allowing the representation of the architectural rationale at any abstraction level. In addition, model transformation processes should be defined in order to integrate those new models as part of the *ArchiMeDeS* framework. Due to the availability of a toolkit for *ArchiMeDeS*, the support for architectural rationale design could be easily supported by creating the corresponding modules for the toolkit presented in Chapter 4.

Support for dynamic architectures

As it has been largely discussed in the introduction and other chapters of this dissertation, the specification of software architectures may be taken from different points of view and with different degrees of detail. This Thesis, for example, only gives support for the modelling of static representation of software architectures based on services. However, it is a reality that any system may suffer

changes and evolve during its lifetime. This changeability has a direct impact on the structure of a system and thus it must be reflected in its architecture.

When adopting services as building brick for the architecture, the proposed architectural modelling should support some kind of variability in the architectural representation to one of the different types of dynamisms [44]. In its current state of development, the *ArchiMeDeS* proposal only gives support to a dynamism of data, that is, the models of the architecture created allow the representation of systems (based on services or not) that may change the (type of) information exchanged. The environment does not need to change its component elements to give support to this kind of dynamism.

A second type of dynamism may be a scenery in which the elements present at run-time may vary, that is, that new instances of the elements defined in the architecture appear within the scope of the system but playing a role and with a behaviour already known and defined in the scope of the system. This type of dynamisms requires that new connections between the already existent elements appear and be formed correctly.

The third type of dynamism is that in which new types of elements appear to play a role within the environment and execute a previously unknown behaviour that must maintain the coherence and stability of the system as a whole. To do so, it is important that the system know both the current architectural state of its components and the upcoming situation in which new elements may take part in order to achieve a concrete goal.

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.

A.1 Antecedentes

Es un hecho ampliamente reconocido que durante las últimas décadas, las tecnologías de la información (TI) se han posicionado en el centro de los procesos de negocio. La consecuencia directa ha sido la aparición de una fuerte dependencia de las empresas hacia los sistemas de información. Esta dependencia no sólo se refiere a la utilización de recursos y dispositivos hardware sino que también, e incluso más significativamente, se amplía al uso de sistemas software. El ejemplo más claro puede encontrarse en la red Internet, entendida como fuente y destino de conocimiento, recursos y negocios. Por un lado, ha propiciado la aparición de una nueva oleada de empresas y compañías basadas íntegramente en la Web. Por otra parte, ha supuesto una serie de retos tecnológicos (mejora del rendimiento, alta disponibilidad, escalabilidad óptima, necesidades de estandarización, entornos de integración, etc.) que los profesionales en este ámbito deben resolver eficientemente.

Este cambio de atención hacia el *universo online* implica, desde el punto de vista de la investigación en software, la necesidad de diseñar no sólo nuevas soluciones software, sino también la especificación de nuevas metodologías de desarrollo, plataformas de ejecución, herramientas de soporte y estrategias de diseño y gestión que, en principio, difieren de las conocidas hasta la fecha. Internet establece unas condiciones de desarrollo que hacen que las técnicas de ingeniería actuales no sean apropiadas en este contexto o, al menos, no sean suficientes para abordar las nuevas necesidades impuestas. Además, en una era de reajuste económico, es importante que las aproximaciones de investigación definidas para este contexto puedan ser aplicadas a otros contextos (no solamente centrados en la Web) de tal forma que se centren en aquellos aspectos cruciales de las TI (por ejemplo el desarrollo de la arquitectura software) facilitando el desarrollo de soluciones altamente flexibles.

El desarrollo dirigido por modelos como forma de abordar la Ingeniería del Software actual

Observando la evolución de la Ingeniería del Software, es posible ver cómo la existencia de estrategias, procesos y metodologías concretas para el desarrollo de soluciones software conlleva una mejora de aspectos tales como la calidad, el mantenimiento, la robustez o la escalabilidad de los sistemas desarrollados [193]. Tradicionalmente, las técnicas de Ingeniería del Software se han basado en la conceptualización de las características del sistema bajo desarrollo, abstrayéndose tanto del contexto de la solución como de su entorno. Partiendo de esta premisa, parece bastante obvio pensar que la estrategia a seguir a la hora de ofrecer una solución adecuada a los retos que supone la tendencia actual de los negocios debe ir por esta línea. En este sentido, **el uso de razonamientos basados en diagramas, es decir, mediante el uso de modelos, para todo el ciclo de vida del software parece ser el camino correcto a seguir para asentar los principios de la Ingeniería del Software moderna.**

Como consecuencia, durante los últimos años, la *Ingeniería Dirigida por Modelos* (MDD por sus siglas en inglés) [94] ha crecido en importancia hasta convertirse en una de las estrategias más exitosas a la hora de desarrollar sistemas de información de nueva generación. Aunque la posibilidad de representar las características de un sistema mediante modelos y diagramas es conocida desde los principios de la Ingeniería del Software, el **establecimiento de diferentes niveles de modelos**, desde diferentes puntos de vista y, más específicamente, teniendo la habilidad de definir **reglas de transformación aplicadas a modelos**, son características que se encuentran entre las principales razones de su éxito. Además, los conceptos detrás de la definición de *Lenguajes Específicos de Dominio* (DSLs) asociados a dichos modelos hacen que también las aproximaciones MDD se conviertan en unas alternativas altamente útiles para dirigir las estrategias de Ingeniería del Software hoy en día.

Entre las múltiples propuestas existentes que se basan en los principios de MDD es especialmente interesante la iniciativa MDA (*Arquitectura Dirigida por Modelos*) publicada por el consorcio OMG en 2001 [171]. MDA ha atraído la atención tanto de los departamentos de I+D de la empresa como del ámbito académico. Este hecho es fácilmente comprobable atendiendo a la gran cantidad de iniciativas que han aparecido en los últimos años siguiendo esta aproximación [62][62]. MDA, además de considerar **el modelo como el artefacto primordial para la definición de software**, sugiere la separación de diferentes tipos de modelos **agrupados por niveles de abstracción**. Estos niveles abarcan desde el modelado de los aspectos del sistema relacionados con el negocio (CIM) a

aquellos modelos que reflejan todos los aspectos tecnológicos de la implementación del sistema en desarrollo (PSM). Sin embargo, la mayor contribución de MDA al desarrollo de sistemas no se queda en esta separación en niveles de abstracción, sino en el hecho de apostar por la posibilidad de **definir transformaciones** entre los conceptos especificados en metamodelos. Estos metamodelos son modelos que describen los conceptos utilizados en los modelos que son conformes a ellos [200]. Las reglas de transformación facilitan, en este sentido, una progresión mucho más sencilla del desarrollo del sistema. Eventualmente, los modelos que están más próximos al nivel de implementación deben servir como origen del código fuente del sistema, que, siguiendo este mismo razonamiento, debería ser obtenido de forma automática (o al menos semi-automática).

A pesar de los beneficios conocidos de MDA, se han detectado en los últimos tiempos algunas deficiencias en esta propuesta. Trabajos actuales [136][150] argumentan que **una de las principales desventajas de MDA es la falta de una definición precisa del rol que la Arquitectura debe tener en el contexto de una arquitectura de modelos**. En su especificación original, no hay ninguna mención explícita a la Arquitectura como parte de la estructura de modelos asociada a un proceso de desarrollo basado en MDA. Muchos de los métodos y metodologías actuales que siguen la aproximación MDA evitan modelar explícitamente el aspecto arquitectónico del sistema. Estas propuestas suelen mezclar el diseño topológico del sistema (estructura del mismo a partir de componentes, conectores, y relaciones entre ellos) con la definición de sus funcionalidades (comportamiento dado a cada uno de los elementos con un rol determinado durante la ejecución del sistema)[16][95]. Además, merece la pena resaltar que la falta de una especificación precisa de la arquitectura, como modelo independiente dentro de la arquitectura de modelos, restringe la flexibilidad del proceso de desarrollo en si. Un ejemplo ilustrativo puede encontrarse cuando se quieren aplicar diferentes estilos arquitectónicos como forma de abordar entornos en los que los cambios derivados de las necesidades de negocio suceden con frecuencia. Conociendo los beneficios que aporta la aplicación de estilos arquitectónicos durante el diseño y desarrollo del software [208], no incluir los modelos arquitectónicos como artefactos independientes dentro del proceso de desarrollo obliga a fijar, por defecto, un estilo arquitectónico concreto

El paradigma orientado a servicios como base conceptual para la definición de nuevos procesos de ingeniería

Por otro lado, la necesidad de desarrollar sistemas que se adapten (o, idealmente, se auto-adapten) a cambios en los requerimientos es más evidente al observar la dirección que la Economía global está tomando en los últimos años. La externalización y fragmentación de los procesos industriales ha favorecido la adopción de modelos de negocio basados en ***Servicios***, entendidos como entidades económicas independientes con un valor de negocio y un rédito funcional tanto para la empresa como para el cliente final [114]. Como consecuencia, se ha demostrado la necesidad de describir procesos de negocio que soporten el ciclo de vida completo del software teniendo en cuenta este transfondo económico, en el que es crucial entender que el software desarrollado debe tener una estructura y comportamientos que pueden ser susceptibles de cambiar. En este contexto, las aproximaciones MDD han demostrado ofrecer una forma de crear procesos adecuados basados en estos requisitos, basados en **el alineamiento del modelado de negocios altamente cambiantes con las plataformas y tecnologías de implementación actuales**. Sin embargo, con el fin de abordar estos nuevos tipos de procesos de desarrollo se debería utilizar un paradigma de computación subyacente adecuado. Desde una perspectiva tecnológica, estos nuevos modelos de negocio han recibido su respuesta en la forma del paradigma *Orientado a Servicios (SOC)*[4][182] junto con una serie de estándares y lenguajes asociados. Mediante la utilización de SOC es posible crear sistemas dinámicos poco acoplados que se adaptan perfectamente a posibles necesidades futuras de los negocios.

Sin embargo, la correspondencia entre los servicios de alto nivel (como partes de un procedimiento económico/financiero) y su implementación mediante las tecnologías de servicios actuales no es sencilla y requiere un proceso de transformación que está lejos de considerarse trivial. Actualmente, el paradigma SOC representa un cambio en la forma en la que el software es *analizado* (por ejemplo debido a la necesidad de incluir nuevas dependencias, restricciones o políticas de negocio), *diseñado* (la cantidad de estándares al respecto implica, por ejemplo, la selección del más adecuado a cada caso), *construido* (por ejemplo, las restricciones establecidas por las plataformas de ejecución condicionan el soporte tecnológico), *distribuido* (debido a la necesidad inherente de sincronismo en entornos distribuidos) y *usado* (los conceptos SOC deben coexistir con otros paradigmas de computación). Todo esto ha llevado a que investigadores y desarrolladores deban pensar de nuevo las técnicas que han de utilizarse para construir software con este paradigma. En este sentido, las propuestas dirigidas

por modelos (y la propuesta MDA en particular), de nuevo, ayudan a llenar el salto existente entre las especificaciones de servicios de negocio y el desarrollo de sistemas de información orientados a servicios.

