UNIVERSIDAD
REY JUAN CARLOS

ESCUELA SUPERIOR DE INGENIERÍA
INFORMÁTICA

MÁSTER UNIVERSITARIO EN
INFORMÁTICA INTERACTIVA Y MULTIMEDIA

Curso Académico 2011/2012

Trabajo Fin de Máster

DISEÑO DE INTERFACES DE USUARIO ORIENTADO A MODELOS: EXTENSIÓN DE CIAT

Autor: Laura Díaz García

Tutor: Maximiliano Paredes

A mi madre.

Gracias por haberme dado todo.

Resumen

En los últimos años ha habido una auténtica revolución en cuanto al concepto de telefonía móvil. Si hace unos años los teléfonos móviles sólo se usaban para realizar y recibir llamadas y/o mensajes de texto, con la llegada de los smartphones los teléfonos móviles se han convertido en auténticos ordenadores portátiles. Debido, entre otros factores, a esta nueva situación ha aparecido la necesidad de que una misma aplicación o página web se tenga que visualizar en una gran variedad de dispositivos con unas características de visualización muy diferentes entre ellos.

Gracias a la metodología de Desarrollo de Interfaces de Usuario Basada en Modelos (MBUID) podemos especificar la interfaz de usuario una sola vez con un nivel de abstracción alto para después ir refinando dicha especificación y poder obtener distintas interfaces dependiendo del dispositivo donde tenga que ser ejecutada o visualizada.

Este trabajo propone un método para el Desarrollo de Interfaces de Usuario Basada en Modelos. Para ello hemos analizado la arquitectura general del Desarrollo de Interfaces de Usuario Basado en Modelos y los modelos declarativos que utiliza para conseguir la automatización del proceso de desarrollo. Hemos centrado nuestro estudio en la herramienta CIAT, la cual implementa la metodología CIAM, analizando sus puntos fuertes y débiles y proponiendo una posible solución a las limitaciones detectadas.

Nuestro objetivo es por tanto proponer un método que mejore la limitación de la propuesta de Limbourg, que implementa la herramienta CIAT en su última fase, y aplicarlo en dicha herramienta. Para ello nos hemos apoyado en las técnicas de generación de ETS que propone Luyten.

Abstract

In the last few years there has been an authentic revolution as for as the concept of mobile telephones. A few years ago mobile phones were only used to make or receive calls and / or text messages but with the arrival of smartphones mobile phones have practically turned into laptop computers. Due to the advances in technology and between other varying factors, the need for web pages and other applications to have the ability to be seen in a great variety of formats to suit ranging devices with a consistent format has emerged as a pressing need.

Thanks to the methodology of Model-Based User Interface Development (MBUID) we can specify the user's interface a single time with a high level of abstraction while later refining the above mentioned specifications and ability to obtain different interfaces depending on the device where it will be used or visualized.

This work proposes a method for the Model-Based User Interface Development. For it we have analyzed the general architecture of the Model-Based User Interface Development and the declarative models that it uses to obtain the automation of the process of development. We have centered our study on the tool CIAT, which implements the methodology CIAM, analyzing the strengths and weaknesses and proposing a possible solution to detected limitations.

Our aim is therefore to propose a method that improves the limitation of Limbourg's offer, which implements the tool CIAT in his last phase, and to apply it in the above mentioned tool. For it we have relied on the technologies of generation ETS that Luyten proposes.

CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

Capítulo 1 Introducción

1.1 Motivación

El rápido desarrollo de la tecnología de los últimos años ha permitido un aumento exponencial de las prestaciones de los equipos, dando lugar a Interfaces de Usuario (IU) cada vez más complejas. De la misma forma en la que parece haber aumentado la complejidad de la IU en cuanto a diseño y componentes, podemos observar que se va disminuyendo la complejidad de su uso. Los primeros computadores, cuya interfaz se componía de un terminal de texto, sólo podían ser utilizados por personas que dedicaban esfuerzo a dominar la tecnología, con la llegada de las actuales interfaces de usuario (GUI, *Graphical User Interface*) parece patente que cualquier persona con un conocimiento básico, puede interactuar de forma sencilla con un ordenador.

Además del aumento de prestaciones de los equipos hay que destacar la variedad de dispositivos que permiten la visualización de datos y que crea la necesidad de diseñar IU que tengan que ser visualizadas tanto en pantallas de pequeño formato, como teléfonos móviles o computadoras de bolsillo, como en pantallas de pared y grandes pantallas, como pantallas de plasma. La plasticidad de sus diseños debe asegurar una conversión suave entre variaciones del tamaño de visualización, distribución por medio de navegadores web o por medio del teléfono, traducción a varios idiomas y compatibilidad con dispositivos con soporte para la accesibilidad de usuarios discapacitados (Shneiderman, 05).

En este contexto cobra cada día mayor importancia soluciones que permitan la automatización del proceso de generación de la IU, pudiendo obtener diferentes interfaces dependiendo del dispositivo final. Además, hay que añadir la complejidad en las interfaces para aplicaciones con soporte al trabajo en grupo, como son las aplicaciones CSCL (*Computer Supported Collaborative Learning*). Esto añade un concepto más a tener en cuenta a la hora de diseñar e implementar dichas interfaces, debido a que varios usuarios pueden estar interactuando sobre la misma aplicación desde dispositivos heterogéneos.

Para este tipo de sistemas la interfaz de usuario se puede especificar una vez y luego irse refinando para cada uno de los dispositivos de destino hasta llegar a la aplicación o interfaz de usuario final. Este enfoque puede ser apoyado por la

metodología de desarrollo de software MDE (*Model-Driven engineering*) (MDE) que consiste en la creación de modelos o abstracciones, más cercanos a un dominio particular que otros conceptos computacionales. Este enfoque permite incrementar la productividad maximizando la compatibilidad entre sistemas y simplificando el proceso de desarrollo. Esta metodología aplicada en el campo del diseño de las IU, se denomina desarrollo de interfaces de usuario basado en modelos (MB-UID –*Model-Based User Interface Development*) (López Jaquero, 2005) y consiste en la especificación de la interfaz de usuario utilizando modelos declarativos que describen los distintos componentes a tener en cuenta en el desarrollo de una interfaz de usuario.

Las herramientas que implementan esta metodología son las denominadas MBUIDE. Habitualmente proporcionan una herramienta visual que hace uso de una notación gráfica para la especificación de los distintos modelos. El objetivo de estas herramientas es el de aumentar el nivel de abstracción en el diseño de Interfaces de Usuario, y en una última fase mediante el uso de generadores de código conseguir la generación de la IU de forma automática. De esta forma si hubiera algún cambio en los requisitos del sistema, bastaría con cambiar los modelos abstractos de la Interfaz y el coste de generación del código final se reduciría.

Nuestro trabajo se basa en proponer una mejora en la herramienta CIAT(GUI) ya que se han detectado algunas carencias importantes en su funcionamiento. En una fase inicial observamos que CIAT(GUI) se podría mejorar mediante un nuevo mecanismo de transformación de Interfaces de Usuario Abstractas a Interfaces de Usuario Concretas (Díaz, 2010) (Díaz et al, 2010). En el presente trabajo refinamos dicha propuesta definiendo formalmente los nuevos elementos de la metodología que proponemos e integrando éstos en la herramienta.

1.2 Objetivos

De acuerdo a los motivos que se han expuesto, los objetivos que pretende el presente proyecto son:

- Estudio y análisis de la implantación de procesos computacionales para la generación automática de Interfaces de Usuario.
- Estudio de la metodología MBUID así como de las herramientas que consideramos relevantes que la soportan.

- Estudiar la metodología CIAM para el desarrollo de aplicaciones colaborativas.
- Estudiar qué metodologías existentes en la literatura son útiles para mejorar las limitaciones de las herramientas ya desarrolladas. En concreto nos centramos en el enfoque propuesto por Limbourg y por consiguiente en la herramienta CIAT (GUI) que implementa dicho enfoque.
- Estudiar y valorar la utilización de la propuesta de Luyten (ETS) para resolver las limitaciones encontradas.
- Estudiar el desarrollo de una herramienta que mejore y resuelva las limitaciones de CIAT(GUI), utilizando el modelo de datos ETS, de tal forma que dicha herramienta se integre con CIAT(GUI).
- Desarrollar dicha herramienta e integrarla con CIAT(GUI).
- Proponer una línea futura de investigación para lograr la automatización de generación de Interfaces de Usuario complejas en entornos colaborativos.