El uso de aproximaciones de desarrollo basadas en modelos (MDD) en conjunción con el paradigma SOC ha demostrado ser de gran ayuda en los últimos años. El gran número de proyectos europeos dedicados a estos temas [62][62], en los que muchas de las grandes empresas a nivel mundial están involucrados, apoya este razonamiento. La definición de diferentes modelos agrupados en niveles de abstracción y la especificación de reglas de transformación de modelos convierten a MDA en una de las mejores alternativas a la hora de decidir la forma en la que las soluciones software orientadas a servicios deben ser desarrolladas. En efecto, debido a la existencia de una especificación jerárquica de modelos, **el uso de MDA facilita la transición desde servicios de alto nivel hacia sus correspondientes servicios tecnológicos**, además de facilitar la migración de plataforma y mejorar la adaptabilidad del sistema.

Debido a su naturaleza débilmente acoplada, la aplicación del paradigma SOC al desarrollo de software tiene un impacto directo sobre el aspecto arquitectónico. No sólo es necesario definir la estructura topológica del sistema y sus elementos constitutivos, sus estructuras de composición y las relaciones existentes entre ellos sino que también es necesario decir cómo estos elementos van a evolucionar durante todo su ciclo de vida. La forma de organizar las infraestructuras y aplicaciones en un conjunto de servicios interactivos es lo que habitualmente se conoce con el nombre de *Arquitectura Orientada a Servicios (SOA)*[59].

El papel de la Arquitectura Software en el desarrollo de software dirigido por modelos y orientado a servicios

Los párrafos anteriores destacaban la importancia de considerar el aspecto arquitectónico como elemento crucial tanto del paradigma SOC como de la aproximación MDA. Desde el punto de vista de SOA, la arquitectura se concibe como una forma de estructurar, organizar y mostrar el comportamiento y evolución de un sistema orientado a servicios. En el caso de MDA, por el contrario, se entiende como un artefacto contenedor y descriptor de los elementos centrales de un sistema y, por lo tanto, de los elementos que aparecen en los modelos definidos durante el proceso de desarrollo. La Arquitectura Software, entendida como la organización fundamental de un sistema a partir de sus componentes, las relaciones entre ellos y su entorno y los principios que gobiernan su diseño y evolución [208] representa, por consiguiente, **el nexo de unión entre**

las tendencias actuales dirigidas por modelos y las aproximaciones tecnológicas que conforman la base de los sistemas de información actuales.

Los conceptos detrás de la especificación de Arquitecturas Software se alinean perfectamente con el uso de SOA y MDA para el desarrollo de sistemas. Por un lado, la especificación de arquitecturas de servicios en un entorno basado en MDA permite la separación de la estructura de un sistema de otros aspectos del mismo (tales como estrategias de almacenamiento, definición de interfaces o modelado de la funcionalidad concreta del sistema mediante modelos de comportamiento). **Al usar MDA, la especificación de la arquitectura puede ser diseñada sin verse afectada por las restricciones impuestas por las plataformas, estándares o tecnologías** utilizadas para implementar dicha arquitectura. La separación en niveles de abstracción también favorece la aplicación de diferentes estrategias de diseño de servicios o estilos arquitectónicos separadamente, además de facilitar la migración del sistema a una plataforma destino diferente (no orientada a servicios, por ejemplo) cuando se necesite.

Tradicionalmente, en Ingeniería del Software, se le ha otorgado a la arquitectura un papel central en los procesos de desarrollo software. El *Proceso Unificado de Desarrollo* [99], por ejemplo, basa su ciclo de desarrollo en la definición de diversas fases en las que el sistema se construye a través de la definición iterativa de la arquitectura, por lo que se considera que juega un papel central. En un entorno metodológico basado en MDA, el papel otorgado a los modelos arquitectónicos representa también un aspecto esencial. En este caso, ya que la especificación de la arquitectura no representa una parte aislada del sistema, **la influencia de la arquitectura debe propagarse al resto de modelos**. Los elementos especificados en el modelo arquitectónico deciden qué otros modelos deben crearse y qué elementos dentro de esos modelos deben ser incluidos. El contenido de los modelos arquitectónicos, por consiguiente, guía los pasos que deben darse a la hora de construir el sistema. Los procesos y metodologías de desarrollo que siguen esta aproximación se conocen con el nombre de *architecture-centric*.

Conocida la sinergia que puede establecerse entre el paradigma orientado a servicios (SOC/SOA), el desarrollo dirigido por modelos (MDD/MDA) y el papel que juega la arquitectura en el desarrollo de software, esta Tesis Doctoral se presenta el **desarrollo de un marco de trabajo para la especificación de arquitecturas software siguiendo un proceso dirigido por modelos (centrado en la propuesta de MDA) y utilizando los principios definidos por el paradigma SOC**.

Como partes constituyentes de este marco de modelado, denominado *ArchiMeDeS*, se especifican un conjunto de lenguajes específicos de dominio para los niveles PIM y PSM de la arquitectura MDA. La notación utilizada para los mismos se basa en la definición de perfiles UML que contienen un conjunto de estereotipos propios para cada lenguaje. Además, se establecen las reglas de transformación necesarias para, por un lado, obtener los modelos arquitectónicos de nivel PSM a partir de los modelos PIM y, por otro lado, para incorporar información referente a estilos arquitectónicos dentro de los modelos de nivel PIM. Con el fin de dotar al marco de trabajo de un soporte tecnológico, se construye una herramienta de modelado que soporta tanto la edición y validación de los modelos y metamodelos referidos a los DSL especificados como la especificación y ejecución de las transformaciones de modelos definidas. La validación del marco de trabajo descrito se realiza mediante su aplicación a diferentes casos de estudio.

A.2 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 es que *es posible especificar arquitecturas software orientadas a servicios utilizando una aproximación dirigida por modelos en la que la noción de servicio actúe como concepto principal tanto de la arquitectura especificada como del proceso de desarrollo en sí mismo.*

El principal objetivo de la Tesis, derivado directamente de la hipótesis previa, es *especificar un marco para el modelado de arquitecturas software en el que la especificación de la arquitectura se obtiene mediante un proceso dirigido por modelos (basado en la propuesta MDA) y en el que los conceptos del paradigma orientado a servicios son considerados como la base para los elementos contenidos en la arquitectura.*

Este objetivo ha sido desglosado en los siguientes subobjetivos:

Obj. 1.- Análisis y evaluación de trabajos e iniciativas previas relacionadas con los temas en los que se centra el trabajo de Tesis Doctoral. Teniendo en cuenta que la Tesis se apoya en tres áreas de conocimiento claramente diferenciadas en el contexto del desarrollo de software, este objetivo se puede subdividir en los siguientes:

Obj. 1.1. Estudio detallado de las propuestas actuales sobre desarrollo de Arquitecturas orientadas a servicios, poniendo especial interés en la forma en que abordan la perspectiva arquitectónica.

Obj. 1.2. Estudio detallado de de las iniciativas actuales en el ámbito del desarrollo dirigido por modelos centrándose en aquellas propuestas para la especificación de Arquitecturas Software y en aquellas soluciones para el desarrollo orientado a servicios.

Obj. 1.3. Análisis de las características definitorias del paradigma orientado a servicios, incluyendo: composición y coordinación de servicios, estándares relacionados con servicios, restricciones, contratos e interfaces, etc.

Obj. 2.- Especificación de una vista conceptual de arquitecturas software utilizando servicios. Para ello, se tendrá que proporcionar una definición a nivel PIM del correspondiente DSL para Arquitecturas de Servicios. Para alcanzar este objetivo, diversos subobjetivos deben cumplirse:

Obj. 2.1. Definición de un metamodelo de nivel PIM cuyos elementos básicos se refieran a todos los conceptos significativos del paradigma orientado a servicios recogiendo adecuadamente la semántica del DSL a este nivel.

Obj. 2.2. Definición de un metamodelo que permita la especificación de estilos arquitectónicos y/o cualquier otro patrón de diseño habitual basado en servicios.

Obj. 2.3. Definición de la notación sintáctica de los DSLs anteriores en UML (mediante un perfil UML) con el fin de completar el DSL a este nivel.

Obj. 3.- Especificación de los elementos de modelado necesarios para representar las particularidades de las plataformas de ejecución a través de modelos arquitectónicos, esto es, la definición de los DSLs necesarios mara la especificación de arquitecturas de servicios en función de la plataforma escogida. Para ello, los siguientes subobjetivos se han descrito.

Obj. 3.1. Definición de un metamodelo de nivel PSM conteniendo los elementos necesarios para describir una solución software de acuerdo a la plataforma de ejecución destino y la tecnología empleada para su implementación. Este metamodelo servirá como base para la definición de un DSL para arquitecturas software a este nivel

Obj. 3.2. El metamodelo de nivel PSM debe permitir la representación de arquitecturas software referidas, por lo menos, a las siguientes plataformas de implementación: Servicios Web (basados en los estándares del W3C [236][238]) y Servicios Grid (basados en la plataforma *Globus Toolkit* y la arquitectura OGSA [72]).

Obj. 3.3. Definición de la notación sintáctica para los metamodelos previos en UML (mediante un perfil UML) con el fin de completar el DSL para este nivel de abstracción.

Obj. 4.- Especificación de reglas de transformación de modelos.

Obj. 4.1. Definición de reglas de transformación desde modelos PIM a modelos concretos de nivel PSM.

Obj. 4.2. Definición de reglas de transformación que permitan la inclusión de características de estilos arquitectónicos en los modelos de nivel PIM.

Obj. 5.- Construcción de un conjunto de herramientas de modelado que pemita soportar el proceso de modelado de la arquitectura.

Obj. 5.1. Definición de una herramienta que permita la edición de modelos y su posterior verificación conforme a los DSLs definidos.

Obj. 5.2. Inclusión de soporte gráfico para el modelado de los DSLs definidos de acuerdo a la sintaxis concreta definida con anterioridad.

Obj. 5.3. Implementación de una herramienta que permita la ejecución y validación de las reglas de transformación definidas, ya sean PIM-a-PIM o PIM-a-PSM.

Obj. 6.- Validación del marco de especificación de arquitecturas mediante su aplicación a diferentes casos de estudio e implementación de pruebas de concepto sobre casos parciales:

Obj. 6.1. Implementación de pruebas de concepto centradas en la validación del editor gráfico de modelado, los metamodelos especificados y las transformaciones de modelos definidas utilizando el conjunto de herramientas con el fin de comprobar aspectos concretos del marco desarrollado.