1.3 Método de trabajo

El proyecto que se aborda en este trabajo plantea una metodología que principalmente se basa en la búsqueda de información e investigación, además de sintetizar la información recogida para obtener unas conclusiones. Esta metodología pretende conseguir los diferentes objetivos que se especificaron en el apartado anterior. El método de trabajo que se va a seguir en este proyecto, es el siguiente:

- Estudiar los entornos de desarrollo basados en modelos para el desarrollo de IU junto con las herramientas disponibles.
- Estudiar las reglas de transformación de Limbourg.
- Estudiar la propuesta de Luyten y valorar su implantación computacional.
- Estudiar la herramienta CIAT(GUI).
- Realizar las modificaciones oportunas en CIAT(GUI) para implementar la propuesta de Luyten.

1.4 Estructura de la memoria

En este apartado se describe la estructura de la presente memoria.

En el capítulo 1, **"Introducción",** se describe en qué se ha motivado el proyecto, presentando los objetivos y el método de trabajo seguido en su desarrollo.

En el capítulo 2, **"MB-UID",** se describe la arquitectura general del desarrollo de interfaces de usuario basado en modelos. Además se analizan tres de los entornos más significativos para nuestro trabajo que utilizan dicho paradigma y se realiza una comparación entre los entornos más relevantes dentro del campo de la investigación.

En el capítulo 3, **"Marco de trabajo CIAM"**, se analiza la metodología de desarrollo de aplicaciones colaborativas en las que se basa la herramienta que vamos a mejorar.

En el capítulo 4, **"Propuesta de Limbourg",** se realiza un estudio detallado de la propuesta que implementa la herramienta CIAT(GUI), estudiada en el apartado anterior.

En el capítulo 5, **"Una propuesta basada en ETS",** se analiza una propuesta para generación automática de Interfaces de Usuario basada en *Enabled Task Sets* (ETS) y se analiza cómo esta solución permite solucionar las limitaciones que se han detectado en el análisis de la herramienta MBUID CIAT(GUI), y en las reglas de transformación de Limbourg.

En el capítulo 6, **"Aplicación de la propuesta: mejora de CIAT(GUI)",** se describen las tecnologías que utiliza la herramienta CIAT(GUI) y se analiza la solución desarrollada para integrar la propuesta basada en ETS con CIAT(GUI).

Por último en el capítulo 7, **"Conclusiones y trabajos futuros",** se describen las conclusiones obtenidas del presente trabajo y se analizan los logros alcanzados y las dificultades que han ido apareciendo a lo largo de la investigación. Finamente se proponen posibles trabajos futuros que se pudieran llevar a cabo a partir de este trabajo.

Capítulo 2 MB-UID

2.1 Introducción

El desarrollo e implementación de sistemas de información (SI) (Traetteberg, 2002) y el diseño de interfaces han sido realizados tradicionalmente mediante diferentes métodos y técnicas, a menudo, las distintas personas que han realizado este trabajo formaban parte de ámbitos de trabajo distintos, por ejemplo, una persona era la encargada del desarrollo de la lógica de la aplicación y otra persona distinta se encargaba del diseño de la interfaz. Tal separación puede resultar equivocada ya que la mayoría de los SI interactúan con el usuario final de la misma, a través de una IU, y a menudo, dicha interfaz es percibida como el sistema completo por el usuario, por lo tanto, para el usuario final la distinción entre interfaz y sistema carece de sentido. A esta situación hay que añadir el hecho de que el uso de una aplicación concreta forma parte de un mayor contexto de información que puede ser informático o no. Por estos motivos, puede considerarse erróneo aislar el diseño de la IU del diseño del sistema, aún así, sí hay que tener en cuenta la distinción contextual entre ambos. Además, el usuario final de una aplicación interactiva espera que ésta sea fácil de usar y que la curva de aprendizaje sea mínima. Las interfaces que cumplen estos requisitos suelen ser difíciles de diseñar e implementar. Cuanto más sencilla de usar es una interfaz, más difícil es el proceso de creación (Schlungbaum, 1996)