Obj. 6.2. Evaluación de los resultados obtenidos mediante la implementación de diferentes casos de studio con el fin de comprobar la validez del marco desarrollado como un todo, la consistencia de los conceptos recogidos en los modelos, las arquitecturas orientadas a servicios especificadas con dichos modelos (según las tecnologías de Servicios Web y Servicios Grid) y las herramientas de modelado creadas.

A.3 Metodología

El método de investigación utilizado en esta Tesis está adaptado del propuesto por Marcos & Marcos [137] para la investigación en el ámbito de la Ingeniería del Software. Este método se basa en el método hipotético-deductivo de

Bunge [32], que se compone de varios pasos, suficientemente generales, para ser aplicados a cualquier tipo de actividad investigadora. Las principales fases del proceso de investigación seguido para completar la Tesis Doctoral actual se muestran en la Figura A.1.

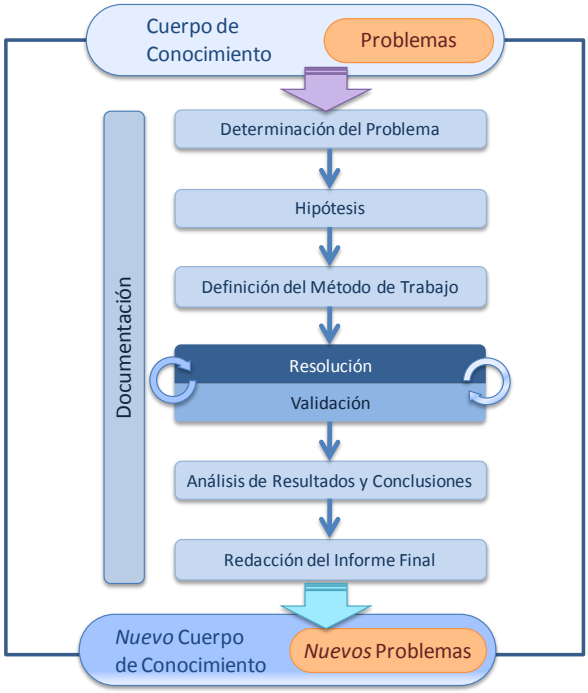

Figure A-1.Esquema del método de investigación.

El '*Cuerpo de Conocimiento*' en el que se enmarca la tesis, recoge las cuestiones establecidas en la sección de Antecedentes de este capítulo. Este elemento actúa de preámbulo para el proceso de investigación y cubre todos los términos que conlleva el manejo de los tres pilares temáticos nombrados anteriormente. Desde este contexto, la identificación de características compartidas que deben resolverse a partir de los principios de MDE, de la Orientación a Servicios y el papel que juega la Arquitectura software conforman la línea de actuación que se debe seguir a la hora de definir con precisión el problema a resolver (fase de 'Determinación del problema') a partir del cual se formula una 'Hipótesis' como punto de partida del trabajo de investigación.

Como puede verse en la Figura A-1, la definición del método de trabajo en si mismo se considera como otro paso dentro del método de investigación. Se trata de una fase necesaria debido a la capacidad del método para adaptarse a diferentes

contextos. Cada proyecto de investigación tiene sus propias características intrínsecas y por lo tanto no hay un método universal que pueda aplicarse a cualquier tipo de investigación. En el contexto de la Tesis actual, la fase correspondiente a la 'Resolución y Validación' es de especial interés ya que es la que representa el núcleo del trabajo de investigación realizado. Más adelante se explicará esta fase con más detalle.

Una vez que se ha llegado a plantear una propuesta a partir de las fases anteriores, es el momento de analizar los resultados obtenidos como consecuencia de la aplicación de dicha propuesta a escenarios concretos (fase de 'Análisis de resultados y conclusiones'). Estas tareas de análisis sirven de base para la extracción de ciertas conclusiones acerca del trabajo realizado. El siguiente paso es recopilar toda la experiencia de investigación adquirida en un informe final.

A pesar de que el paso final del proceso de investigación se puede enmarcar en la tarea de escritura del informe final de Tesis, cualquier actividad de investigación genera un nuevo cuerpo de conocimiento que se crea a través de la incorporación de todos los artefactos de investigación como parte del mismo. De esta nueva situación se derivan nuevos problemas que pueden ser objeto de tareas e iniciativas de investigación futuras.

Fase de 'Resolución y Validación'

El método escogido en esta Tesis Doctoral para la fase de 'Resolución 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 Figura A-2.

Como puede verse, esta fase se subdivide en varias fases o pasos distribuidos en varias iteraciones consecutivas. La primera iteración centra su atención en la definición de la parte del marco propuesto dedicado al modelado de la vista conceptual de una Arquitectura de Software, esto es, al nivel PIM de la propuesta MDA. La segunda iteración, por el contrario se dedica a proveer una vista más cercana al nivel de implementación (PSM) de los modelos arquitectónicos. La tercera iteración de estas tareas se dedica a la definición de las reglas de transformación entre modelos (Transformaciones M2M en la Figura A-2). Aunque estas tres iteraciones se describen aquí como secuenciales, en muchos momentos de la investigación realizada las tareas asociadas se han realizado de forma entrelazada. Además, algunos otros aspectos que son necesarios para completar el marco propuesto en esta Tesis se han realizado durante el proceso de

investigación, como por ejemplo la implementación de la herramienta de modelado asociada al marco de especificación de arquitecturas descrito.

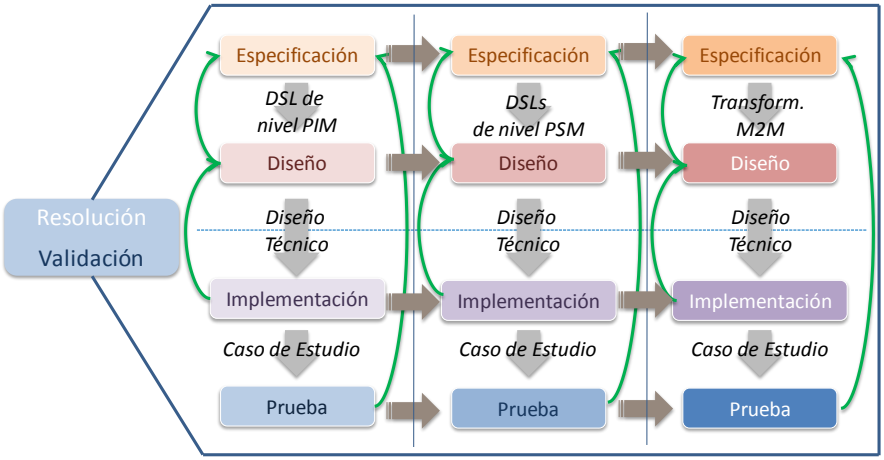

Figure A-2.Vista general de la fase de 'Resolución y Validación'.

Profundizando en cada una de las iteraciones, la tarea de 'Especificación' se centra en la definición de los aspectos y características centrales necesarias para la construcción de los diferentes DSLs para arquitecturas de servicios propuestos y aquellos aspectos teóricos que permiten soportar el marco propuesto mediante servicios y a diferente nivel de abstracción.

Después de una primera especificación de los metamodelos que permiten la especificación de las arquitecturas software, es necesario refiner dichos lenguajes y modelos así como progresar en la validación de los mismos. Esta validación se consigue siguiendo dos líneas de actuación diferentes. Por un lado, se require que el marco descrito esté soportado por una herramienta de modelado gráfico que facilite la definición de modelos arquitectónicos. Esta herramienta sirve tanto como entorno gráfico de modelado como de plataforma de validación los modelos y los metamodelos con los que se supone que son conformes y su eventual transformación a otros modelos. El proceso de diseño y su consiguiente implementación se consideran asimismo como pasos intermedios en cada iteración.

La herramienta en sí misma puede considerarse como una prueba de concepto para los DSL creados ya que permite establecer, ejecutar y comprobar las reglas y restricciones que se han definido para cada lenguaje. Por otra parte, la viabilidad del marco propuesto y su aplicabilidad real se comprueba mediante su utilización en escenarios de casos de estudio del mundo real.

Para concluir, los tres pasos mencionados anteriormente (especificación de los DSLs, diseño de la herramienta y comprobación mediante casos de estudio) no representan un proceso sencillo y directo sino que se realiza de forma iterativa permitiendo que la información fluya entre las tareas que lo componen con el fin de mejorar cada uno de los pasos de investigación.

A.4 Conclusiones

Esta tesis proporciona una serie de contribuciones, no sólo en el ámbito de la investigación planteada como punto de partida (la especificación de *ArchiMeDeS* como marco de modelado de arquitecturas software), sino también en relación con otros aspectos complementarios. Estas contribuciones se resumen a continuación:

Especificación de un marco de trabajo para el desarrollo dirigido por modelos de Arquitecturas Software

La principal contribución de esta Tesis ha sido el desarrollo de *ArchiMeDeS*, un marco de trabajo completo que permite la especificación de arquitecturas cuya principal característica se encuentra en el uso del paradigma SOC como base conceptual de la arquitectura. De las propuestas analizadas en este trabajo doctoral, pocas de ellas aprovechan los beneficios que el uso de servicios puede ofrecer a la hora de especificar arquitecturas software. En esta Tesis se ha definido una arquitectura de modelos para la especificación de arquitecturas software con el fin de reducir el salto existente entre las concepciones de alto nivel de las organizaciones de negocio y sus potenciales implementaciones desde el punto de vista arquitectónico.

Las características de *ArchiMeDeS* contribuyen al establecimiento de una forma diferente de abordar la definición de arquitecturas software así como en el momento de establecer la relación entre el aspecto arquitectónico y otras consideraciones del desarrollo de sistemas software. El uso de modelos para este fin también facilita el entendimiento del nuevo sistema y los pasos y elementos que deben ser definidos como parte de dicho proceso de desarrollo. Esto se consigue a través de la definición de una gramática para arquitecturas concreta en la forma de distintos DSLs a diferentes niveles de abstracción.

Tal y como se ha descrito en esta memoria de Tesis, el uso de MDA como aproximación fundamental a la especificación arquitectónica, con su propuesta de separación en niveles PIM/PSM, ofrece la oportunidad de diversificar la forma de implementar una solución software y permite facilitar potenciales migraciones del sistema o incluso cambiar la plataforma o tecnología de implementación escogida.

El procedimiento inverso (obtener una representación PIM a partir de diferentes modelos PSM), aunque no ha sido considerada como parte de esta Tesis, también sería posible.