Con los avances tecnológicos surgidos en los últimos años que han hecho llegar al mercado nuevos tipos de dispositivos tales como PDAs, teléfonos móviles, etc., ha aumentado la necesidad de desarrollar distintas IU para el mismo sistema de información. Las diferentes características de estos dispositivos dan lugar a diferentes interfaces, mientras que el sistema de información permanece siendo esencialmente el mismo.

El concepto de sistema interactivo parece apreciar la dualidad y la necesidad de integración entre el desarrollo de sistemas y diseño de la interfaz. La gestión de esta integración, así como sus detalles por separado son la clave para el éxito del desarrollo de sistemas interactivos.

En este contexto, surge la necesidad de establecer una metodología que permita integrar el desarrollo de sistemas de información con el diseño de las interfaces de

usuario teniendo en cuenta la variedad de dispositivos en los que finalmente se ejecutará el sistema de información resultante. Para este tipo de sistemas la interfaz de usuario se puede especificar una vez y luego irse refinando para cada uno de los dispositivos de destino hasta llegar a la aplicación. Este enfoque puede ser apoyado por la metodología de desarrollo de software MDE (*Model-Driven engineering)* [2] que consiste en la creación de modelos o abstracciones, más cercanos a un dominio particular que otros conceptos computacionales. Este enfoque permite incrementar la productividad maximizando la compatibilidad entre sistemas y simplificando el proceso de desarrollo. Esta metodología aplicada en el campo del diseño de las IU se denomina desarrollo de interfaces de usuario basado en modelos (MB-UID –*Model-Based User Interface Development*) (López Jaquero, 2005) y consiste en la especificación de la interfaz de usuario utilizando modelos declarativos que describen los distintos componentes involucrados en el desarrollo de una interfaz de usuario. La creación de los modelos suele realizarse mediante herramientas visuales donde el usuario hace uso de una notación gráfica que permite la especificación de los distintos modelos de una manera más sencilla, por lo que para poder realizar dicha especificación no serían necesarios conocimientos, por ejemplo, del lenguaje final de implementación. Las aproximaciones basadas en modelos persiguen aumentar el nivel de abstracción usado en el diseño de la interfaz de usuario, dejando los detalles de implementación a generadores de código, y permitiendo una generación total o parcial de la interfaz de usuario de forma sencilla cuando los requisitos cambian. De igual manera, dentro de dichas aproximaciones también se persigue la portabilidad de las interfaces de usuario, de forma que un mismo diseño pueda ser convertido en código ejecutable para distintas plataformas o lenguajes sin necesidad de un rediseño de la interfaz. De esta forma podemos observar que con este tipo de aproximación se puede dar una solución a los problemas descritos anteriormente.

2.2 Arquitectura general MB-UID

La arquitectura general dentro del diseño de interfaces de usuario basado en modelos puede observarse en la Figura 4 (Schlungbaum, 1996). Los principales componentes de esta arquitectura son los denominados Modelos de Interfaz, que representan el conocimiento que se tiene de la interfaz de usuario en distintos niveles, el desarrollador utiliza una herramienta de modelado en un entorno interactivo, que le permite crear y

modificar los distintos modelos. Además en todo el proceso se hace uso de una base de conocimiento donde se recopila la experiencia adquirida por los desarrolladores.

Un motor de generación automática de código generará el código de la interfaz de usuario usando para ello los modelos creados por el desarrollador de la interfaz de usuario y la experiencia recopilada sobre el diseño de interfaces de usuario.

Como se puede ver en la figura 4 en esta aproximación existe una separación entre el desarrollo de la parte funcional de la aplicación y su interfaz de usuario. Cada parte es desarrollada por separado para más tarde unirlas para generar la aplicación final.

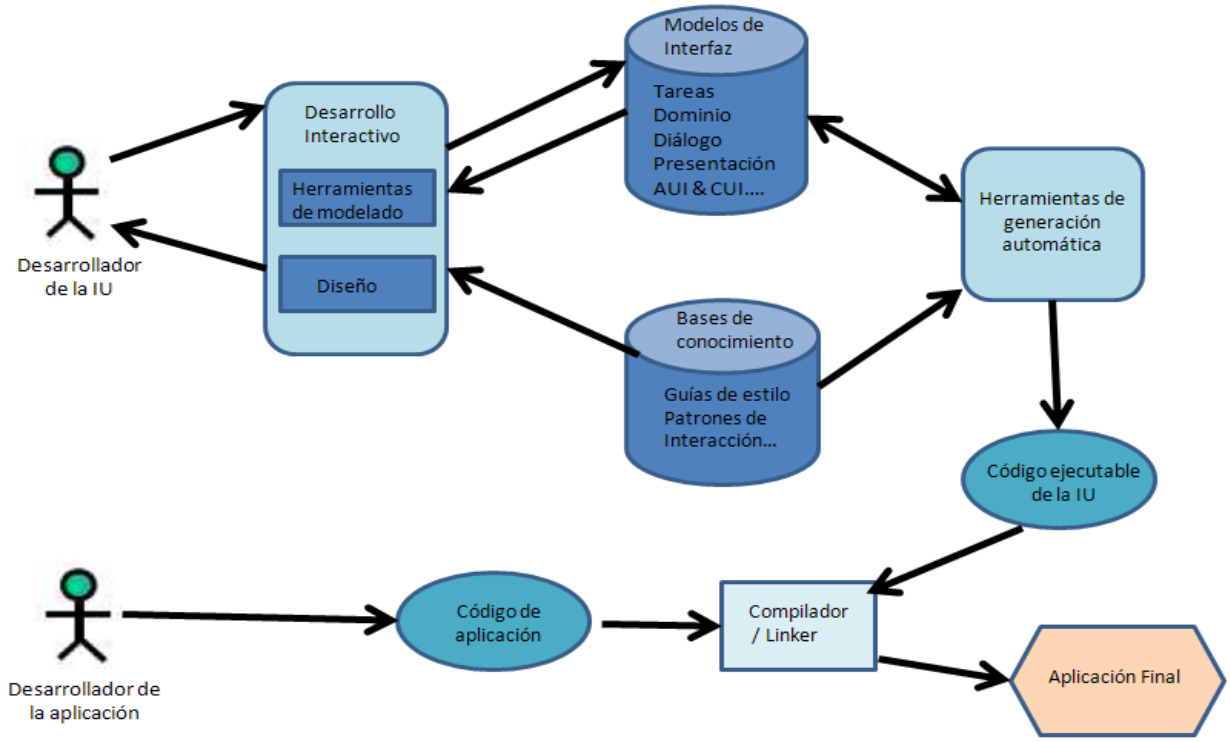


Figura 1: Arquitectura general de MB-UID

La utilización de MBUID presenta una serie de ventajas (Clerckx et al, 2004):

- Diseño de interfaces abstractas: permiten una descripción más abstracta de la interfaz de usuario que los métodos tradicionales, de este modo, diseñadores sin conocimientos concretos de la aplicación se pueden encargar del diseño de la IU. Además estos modelos suelen ser construidos por medio de herramientas que proporcionan una interfaz gráfica intuitiva y suelen ser fáciles de aprender.

- Asistencia/Automatización en la generación de IU: Una vez definidos los modelos abstractos, existen herramientas que permiten la transformación de esos modelos abstractos a modelos de un menor nivel de abstracción. Esta característica hace que el proceso de diseño de interfaces de usuario sea más rápido.
- Verificación y validación: El uso de modelos formales y descripciones facilita la comprobación de los requisitos del sistema. De esta forma se puede validar el sistema en una fase de desarrollo más temprana.
- Consistencia de las IU. Las interfaces de usuario construidas mediante la misma herramienta MBUID son consistentes. Por ejemplo, si se construye una IU para una aplicación determinada y pasado un tiempo se necesita ampliar dicha aplicación, si la nueva IU se construye con la misma herramienta MBUID, la IU resultante será consistente respecto a la generada previamente.