Además, el marco desarrollado incluye la posibilidad de definir decisiones y restricciones arquitectónicas mediante estilos arquitectónicos. La novedad frente a otras propuestas radica en que, teniendo en cuenta que la arquitectura es construida con servicios, es posible dar soporte no sólo a estilos arquitectónicos tradicionales sino también a patrones de diseño que han surgido a partir de la utilización de servicios en el ámbito empresarial y que han conllevado la aparición de estrategias de diseño de entornos de servicios ampliamente documentadas [59]. Asimismo, tal y como se describe en la sección dedicada a las transformaciones de modelos arquitectónicos (Sección 3.2.3), *ArchiMeDeS* es un marco de trabajo lo suficientemente general como para soportar otros tipos de transformaciones arquitectónicas como pueden ser la fusión de elementos o la descomposición de los mismos atendiendo a otros criterios o con el fin de representar procesos de coordinación de servicios más complejos.

Soporte para la generación semi-automática de arquitecturas dependientes de la plataforma mediante la utilización de un entorno gráfico de modelado

La creación de un conjunto de herramientas de modelado para la definición de modelos y metamodelos se necesita con el fin de poder representar configuraciones arquitectónicas. Concebido inicialmente como una forma de comprobar que los modelos son conformes a los metamodelos de los que se pueden derivar, el desarrollo de estas herramientas sobrepasó las expectativas iniciales y se convirtió al final en una potente herramienta que no sólo soporta la edición gráfica de los modelos (utilizando un esquema en árbol o basado en UML) sino que también es una herramienta que permite comprobar la corrección de los modelos, la coherencia sintáctica de los metamodelos y, eventualmente, la validación de los modelos obtenidos como resultado de la ejecución de transformaciones entre modelos.

La plataforma sobre la que se asienta la herramienta, Eclipse, demostró ser una elección adecuada para alcanzar los objetivos marcados en esta Tesis. Eclipse ha demostrado su utilidad a la hora de abordar tareas de modelado y metamodelado (mediante el uso de extensiones para Eclipse tales como EMF o GMF) pero también, y quizá sea un detalle más importante, ha permitido incorporar la definición de reglas de transformación (expresadas con ATL) dentro del propio entorno de modelado, de tal forma que se han podido (semi-)automatizar los procesos de transformación definidos. Esta circunstancia fue

posible gracias a la capacidad de la plataforma de ser fácilmente extendida con nuevas funcionalidades añadiendo, simplemente, una serie de módulos o plug-ins a la plataforma base de Eclipse.

La transformación de modelos ha sido posible mediante la definición de procesos de transformación que abarcan desde simples transformaciones modelo-a-modelo (en el caso de obtener el modelo PSM de la arquitectura de su correspondiente modelo PIM) hasta la posibilidad de definir un proceso de fusión completo que permita anotar los modelos PIM con información proveniente de modelos de estilos arquitectónicos.

Definición de un estudio del estado del arte en el ámbito de tres disciplinas de ingeniería con una importancia creciente en la actualidad

La tarea de aunar las características de los tres pilares en los que se asienta la Tesis presentada ha requerido partir de diferentes premisas: por un lado, los **criterios** seleccionados necesitaban ser lo representativos en el contexto de la especificación de arquitecturas; por ejemplo, en referencia al rol otorgado a la Arquitectura en un proceso de desarrollo (en el contexto de MDD) o el paradigma base utilizado para describir la arquitectura (en el contexto de SOC). Por otro lado, los criterios utilizados debían ser lo suficientemente significativos para poder clasificar las diferentes aproximaciones que las iniciativas de investigación. Tal es el caso, por ejemplo, de la identificación del nivel de abstracción al cual se considera la especificación de la arquitectura o la aproximación de modelado seguida por iniciativas dirigidas por modelos. Además, es importante remarcar que la investigación llevada a cabo durante la realización de esta Tesis hereda las cuestiones y soluciones derivadas del marco metodológico en el que se enmarca *ArchiMeDeS*. Los trabajos previos de investigación resultaron en la identificación de aspectos que necesitaban ser resueltos mediante MDE, SOA o especificaciones arquitectónicas. La experiencia previa en estos ámbitos también condicionó la selección de los criterios realizada.

El resultado ha sido, por un lado, la definición de un conjunto representativo de criterios que permite la identificación de las diferentes propuestas que están relacionadas, de alguna forma, con los temas contenidos en la Tesis; y, por otro lado, la realización de un análisis del estado del arte al respecto incluyendo aquellas iniciativas más relevantes en este contexto.

En resumen, la realización del estado del arte no sólo representa uno de los primeros intentos de identificar las características deseables de la propuesta condensada en *ArchiMeDeS*. Además, este estudio ha permitido reconocer la necesidad de incorporar, conjuntamente, las aproximaciones de modelado actuales

(como MDA), los aspectos claves del software como forma de guiar los procesos de desarrollo (como la especificación de la Arquitectura) y los paradigmas de computación que más se están extendiendo en la actualidad (como es el caso de SOC) con el fin de obtener éxito en el ámbito de la Ingeniería del Software.

Implementación de un caso de estudio real y plenamente funcional derivado de la investigación en neuromedicina

Finalmente, la última contribución se refiere al principal caso de estudio que se ha utilizado para validar la propuesta de la Tesis. El sistema *GESiMED* representa un ejemplo real en el que los aspectos primordiales descritos en la Tesis pueden utilizarse obteniendo resultados altamente positivos. De esta forma, *ArchiMeDeS* se utiliza como forma de ofrecer una respuesta tecnológica a las necesidades de negocio detectadas en un ámbito concreto como es el de la investigación con neuroimágenes, tal y como lo demuestran algunas publicaciones al respecto ([96]).

Mediante la utilización de una aproximación dirigida por modelos para la especificación de la arquitectura de *GESiMED* se facilita la posible migración de dicho sistema de una plataforma a otra. Esta circunstancia es viable debido a la posibilidad de definir la arquitectura del sistema a con un nivel de abstracción suficiente (PIM) de tal forma que puede ser adaptado a diferentes plataformas de implementación (TDM). Además, las capacidades provistas por ArchiMeDeS para la inclusión de estilos arquitectónicos en el modelo arquitectónico, facilita la adaptación del sistema a las restricciones de negocio concretas que suelen darse en el contexto de la investigación en neurociencias, por ejemplo, debido a requisitos de carga del sistema.

Appendix B:
Bibliography and Online Resources

Bibliography

[1] Abeti, L., Ciancarini, P. & Moretti, R. (2006). A Service Oriented Approach to Model a Grid System for the Civil Protection. In *Proceedings of the International Workshop on complex Network and Infrastructure Protection*. Rome (Italy).

[2] Acuña, C. J. (2007). *PISA – Arquitectura de integración de portales Web: un enfoque dirigido por modelos y basado en servicios Web semánticos*. PhD Thesis. Rey Juan Carlos University.

[3] Acuña, C., Marcos, E., de Castro, V., & Hernández, J.A. (2004). A Web Information System for Medical Image Management. In Barreiro, JM, Martín-Sánchez, F., Maojo, V., Sanz, F. (eds.), *International Symposium on Biological and Medical Data Analysis* (pp.49-59). LNCS 3337. Springer-Verlag.

[4] Aiello, M. & Dustdar, S. (2006). Service Oriented Computing: Service Foundations. In *Proceedings of the Dagstuhl Seminar 2006*. Service Oriented Computing, vol. 05462. Germany.

[5] Akerman, A., Tyree, J., & Coglianese, L. (2004). An Architecture Process for System Evolution. *Enterprise Architect Magazine*. vol. 2 (1). www.ftponline.com /ea/magazine/spring/features/aakerman

[6] Allen, R. (1997). *A Formal Approach to Software Architecture.* Ph.D. Thesis. Carnegie Mellon University. CMU Technical Report CMU-CS-97-144.

[7] Allilaire, F., Idrissi, T. (2004). *ADT: Eclipse Development Tools for ATL*. EWMDA-2. Kent. Retrieved from: http://www.cs.kent.ac.uk/projects/kmf/ mdaworkshop/

[8] Alti, A., Khammaci, T., Smeda, A., Bennouar, D. (2007). Integrating Software Architecture Concepts into the MDA Platform. In *Proceedings of ICSOFT'07* (SE) (pp. 144-149).

[9] Amir, R. and Zeid, A. (2004). An UML Profile for Service Oriented Architectures. *Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA'04* (pp. 192–193)

[10] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S. (2003). *Business Process Execution Language for Web Services* (BPEL), Version 1.1 Specification. BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems.

[11] Arsanjani A. (2008). Toward a pattern language for Service-Oriented Architecture and Integration. *IBM DeveloperWorks site*. Retrieved from www.ibm.com/developerworks/webservices/library/ws-soa-soi/.

[12] Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S. and Holley, K. (2008). SOMA: A method for developing service-oriented solutions. *Published online August 6, 2008*. IBM Systems Journal. Vol. 47 (3).

[13] ATOS (2007). *SeCSE methodology*, Version 3, IST European Integrated Project SeCSE, 6th Framework Programme, Deliverable A5.D4.2, available online at http://secse.eng.it/wp-content/uploads/2007/08/a5-d4-2-secse-ethodologyversion-3.pdf

[14] Autili, M., Cortellessa, V., Di Marco, M. and Inverardi, P. (2006). A Conceptual Model for Adaptable Context-aware Services. *In Proceedings of Web Services Modeling and Testing (WS-MaTe 2006)*, Palermo, Sicily, Italy.

[15] Avison, D., Lan, F., Myers, M. y Nielsen, A. (1999). Action Research. *Communications of the ACM*, 42(1), pp. 94-97.

[16] Baresi, L., Heckel, R., Thone, S., and Varro, D. (2003). Modeling and validation of service-oriented architectures: Application vs. style. *In Proc. ESEC/FSE 2003*, Helsinki, Finland..

[17] Barros, A., Decker, G., Dumas, M. (2006). Multi-staged and Multi-viewpoint Service Choreography Modelling. *Technical Report 4668*, Queensland Univ. of Technology.

[18] Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture in Practice*. Addison-Wesley, 2nd edition.

[19] Berners-Lee, T., Fielding, R., Masinter, L. (1998). *Uniform Resource Identifiers (URI): Generic Syntax, IETF RFC 2396*. http://www.ietf.org/rfc/rfc2396.txt

[20] Bernstein, P A. (2003). Applying Model Management to Classical Meta Data Problems. *In proc. of 1st Biennial Conference on Innovative Data Systems Research*, CA, USA.

[21] Bézivin, J. (2004). In search of a Basic Principle for Model Driven Engineering. *Novatica/Upgrade*, Vol. 2, pp. 21-24.

[22] Bezivin,, J. and Jouault, F. (2005). Using ATL for Checking Models. *In Proc. of GraMoT 2005*. LNCS Vol. 152, pp. 69-81.

[23] Bézivin, J., Bouzitouna, S., Del Fabro, M., Gervais, M. P., Jouault, F., Kolovos, D., et al. (2006). A Canonical Scheme for Model Composition. *Paper presented at the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA'06)*, Bilbao, Spain.