Por otra parte, estas aproximaciones también presentan algunos problemas por resolver:

- Complejidad de los modelos: provoca que en la mayoría de los casos aprender su funcionamiento sea complicado. Esta complejidad puede disminuir mediante el uso de herramientas de diseño visuales.
- La integración de la parte funcional de la aplicación y de su interfaz de usuario todavía no está totalmente resuelta.
- No existe un estándar sobre cuáles deben ser los modelos para realizar el modelado de la interfaz de usuario ni cuáles de los aspectos de ésta deben modelarse.

2.2.1 Modelos MB-UID

Aunque no existe un estándar sobre cuáles son los modelos específicos que tienen que ser utilizados en el desarrollo basado en modelos, sí que existe una serie de modelos comunes que aparecen en casi todas las aproximaciones. Estos modelos son: tareas, dominio, usuario, diálogo y presentación.

- **Modelo de tareas.** El modelo de tareas expresa cuáles son las tareas que va a realizar el usuario de la aplicación a través de la IU. Las tareas se descomponen en acciones indivisibles que representan los pasos necesarios para alcanzar los objetivos de la tarea. Para la especificación del modelo de tareas se han utilizado

distintas aproximaciones, cabe destacar el método basado en formalismos ConcurTaskTrees (CTT) (Paternò et al. 1997) desarrollado por Paternò, que se ha convertido en un estándar de facto en los últimos años. En a sección 6.2.7 detallamos esta notación.

- **Modelo de dominio.** El modelo de dominio incluye una visión de los objetos sobre los que actúan las tareas capturadas en el modelo de tareas. La especificación del modelo de dominio va muy ligada a las especificaciones realizadas dentro del modelado del dominio de la parte funcional.
- **Modelo de usuario.** El modelo de usuario captura las características y requisitos individuales de cada usuario o grupo de usuarios. Habitualmente cada uno de los tipos de usuario que interactuarán con la aplicación es denominado rol (papel). El objetivo de este modelo es ofrecer una IU que se ajuste a las características y requisitos de cada usuario. De esta forma, en tiempo de diseño se podrá definir cuáles serán las tareas disponibles según el tipo de usuario. Cabe destacar que este modelo resulta especialmente útil dentro del diseño de tareas colaborativas.
- **Modelo de diálogo.** El modelo de diálogo describe las posibles interacciones entre la IU y el usuario. Por ejemplo, puede representar que el usuario puede introducir datos, seleccionar un determinado objeto de la interfaz o decidir cuándo se muestran los datos. En general, representa una secuencia de entradas y salidas.
- **Modelo de presentación.** El modelo de presentación contiene una descripción de la interfaz de usuario final con la que el usuario interactuará.

  En algunos casos, existen dos modelos de presentación, en primer lugar se construye el modelo abstracto, que describe la interfaz de usuario en función de objetos abstractos de interacción (AIO – *Abstract Interaction Object)*, y por otro lado se construye un modelo de presentación concreto, que estará compuesto por objetos concretos de interacción (CIO – *Concrete Interaction Object).* El conjunto de objetos concretos de interacción para una plataforma no tiene por qué coincidir con los objetos de interacción concretos de otra plataforma. Esta separación en nivel abstracto y concreto del modelo de presentación permite una generación de la interfaz de usuario para distintas plataformas a partir de una misma descripción abstracta de la interfaz, donde será necesario seleccionar los objetos concretos de interacción correspondientes a cada objeto abstracto de interacción.

2.3 Entornos de desarrollo de interfaces de usuario orientado a modelos (MBUIDE)

Existen numerosos entornos para el desarrollo basado en modelos de interfaces de usuario, en general, estos entornos se pueden clasificar en varias generaciones (Gooma et al, 2005). La primera generación tiene como objetivo el de ofrecer una estrategia para generar una IU a partir de los modelos de alto nivel. Las herramientas de esta generación hicieron hincapié en la generación automática de interfaces en lugar de en el proceso de diseño de la IU. En la segunda generación destaca la participación de los usuarios en el proceso de desarrollo y es en este punto en el que comienzan las MBUIDE centradas en el usuario. En la segunda generación, el modelo de interfaz se describe de mejor forma. Podemos incluso aventurarnos a incluir una tercera generación de herramientas, que se corresponderían con las herramientas que contemplan aspectos colaborativos de las IU. Estas herramientas se caracterizarían por proveer de una notación específica para aplicaciones colaborativas.