[24] Bézivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., & Lindow, A. (2006). Model Transformations? Transformation Models! In *Proc. of the 9th International Conference on Model Driven Engineering Languages and Systems*, MoDELS 2006, Genève, Italy.

[25] Blanc, X., Gervais, M.-P., & Sriplakich, P. (2005). Model Bus: Towards the Interoperability of Modelling Tools. *Paper presented at the European MDA Workshop: Foundations and Applications, MDAFA 2004*, Linköping, Sweden

[26] Bollati V. A. (2011). *MeTAGeM: un Entorno de Desarrollo de Transformaciones de Modelos Dirigido por Modelos*. PhD Thesis. Rey Juan Carlos University, February 2011.

[27] Bravetti, M., Zavattaro, G. Service Discovery based on Behavioural Contracts. In *International School on Formal Methods for the Design of Computer,*

Communication and Software Systems: Web Services, SFM-09:WS. Revised Lectures, Bertinoro, Italy, June 1-6, 2009.

[28] Broy, M. (2004). Model Driven, Architecture-Centric Modeling in Software Development. *In Proceedings of 9th Intl. Conf. in Engineering Complex Computer Systems (ICECCS'04)*, pp. 3-12, IEEE Computer Society.

[29] Budinsky, F. et al. (2008). *Eclipse Modeling Framework. 2nd Edition*. Addison-Wesley Professional.

[30] Burge, J.E., Carroll, J.M., McCall R., Mistrík I. (2008). *Rationale-Based Software Engineering*. Springer-Verlag, Heidelberg.

[31] Buttner, F., & Gogolla, M. (2004). Realizing UML Metamodel Transformations with AGG, *In Proceedings of the Workshop on Graph Transformation and Visual Modelling Techniques (GT-VMT 2004)*, Barcelona, Spain

[32] Bunge, M. (1979). *La Investigación Científica*. Barcelona: Ariel.

[33] Cabrera, F., Copeland, G., Freund, T., Klein, J., Langworhty, D., Orchard, D., Shewchuk, J., Storey, T. (2002). *Web Services Coordination (WS-Coordination).* BEA Systems, International Business Machines Corporation, Microsoft Corporation. http://www-106.ibm.com/developerworks/library/ws-coor/

[34] Cabrera F., Copeland G., Freund T., Klein J., Langworhty D., Orchard D., Shewchuk J., Storey T. (2004). *Web Services Transaction (WS-Transaction).* IBM, BEA Systems, Microsoft, Arjuna, Hitachi, IONA. Retrieved from: www.ibm.com/developerworks/library/specification/ws-tx/

[35] Cáceres P., Marcos E. and Vela B. (2007). A MDA-Based Approach for Web Information System Development. *Workshop in Software Model Engineering.* http://www.metamodel.com/wisme-2003/

[36] Cáceres, P., de Castro, V., Vara, J.M., Marcos, E. (2006). Model Transformations for Hypertext Modeling on Web Information Systems. *In Proceedings of the 21st Annual ACM Symposium on Applied Computing (SAC 2006) - Track on Model Transformation*, Ed.: The Association for Computing Machinery, Inc. pp. 1232-1239.

[37] Carnegie-Mellon University, Software Engineering Institute. *The Capability Maturity Model: Guidelines for Improving the Software Process*, SEI Series in Software Engineering, Addison Wesley, 1995

[38] Cicchetti, A., Ruscio, D. D., Eramo, R., & Pierantonio, A. (2008). Automating Co-evolution in Model-Driven Engineering. *Paper presented at the 12th International IEEE Enterprise Distributed Object Computing Conference - EDOC 2008*, München, Germany.

[39] Clements P. (1996). A Survey of Architecture Description Languages, in *Proceedings of the 8th International Workshop on Software Specification and Design*, pp. 16-25. Germany, Mar. 1996. Schloss Velen, Germany, 22-23.

[40] Clements P., F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord and J. Stafford. (2002). *Documenting Software Architectures, Views and Beyond.* Pearson Education, Inc., Addison-Wesley, Bosto.

[41] Colombo, M., Di Nitto, E., Di Penta, M., Distante, D. and Zuccalà, M. (2005). Speaking a Common Language: A Conceptual Model for Describing Service-Oriented Systems. *In 3rd International Conference on Service Oriented Computing (ICSOC'05)*, Amsterdam, the Netherlands.

[42] Cox, W., Cabrera, F., Copeland, G., Freund, T., Klein, J., Storey, T., Thatte, S. (2002). *Web Services Transaction (WS-Transaction)*. BEA Systems, International Business Machines Corporation, Microsoft Corporation. Available from: http://dev 2dev.bea.com/pub/a/2004/01/ws-transaction.html.

[43] Cuesta, C. E., de la Fuente, P., Barrio-Solórzano, M., Beato, M. E. (2002). Introducing Reflection in Architecture Description Languages. *In Proc. of WICSA 2002*, pp. 143-156.

[44] Cuesta C. E., Romay M. P., de la Fuente P., Barrio-Solórzano M., Younessi H.: Coordination in Architectural Connection. Reflective and Aspectual Introduction. *L'OBJET* 12(1): 127-151 (2006)

[45] Dardenne, A., van Lamsweerde, A., Fickas, S. (2003). Goal-directed Requirements Acquisition. *Science of Computer Programming* 20(1-2), pp. 3–50.

[46] 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*. PhD. Thesis, Rey Juan Carlos University, 2007.

[47] De Castro, V., López-Sanz, M., Marcos, E. (2006). Business Process Development based on Web Services: A Web Information System for Medical Images Management and Processing. *Proceedings of IEEE International Conference on Web Services.* IEEE Computer Society. Ed.: F. Leymann, L.J. Zhang, pp. 807-814.

[48] De Castro, V., Marcos, E., & Cáceres, P. (2004). A User Service Oriented Method to Model Web Information Systems. *In Proc. of WISE'04,* Vol. 3306, pp. 41-52. Springer-Verlag.

[49] De Castro, V., Marcos, E. and Wieringa, R. (2009). Towards a Service-oriented MDA-Based Approach to the Alignment of Business Processes with it Systems: From the Business Model to a WS Composition Model. *Int. Journal on Cooperative Information Systems*. 18(2): 225-260.

[50] De Castro, V., Vara, J. M., Herrmann, E. & Marcos, E. (2008). A Model Driven Approach for the Alignment of Business and Information Systems Models. *Paper presented at the Proceedings of the 2008 Mexican International Conference on Computer Science, ENC'08*. Mexicali, Baja California, Mexico

[51] De Roure, D. et al (2003). The Semantic Grid: A Future e-Science Infrastructure, *International Journal of Concurrency and Computation: Practice and Experience.* Vol. 5.

[52] Di Ruscio, D., Malavolta, I., Muccini, H., Pelliccione, P., and Pierantonio, A. (2010). Developing next generation ADLs through MDE techniques. *In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010)*. Vol. 1. pp 85-94. Cape Town, South Africa.

[53] Di Ruscio, D., Muccini, H., Pelliccione, P., Pierantonio, A. (2006). Towards Weaving Software Architecture Models. *In Proc. MBD/MOMPES Workhops within the IEEE ECBS 2006*, pp. 103-112

[54] Didonet, M., Bézivin, J. and Valduriez, P. (2006) Weaving Models with the Eclipse AMW plug-in. *Eclipse Modeling Symposium, Eclipse Summit Europe*, Germany.

[55] Didonet, M. (2007). *Metadata management using model weaving and model transformation.* Ph.D. Thesis. University of Nantes.

[56] Dijkman, R. M. and Dumas, M. (2004). Service-Oriented Design: A Multi-Viewpoint Approach. *Int. J. Cooperative Inf. Syst*. 13(4): 337-368.

[57] Elleuch, N., Khalfallah, A., Ben Ahmed, S. (2007). ArchMDE: Approach for the Development of Embedded Real Time Systems. *In Proc of Ada-Europe 2007, pp* 142-154.

[58] Emmerich, W., Butchart, B., Chen, L., Wasserman, B., and Price, S. L. (2005) *Grid-Service Orchestration Using Business Process Execution Language (BPEL)*, University College, London, CS Research Note RN/05/07.

[59] Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River: Prentice Hall PTR.

[60] Erl, T. (2008). *SOA Principles of Service Design*. Prentice Hall. ISBN 0-13-234482-3

[61] Erl, T. (2009). *SOA Design Patterns*. Prentice Hall. ISBN 0-13-613516-1

[62] *European Commission – Research*. The Sixth Framework Programme (2002-2006). Available at: http://ec.europa.eu/research/fp6/index_en.cfm.

[63] *European Commission – Research*. The Seventh Framework Programme (2007-2013). Available at: http://ec.europa.eu/research/fp7/index_en.cfm.

[64] Farenhorst, R., Lago, P., and van Vliet, H. (2007). EAGLE: Effective Tool Support for Sharing Architectural Knowledge. *International Journal of Cooperative Information Systems* (IJCIS), 16(3/4):413--437

[65] Favre, J. (2004). Towards a Basic Theory to Model Model Driven Engineering. *Paper presented at the Workshop on Software Model Engineering, WISME 2004, joint event with UML2004*, Lisbon, Portugal.

[66] Feiler, P. H., Lewis, B. A. and Vestal, S. (2006). The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pp. 1206-1211

[67] Fiadeiro, J.L., Lopes, A., Bocchi, L. (2008). An Abstract Model of Service Discovery and Binding.

[68] Fiadeiro, J.L., Lopes, A., Bocchi, L. (2006). *The SENSORIA Reference Modelling Language: Primitives for Service Description*. Available at www.sensoria-ist.edu.

[69] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD Thesis. University of California-Irvine.

[70] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T. (1999). *Hypertext Transfer Protocol -- HTTP/1.1*; RFC 2616.

[71] Foster, H., Uchitel, S., Magee, J., Kramer, J. (2006). WS-Engineer: A Tool for Model-Based Verification of Web Service Compositions and Choreography, *IEEE International Conference on Software Engineering (ICSE 2006)*, Shanghai, China.

[72] Foster. I. Globus Toolkit Version 4: Software for Service-Oriented Systems (2003). *IFIP International Conference on Network and Parallel Computing*, Springer-Verlag LNCS 3779, pp 2-13.

[73] Foster, I. and Kesselman, C. (1999). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann. ISBN 1558604758

[74] Foster, I., Frey, J., Graham, S., et al., (2004). Modeling Stateful Resource with Web Services.

[75] Foster, I., Kesselman, C. and Tuecke, S. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3).

[76] Foster, I., Kesselman, C., Nick, J. and Tuecke, S. (2002). *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. Open Grid Service Infrastructure WG, Global Grid Forum.

[77] Fowler, M. (2005). *Language Workbenches and Model-Driven Architecture*. Retrieved from http://martinfowler.com/articles/mdaLanguageWorkbench.html

[78] Frankel, D. et al. (2003) *The Zachman Framework and the OMG's Model Driven Architecture White paper*. Business Process Trends.

[79] Frankel, D. (2002). *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley & Sons, New York, USA.

[80] French, W.L. and Bell, C.H. Jr. (1996). *Desarrollo organizacional (quinta edición)*. Prentice-Hall, Naucalpán de Juárez, México.

[81] Fujitsu-Arjuna. (2003). *WS-CF: WS-Coordination Framework*, Oracle IONA, Sun.

[82] Gannon, P. and Bratt, S. *Memorandum of Understanding between OASIS and W3C*. Document available at http://www.w3.org/Submission/2006/01/w3c-oasis-cgm-final-051215.pdf. Dec 2005.

[83] Garlan, D., Monroe, R., and Wile, D. (1997). ACME: An Architecture Description Interchange Language. In *Proc. Conf. Centre for Advanced Studies on Collaborative Research, CASCON'97*. pp 169–183. Toronto, Ontario, Canada.

[84] Gerber, A., Lawley, M., Raymond, K., Steel, J., Wood, A. (2002). Transformation: The Missing Link of MDA. *In proc of Intl. Conf. on Graph Transformation 2002*. pp 90-105.

[85] *The Global Grid Forum*: http://www.gridforum.org/

[86] Gordijn, J. and Akkermans, J. M. (2003). Value based requirements engineering: exploring innovative e-commerce idea. Requirements Engineering Journal 8(2), pp. 114 -134. Springer-Verlag.

[87] Gomaa, H. (2005). Architecture-Centric Evolution in Software Product Lines. *In Proc. of ECOOP'2005 Workshop on Architecture-Centric Evolution (ACE'2005), Glasgow*.

[88] Gómez, J., & Cachero, C. (2003). OO-H Method: extending UML to model web interfaces. *In Information Modeling for Internet Applications* (pp. 144-173): IGI Publishing.

[89] Gönczy, L, Kovács, M, Varró, D. (2007). Modeling and Verification of Reliable Messaging by Graph Transformation Systems. *Electr. Notes Theor. Comput. Sci*. 175(4): 37-50.

[90] Gordijn, J., Akkermans, H. (2003). Value based requirements engineering: exploring innovative e-commerce idea. *Requirements Engineering Journal* Vol. 8 (2), pp. 114–134.

[91] Graham, S., Karmarkar, A., Mischkinsky, J., Robinson, I., Sedukhin, I. (2005). Web Services Resource Framework 1.2. OASIS WSRF-TC.

[92] Greenfield, J., Short, K., Cook, S., Kent, S. (2004). *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. John Wiley & Sons.

[93] Gronback, R. C. (2009). *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.* Addison-Wesley Professional.

[94] Hailpern, B., & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal*, 45(3), 451-461

[95] Heckel, R., Lohmann, M., Thöne, S. (2003). Towards a UML Profile for Service-Oriented Architectures. *In Proc. of Workshop on Model Driven Architecture: Foundations and Applications (MDAFA '03)*, University of Twente, Enschede.

[96] Hernández, J.A., Acuña, C., de Castro, V., Marcos, E., López, M., Malpica, N. A. (2006). WEB-PACS for Multi-center Clinical Trials. *IEEE Transactions on Information Technology in Biomedicine*. Vol. 11 (1). pp. 87-93.

[97] IEEE AWG. (2000). *IEEE RP-1471-2000: Recommended Practice for Architectural Description for Software-Intensive Systems*. IEEE Computer Society Press.

[98] Inverardi, P., Muccini, H. and Pelliccione, P. (2005). DUALLY: Putting in Synergy UML 2.0 and ADLs. *In 5th IEEE/IFIP Working Conference on Software Architecture (WICSA 2005).* Pittsburgh, PA, 6-9.

[99] ISO (International Standards Organization for Standardization) & IEC (International Electrotechnical Commission) (2003). *ISO/IEC 9075:2003 Information technology – Database languages – SQL:2003*.

[100] Ivers, J., Clements, P., Garlan, D., Nord, R., Schmerl, B. & Oviedo, J. (2004). *Documenting Component and Connector Views with UML 2.0. Technical Report CMU/SEI-2004-TR-008*, Sofware Engineering Institute, Carnegie Mellon University. Available: http://www.sei.cmu.edu/pub/documents/ 04tr008.pdf.

[101] Jacobson, I., Booch, G., Rumbaugh, J. (1999). *The Unified Software Development Process*, Addison-Wesley, Reading, Mass. ISBN: 0-201-57169-2

[102] Jia, X., Ying, S., Cao, H. and Xie, D. (2007). A New Architecture Description Language for Service-Oriented Architecture. *In proc of GCC 2007*, pp. 96-103

[103] Johnston, S. (2005). UML profile for software services. *IBM DeveloperWorks Site*, http://www- 128.ibm.com/developerworks/rational/ library/05/419_soa/

[104] Jouault, F. and Kurtev, I. (2005). Transforming Models with ATL. Model Transformations in Practice. *Workshop at MoDELS Conference*, Montego Bay.

[105] Katz, S. (1993). A Superimposition Control Construct for Distributed Systems. *ACM Trans. Program. Lang. Syst*. 15(2): 337-356.

[106] Kelly, S. (2005). XMI, MOF and MetaEdit+. Retrieved from: http://www.metacase.com/blogs/stevek/

[107] Kiczales, G.; Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J-M. and Irwin J. (1997). Aspect-Oriented Programming. *Proceedings of the European Conference on Object-Oriented Programming*, vol.1241. pp. 220–242.

[108] Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley.

[109] Kloppmann M. et al. (2004). WS-BPEL Extension for People – BPEL4People, International Business Machines Corporation, SAP AG. Available from: ftp://www6.software.ibm.com/software/developer/library/ws-bpel4people.pdf.

[110] Knapp, A., Koch, N., Moser, F., and Zhang, G. (2003). ArgoUWE: A CASE Tool for Web Applications. *In Proc. 1st Int. Wsh. Engineering Methods to Support Information Systems Evolution (EMSISE'03)*, Geneve, 14 pages.

[111] Koch, N. (2001). *Software Engineering for Adaptative Hypermedia Applications*. PhD Thesis, FAST Reihe Softwaretechnik Vol(12), Uni-Druck Publishing Company, Munich. Germany.

[112] Koch, N., Mayer, P., Heckel, R., Gönczy, L., Montangero, C. (2007). *UML for Service-Oriented Systems.* SENSORIA D1.4a. Available at www.pst.ifi.lmu.de /projekte/Sensoria/del_24/D1.4.a.pdf.

[113] Kozlenkov, A., Fasoulas, V., Sanchez-Cid, F., Spanoudakis, G., Zisman, A. (2006). A Framework for Architecture-driven Service Discovery. *The 2006 International Workshop on Service Oriented Software Engineering (IW-SOSE' 06)*. Shanghai (China).

[114] Krafzig, D.; Banke K., Slama D. *Enterprise SOA Service Oriented Architecture Best Practices*. Upper Saddle River: Prentice Hall PTR. 2004

[115] Kristensen, B. B. (1996). *Object-Oriented Modeling with Roles*. *In Proc. of Object-Oriented Information Systems (OOIS'95)*, pp. 57-71. Springer-Verlag.

[116] Krüger, I. H., Mathew, R. (2004). Systematic Development and Exploration of Service-Oriented Software Architectures. *In Proc. of the 4th Working IEEE/IFIP Conference on Software Architecture (WICSA-04)*, pp. 177-187. Oslo, Norway. IEEE/IFIP.

[117] Kulkarni, V. & Reddy, S. (2003). Separation of Concerns in Model-Driven Development. *IEEE Software*, 20(5), 64-69.

[118] Kurtev, I. (2005). *Adaptability of model transformations.* PhD. Thesis. University of Twente, Enschede. Retrieved from http://purl.org/utwente/50761.

[119] LaLonde, W. and Pugh, J. (1991). Subclassing neq Subtyping neq Is-a. *Journal of Object-Oriented Programming*, 3(5):57–62.

[120] Lohmann, M., Mariani, L., Heckel, R. A Model-Driven Approach to Discovery, Testing and Monitoring of Web Services. *In Proc. of Test and Analysis of Web Services 2007*, pp. 173-204.

[121] López, M., Bosque, JL, de Castro. V., Marcos, E. (2007). A Comparative Study between Web Service and Grid Service Developments in a MDA Framework. *In Proc. of ICEIS 2007*, pp. 114-121.

[122] López-Sanz, M., Acuña, C. J., Cuesta, C. E. and Marcos, E. (2007). Modelling of Service-Oriented Architectures with UML. *Proceedings of the 6th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA'07), 18th International Conference on Concurrency Theory (CONCUR 2007)*, pp. 21-36.

[123] López-Sanz, M., Acuña, C. J., Cuesta, C. E. and Marcos, E. (2007). UML Profile for the Platform Independent Modelling of Service-Oriented Architectures. *Proceedings of the 1st European Conference on Software Architecture (ECSA 2007)*, LNCS 4758, Eds. F. Oquendo, pp. 304-307. ISBN: 978-3-540-75131-1.

[124] López-Sanz, M., Qayyum, Z., Cuesta, C. E., Marcos, E., Oquendo, F (2008). Representing Service-Oriented Architectural Models using π-ADL. Emerging Research Paper. *Proceedings of the 2nd European Conference on Software Architecture (ECSA'08)*, LNCS 5292, pp.273-280. Eds. R. Morrison, D. Balasubramaniam, K. Falkner. ISBN: 978-3-540-88029-5. ISSN: 0302-9743.

[125] López-Sanz, M., Acuña, C. J., Cuesta, C. E. and Marcos, E. (2008). Defining Service-Oriented Software Architecture Models for a MDA-based Development Process at the PIM-level. *In Proc. of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*. Vancouver, BC, Canada.

[126] López-Sanz, M., Vara J. M., Marcos E., Cuesta C. E. (2009). A Model-Driven Approach to Weave Architectural Styles into Service-Oriented Architectures. *Proceedings of the First International Workshop on Model-Driven Service Engineering and Data Quality and Security (MOSE+DQS'09)*, Hong Kong, China. D. Cheung, Il-Yeong Song, W. Chu, X. Hu, J. Lin, J.Li and Z. Peng. ISBN: 978-1-60558-816-2.

[127] Lublinsky, B. (2007). Defining SOA as an architectural style: Align your business model with technology. *IBM DeveloperWorks site*. http://www-128.ibm.com/developerworks/webservices/library/ar-soastyle/index.html.

[128] Luckham, D. C., Kenney, J. J., Augustin, L. M., et al. (1995). Specification and Analysis of System Architecture Using Rapide. *IEEE Trans. Software Eng*. 21(6): 576.

[129] Magee, J. and Kramer, J. (1996). Dynamic Structure in Software Architectures. *In Proc. of ACM/SIGSOFT FSE*. San Francisco, SEN, Vol. 21(6), pp. 3-14.

[130] Malavolta, I, Muccini, H, Pelliccione P. (2008). DUALLY: A framework for Architectural Languages and Tools Interoperability. *In Proceedings of ASE 2008*. pp. 483-484.

[131] Mandel, L. (2009). Describe REST Web services with WSDL 2.0: A how-to guide. *IBM DeveloperWorks*. http://www.ibm.com/developerworks/webservices/library/ws-restwsdl/

[132] Manset, D., Verjus, H., McClatchey, R., Oquendo, F. (2006). A Formal Architecture-Centric Model-Driven Approach For The Automatic Generation Of Grid Applications. In *Proceedings of the 8th International Conference on Enterprise Information Systems*, Paphos (Cyprus), 23 - 27.

[133] Manset, D., Verjus, H., McClatchey, R., Oquendo, F. (2005). A Model-Driven Approach for Grid Services Engineering, *18th Int. Conf. of Software & System*, Vol. 1, pp. 135-142. Paris, France.

[134] Mansurov, N. and Campara D (2003). Extracting High-Level Architecture From Existing Code with Summary Models. In *Proc. IASTED Conf. On Applied Informatics*. Innsbruck, Austria.

[135] Mansurov, N and Campara, D. (2004). Managed Architecture of Existing Code as a Practical Transition Towards MDA. *UML Satellite Activities 2004:* 219-233.

[136] Marcos, E., Acuña, C. J., Cuesta, C. E. (2006). Integrating Software Architecture into a MDA Framework. *In proc. of EWSA 2006*, pp: 127-143. Nantes, France.

[137] Marcos, E. and Marcos, A. (1998). An Aristotelian Approach to the Methodological Research: a Method for Data Models Construction. In*: Information Systems- The Next Generation*. Ed. L. Brooks y C. Kimble. Mc Graw-Hill, pp. 532-543.

[138] Marjan, M., Jan, H. & Anthony, M. S. (2005). When and how to develop domain-specific languages. *ACM Computer Surveys*, 37(4), 316-344.

[139] Matinlassi, M. and Kalaoja, J. (2002). Requirements for Service Architecture Modeling, in *Workshop of Software Modeling Engineering of UML 2002*. Dresden, Germany.

[140] Mattsson, A., Lundell, B., Lings, B., and Fitzgerald, B. (2009). Linking model-driven development and software architecture: A case study. *IEEE Transactions on Software Engineering*, vol. 35 (1), pp. 83-93. Available from: http://dx.doi.org/10.1109/TSE.2008.87

[141] Mayer, P, Baumeister, H. (2007). *SENSORIA Project. Deliverable D7.4b: Report on the Sensoria CASE Tool. Description and Evaluation*.

[142] Mayer, P., Schroeder, A., Koch, N. (2008). UML4SOA: Model-Driven Service Orchestration. *In proc. of. 12th Int. Enterprise Computing Conf. IEEE*, Los Alamitos.

[143] Mazón, J.-N., & Trujillo, J. (2008). An MDA approach for the development of data warehouses. *Decission Support Systems*, 45(1), 41-58.

[144] McTaggart, R. (1991). Principles of Participatory Action Research. *Adult Education Quarterly*, 41(3).

[145] Medvidovic, N., Rosenblum, D., Redmiles, D. and Robbins, J. (2002). Modelling Software Architectures in the Unified Modeling Language, *ACM Transactions on Software Engineering and Methodology*, vol. 11(1), pp. 2-57.

[146] Medvidovic, N., Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93.

[147] Medvidovic, N, Taylor, R. N. and Whitehead E. J., Jr. (1996). Formal Modeling of Software Architectures at Multiple Levels of Abstraction. *In Proceedings of California Symposium 1996*, pp. 28-40.

[148] Mens, T. and Van Gorp P. (2006). A taxonomy of model transformation. *Electr. Notes Theor. Comput. Sci.*, 152:125–142.

[149] Michlmayr, A. Rosenberg, F., Platzer, C., Treiber, M., Dustdar, S. (2007). Towards Recovering the Broken SOA Triangle - A Software Engineering Perspective, *In Proceedings of the 2nd International Workshop on Service-oriented Software Engineering (IW-SOSWE'07)*, Dubrovnik, Croatia, ACM Press.

[150] Mikkonen, T., Pitkänen, R., and Pussinen, M. (2004). On the Role of Architectural Style in Model Driven Development. *In proc of EWSA'04*, pp. 74-87, LNCS, 3047.

[151] Miller, J., Mukerji, J. (2001). *MDA Guide Version 1.0*, OMG Document - omg/2003-05-01.

[152] Milner, R. (1993). The Polyadic π-Calculus: A Tutorial. Logic and Algebra of Specification, Springer-Verlag,

[153] Mitra, S., Kumar, R. and Basu, S. (2007) Automated Choreographer Synthesis for Web Services Composition Using I/O Automata. *In Proc. of ICWS'07.*

[154] Mizuta, S. and Huang, R. (2005), Automation of Grid Service Code Generation with AndroMDA for GT3. *In IEEE CS Proceeding of the 1st International Workshop on Information Networking and Application (INA'05)*, pp. 417-420, Taiwan.

[155] Molina-Espinosa, J. M., Fanchon, J., Drira, K. (2003). A Logical Model for Coordination Rule Classes in Collaborative Sessions. *In Proc. of WETICE 2003*. pp. 65-70.

[156] Montero, F, Navarro, E. (2009). ATRIUM: Software Architecture Driven by Requirement. In Proc. of 14th IEEE International Conference on Engineering of Complex Computer Systems, pp.230-239.

[157] Moore, B., Dean, D., Gerber, A., Wagenknecht, G., & Vanderheyden, P. (2004). *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*. IBM.

[158] Moran, T.P. and Carroll, J.M. (1996). *Design Rationale: Concepts, Methods and Techniques*. Hillsdale, NJ: Erlbaum.

[159] Navarro, E. (2007). *Architecture Traced from Requirements applying a Unified Methodology*, PhD Thesis, Computing Systems Department.

[160] Navarro, E., Cuesta C. E. (2008). Automating the Trace of Architectural Design Decisions and Rationales Using a MDD Approach. *In Proc. 2nd European Conference Software Architecture*, LNCS 5292, Springer-Verlag, pp. 114-130.

[161] Navarro, E., Cuesta, C. E., Perry, D. E. (2009). Weaving a network of architectural knowledge. *In Proc. of WICSA/ECSA '09. Joint Working IEEE/IFIP Conf. on Soft. Arch., & European Conf. on Soft. Arch.*, pp. 241-244.

[162] NEMA Foundation. (2003). *DICOM 3.0: Digital Imaging and Communications in Medicine*, http://medical.nema.org/dicom/2003.html/

[163] NEXOF Project. http://www.nexof-ra.eu/

[164] NEXOF-RA Project Team. (2009). *NEXOF-RA Model V2.0; Public Project Deliverable 6.2_v2.0*. Available from: http://www.nexof-ra.eu/?q=rep/term/140

[165] OASIS. (2007). Reference Model for Service Oriented Architecture. Committee draft 1.0. Available from: http://www.oasis-open.org/committees/download.php/16587/wd-soa-rm-cd1ED.pdf

[166] OASIS: Organization for the Advancement of Structured Information Standards. http://www.oasis-open.org/

[167] Ociepka, B. (2004). Defending the pick and roll. *FIBA assist magazine*, pp.31-34.

[168] Ogrinz, M. *Mashup Patterns: Designs and Examples for the Modern Enterprise*. Ed. Addison-Wesley Professional. ISBN: 032157947

[169] OMG, OASIS, The Open Group. (2009). *Navigating the SOA Open Standards Landscape Around Architecture*. Document-ad/09-08-21. Available at http://www.omg.org/cgi-bin/doc?ad/2009-08-21.

[170] OMG. (2001). *MDA Guide* Version 1.0.1. Document number omg/2003-06-01. Available at: http://www.omg.org/docs/omg/03-06-01.pdf

[171] OMG. (2001). *Model Driven Architecture (MDA)*. Document No. ormsc/2001-07-01. Available at: http://www.omg.com/mda.

[172] OMG. (2001). *The Meta Object Facility (MOF) Core Specification, Version 2.0.* OMG Document - formal/06-01-01

[173] OMG. (2003). *Query/View/Transformation (QVT), Version 1.0*. OMG Document - formal/08-04-03

[174] OMG. (2001). *Object Constraint Language Specification (OCL), Version 2.0*. OMG Document - formal/2006-05-01

[175] OMG. (2007). *Unified Modelling Language (UML): Superstructure, Version 2.1.1*. OMG document - formal/2007-02-05.

[176] OMG. (2007). *XML Metadata Interchange (XMI) specification V2.1.1*. OMG Document - formal/2007-12-01.

[177] *Open Grid Service Infrastructure v1.0 (OGSI)*. Obtained from http://www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf

[178] *Open Grid Services Architecture – OGSA.* Obtained from http://forge.gridforum.org/projects/ogsa-wg/document/draft-ggf-ogsa-spec/en/23

[179] Open Service Oriented Architecture Collaboration. *Service Component Architecture Project*. http://www.osoa.org/

[180] Oquendo, F. (2004). π-ADL: An Architecture Description Language based on the Higher Order Typed π-Calculus for Specifying Dynamic and Mobile Software Architectures. *ACM Software Engineering Notes*, No. 3, May 2004.

[181] Paolucci, M., Srinivasan, N., Sycara, K., and Nishimura, T. (2003). Toward a Semantic Choreography of Web Services: From WSDL to DAML-S. *In Proc. of ICWS'03.*

[182] Papazoglou, M. P. (2003). Service-Oriented Computing: Concepts, Characteristics and Directions. *In Proc. of the Fourth International Conference on Web Information Systems Engineering (WISE'03)*, pp. 3-12. Roma, Italy.

[183] Parnas, D. L. (1972). On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12), 1053-1058.

[184] Pastor, O., Molina, J.C. (2007). *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*. Springer-Verlag.

[185] Pautasso, C., Zimmermann, O., Leymann, F. (2008). RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. *In Proc. of the 17th International World Wide Web Conference (WWW2008)*. Bejing, China.

[186] Pérez J. (2006). *PRISMA: Aspect-Oriented Software Architectures*. PhD Thesis, Department of Information Systems and Computation, Polytechnic University of Valencia.

[187] Perovich D., Bastarrica M. C., Rojas C. (2009). Model-Driven Approach to Software Architecture Design. *In proceedings of the 4th International Workshop on Sharing and Reusing Architectural Knowledge, SHARK 2009*, IEEE Computer Society, pages 1 – 8, Vancouver, Canada.

[188] Perry, D. E. and Wolf, A. L. (1992). Foundations for the study of software architecture. ACM SIGSOFT Software Engineering Notes, 17(4), pp. 40-52.

[189] Perry, D. E. (1998). Generic Architecture Descriptions for Product Lines. ARES II: Software Architectures for Product Families. Gran Canaria, Spain.

[190] Pilato, C., Collins-Sunsman, B., & Fitzpatrick, B. (2008). *Version Control with Subversion*. O'Reilly Media.

[191] PLASTIC Project. (2006). *D1.2: Formal description of the PLASTIC conceptual model and of its relationship with the PLASTIC platform toolset*. http://www-c.inria.fr/plastic

[192] Plaszczak, P. and Wellner, R. (2005). *Grid Computing*. Elsevier/Morgan Kaufmann, San Francisco.

[193] Pressman, R. S. (2005). *Software Engineering: A Practitioner's Approach: 6th Edition*. Ed. McGraw Hill.

[194] Remmert, H. (2003). Analysis of group-tactical offensive behavior in elite basketball on the basis of a process orientated model. *EJSS*, Vol. 3 (3), pp. 1-12.

[195] Ren, X., Ong, M., Allan, G., Kadirkamanathan, V., Thompson, H.A. and Fleming, P.J. (2004). Service oriented architecture on the Grid for FDI integration. *In Proc. of the 3rd UK e-Science All Hands Meeting (AHM 2004)*. Nottingham, UK.

[196] Rennie M. W., Misic V. B. (2004). *Towards a Service-Based Architecture Description Language*. TR 04/08, Technical Report, University of Manitoba, Canada.

[197] Royce W. W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. *In Technical Papers of Western Electronic Show and Convention (WesCon)*. Los Angeles, USA.

[198] Rozanski N. and Woods E. (2005). *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives.* Addison-Wesley Professional.

[199] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. (1991). *Object-Oriented Modeling and Design*. Prentice Hall. ISBN 0-13-629841-9.

[200] Sanchez, D.M., Cavero, J.M., Marcos, E. (2009). The Concepts of Model in Information Systems: a Proposal for an Ontology of Models. *The Knowledge Engineering Review (KER)*. Vol. 24(1):5-21. Ed. Cambridge University Press. United Kingdom. ISSN: 0269-8889.

[201] Sánchez Cuadrado, J. & García Molina, J. (2008). Approaches for Model Transformation Reuse: Factorization and Composition. *Paper presented at the 1st International conference on Theory and Practice of Model Transformations (ICMT 2008)*. Zurich, Switzerland.

[202] Schmidt, D.C. (2006). Guest Editor's Introduction: Model-Driven Engineering. *Computer*, vol. 39(2), pp. 25-31.

[203] Seidewitz, E. (2003). What models mean. *IEEE Software*, 20(5):26–32.

[204] Selic, B. (2003). The pragmatics of Model-Driven development. *IEEE Software*. Vol. 20(5), pp. 19-25.

[205] Selic, B. (2008). Personal reflections on automation, programming culture, and model-based software engineering. *Automated Software Engineering*, 15(3), 379-391

[206] Sendall, S., & Kozaczynski, W. (2010). Model Transformation–the Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5), 42-45

[207] *Service Centric Systems Engineering* (SeCSE) Project. http://www.secse-project.eu/

[208] Shaw, M. and Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall.

[209] Shaw, M. (1996). Procedure Calls Are the Assembly Language of Software Interconnection Connectors Deserve First-Class Status. *In proc of Studies of Software Design*, LNCS 1078, Springer-Verlag.

[210] Smith M., Friese T., Freisleben B. (2006). Model Driven Development of Service-Oriented Grid Applications. In *Proc. of the International Conference on Internet and Web Applications and Services*, Guadeloupe, pp., IEEE Press.

[211] SMPP Forum. *SMPP v5.0 Specification*. Available from http://www.smsforum.net/

[212] *SOA Manifesto* (2009). Accessible from: http://www.soa-manifesto.org/

[213] *Software Engineering for Service-Oriented Overlay Computers* (SENSORIA) Project. http://www.sensoria-ist.eu/

[214] Spanoudakis, G, Zisman, A, Kozlenkov, A (2005). A Service Discovery Framework for Service Centric Systems. *In Proc. of IEEE SCC 2005*, pp. 251-259.

[215] Stahl, T., Volter, M., & Czarnecki, K. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons.

[216] Szyperski, C., Gruntz, D. and Murer, S. (2002). *Component Software: Beyond Object-Oriented Programming. Second Edition*. Addison-Wesley / ACM Press, Boston. ISBN 0-201-74572-0

[217] Tang, A., Babar, M.A., Gorton, I. and Han, J. (2006). A Survey of Architecture Design Rationale. *Journal of Systems and Software*, 79(12):1792-1804.

[218] Taylor, R. N. Medvidovic, N., Anderson, K. M., Whitehead Jr. et al. (1996). A Component- and Message-based Architectural Style for GUI Software. *IEEE Trans. Soft. Eng*. Vol. 22 (6), pp. 390–406.

[219] The Internet Engineering Task Force (IETF). http://www.ietf.org/

[220] The Open Group. (2003). *The Open Group Architecture Framework (TOGAF)* 8.1 Enterprise Edition, Doc Number: G051.

[221] Tikhomirov, A., & Shatali, A. (2008). Introduction to the Graphical Modeling Framework. *Tutorial at the EclipseCON 2008*. Santa Clara, California.

[222] Tratt, L. (2005). Model transformations and tool integration. *Software and Systems Modeling*, Vol. 4 (2), pp. 112 - 122.

[223] Uhl, A. (2008). Model-Driven Development in the Enterprise. *IEEE Software*. Vol. 25(1).

[224] Valverde, F., P. Valderas, et al. (2007). OOWS Suite: Un Entorno de desarrollo para Aplicaciones Web basado en MDA. *In Proc. of IDEAS'07*. Venezuela.

[225] Vanhooff, B., Ayed, D., & Berbers, Y. (2006). A Framework for Transformation Chain Design Processes. *Paper presented at the First European Workshop on Composition of Model Transformations - CMT 2006; European Conference on Model Driven Architecture (ECMDA-FA 2006)*, Bilbao, Spain.

[226] Vara, J. M., Didonet Del Fabro, M., Jouault, F., & Bézivin, J. (2008). Model Weaving Support for Migrating Software Artefacts from AUTOSAR 2.0 to AUTOSAR 2.X. *In proc. of ERTS 2008*. Toulouse, France.

[227] Vara, J. M. (2009). *M2DAT: a technical solution for Model-Driven development of Web Information Systems*. PhD Thesis, Rey Juan Carlos University, Madrid.

[228] Vela B. (2003). *MIDAS/BD: Una Metodología basada en Modelos para el Desarrollo de la Dimensión Estructural de Sistemas de Información Web*. PhD Thesis, Rey Juan Carlos University, Madrid.

[229] Völter, M. (2008). MD* Best Practices. Retrieved February 20, 2008, from http://www.voelter.de.

[230] Vom Brocke, J. & Rosemann, M. (2010), *Handbook on Business Process Management: Strategic Alignment, Governance, People and Culture (International Handbooks on Information Systems)*. Vol. 1. Springer-Verlag.

[231] W3C. (2000). *Resource Description Framework (RDF) Schema Specification 1.0*. Brickley D., Guha, R.V. (Eds.). http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[232] W3C. (2005). *Web Services Addressing (WS-Addressing) Specification.* http://www.w3.org/Submission/ws-addressing/

[233] W3C. (2004). *Web Services Architecture* (WSA). http://www.w3.org/TR/ws-arch/

[234] W3C. (2005). *Web Services Choreography Description Language (WS-CDL) Version 1.0*. http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/

[235] W3C. (2007). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. R. Chinnici, J-J. Moreau, A. Ryman, S. Weerawarana (Eds.). Available at http://www.w3.org/TR/2007/REC-wsdl20-20070626.

[236] W3C. (2002). *Web Service Standards*. http://www.w3.org/2002/ws/

[237] Wada, H., Suzuki, J. and Oba, K. (2006). Modeling Non-Functional Aspects in Service Oriented Architecture. *In Proc. of the 2006 IEEE International Conference on Service Computing*. Chicago, IL.

[238] WebRatio Site. *WebRatio Development Studio*: http://www.webratio.com

[239] Weerawarana S., Curbera F., Leymann F., Storey T., and Ferguson D. F. (2005). *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR.

[240] Xiangyang, J., Shi, Y., Cao H., Xie, D. (2007). A New Architecture Description Language for Service-Oriented Architecture. *In proc of GCC 2007*, pp. 96-103.

[241] Yin R. K. Case Study Research: Design and Methods. Fourth Edition. SAGE Publications. California, 2009. ISBN 978-1-4129-6099-1.

[242] Zdun, U. and Dustdar, S. (2007). Model-Driven Integration of Process-Driven SOA Models. *Intl. J. of Business Process Integration and Management*. Vol. 2(2): 109-119.

[243] Zhang, T., Ying, S., Cao, S., Jia, X. (2006). A Modeling Framework for Service-Oriented Architecture. *Proceedings of the Sixth International Conference on Quality Software (QSIC 2006)*, pp. 219-226.

[244] Zimmermann, O., Gschwind, T., Kuester, J., Leymann, F., Schuster, N. (2007). Reusable architectural decision models for enterprise application development. *In Overhage, S., Szyperski, C., eds.: Quality of Software Architecture (QoSA) 2007*. LNCS, Boston, USA, Springer-Verlag Berlin Heidelberg.

Appendix C:
Acronyms

Table of Acronyms

ACRONYM	*DESCRIPTION*
AMW	ATLAS Model Weaver
ATL	ATLAS Transformation Language
CASE	Computer Aided Software Engineering
CBSE	Component-Based Software Engineering
CIM	Computation Independent Model
DSL	Domain Specific Language
EMF	Eclipse Modelling Framework
EMP	Eclipse Modelling Project
GMF	Generic Modelling Framework
GPL	General Purpose Language
IDE	Integrated Development Environment
M2M	Model to Model
MBSE	Model-Based Software Engineering
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDE	Model-Driven Engineering
MDSD	Model-Driven Software Development
MOF	Meta-Object Facility
OCL	Object Constraint Language
OGSA	Open Grid Service Architecture
OMG	Object Management Group
PDM	Platform Dependent Model
PIM	Platform Independent Model
PSM	Platform Specific Model

QoS	Quality of Service
QVT	Query/View/Transformation
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
REST	REpresentational State Transfer
SCA	Service Component Architecture
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service Oriented Computing
SQL	Structured Query Language
TDM	Technology Dependent Model
URI	Universal Resource Identifier
UML	Unified Modelling Language
W3C	World Wide Web Consortium
WIS	Web Information System
WSA	Web Services Architecture
WSDL	Web Services Description Language
WSRF	Web Service Resource Framework
XML	eXtensible Markup Language