A continuación describimos los entornos que hemos denominado de tercera generación ya que son éstos los que consideramos relevantes para nuestro trabajo. En la tabla 3 se puede observar una comparación en forma resumida de las herramientas MBUIDE más importantes.

2.3.1 Teresa

TERESA (Berti et al. 2004) es un entorno basado en modelos que proporciona apoyo en el diseño y desarrollo de interfaces accesibles a través de diversos tipos de dispositivos sobre entornos basados en la Web. TERESA permite la generación de interfaces de usuario en diferentes plataformas (portátiles, ordenadores de sobremesa, PDA, etc.) a partir del modelado de tareas en la notación CTT (Paternò et al. 1997). TERESA proporciona un entorno completo semiautomático, basado en una serie de transformaciones, que permite a los diseñadores crear y analizar su diseño en diferentes niveles de abstracción. Las principales etapas son: Modelado de tareas de alto nivel y de múltiple contexto, desarrollo del modelo de tareas para diferentes plataformas consideradas, generación de interfaces abstractas y generación de la interfaz de usuario final.

2.3.2 Seescoa

Seescoa (Rigole, 2005) proporciona un mecanismo que produce automáticamente Interfaces de Usuario Finales para distintas plataformas a partir de un conjunto de modelos de partida. Se parte de un modelo independiente de la plataforma en notación de Interfaz de Usuario Abstracta (IUA) y este modelo se va refinando dependiendo de la plataforma donde vaya a ejecutarse. En esta herramienta no se utilizan conceptos o modelos de tareas, el punto de entrada de este enfoque está ubicado en el nivel de AUI. Dygimes (Coninx et al, 2003) es una versión ampliada de Seescoa que adopta el mismo enfoque que en Seescoa, excepto que la AUI se obtiene de un modelo de tareas CTT.

2.3.3 MARIAE

MARIAE (Paternò, 2011) es un entorno basado en modelos que proporciona un mecanismo para el diseño y desarrollo de aplicaciones interactivas para distintos tipos de plataformas (escritorio, smartphones…) y que utiliza Web services para ello. Surge como una evolución del entorno TERESA (Berti et al. 2004), al igual que ésta, se parte del modelado de tareas en notación CTT, una vez definido dicho modelo, se accede a un determinado Web service que asocia las tareas especificadas en él con determinadas tareas del modelo de tareas definido. Gracias a esta conexión el diseñador puede refinar el modelo de tareas con la información obtenida del Web service. Como resultado de esta fase se obtiene un modelo de tareas "enriquecido" con la información obtenida. A partir del modelo de tareas "enriquecido" se genera la interfaz de usuario abstracta, luego ésta se refina para generar la interfaz de usuario concreta y por último se genera la interfaz de usuario para el tipo de dispositivo que se haya elegido y mediante el cual que accederá al Web service de partida.

2.3.4 CIAT(GUI)

CIAT (Giraldo et al. 2009) es una herramienta que surge como soporte de la metodología CIAM (Molina et al. 2008). Esta metodología soporta el proceso de desarrollo y especificación de sistemas colaborativos. En concreto el módulo CIAT(GUI) (García, G. 2010) es el que se encarga de la generación de IU a partir de modelos de tareas y datos.

CIAT(GUI) parte de modelos de tareas en notación CTT (obtenidos en la última fase de la metodología CIAM) y modelos de datos en UML (diagramas de clase).

Partiendo de estos dos modelos se establece la trazabilidad del sistema, denominada modelo de mapping que consiste en la relación existente entre las tareas del modelo CTT y los datos, representados mediante un diagrama de clases UML. Una vez especificados estos modelos se identifica la especificación de la IU tanto a nivel abstracto (AIU) como a nivel concreto (CUI), para conseguir dichas especificaciones se utilizan las reglas de transformación de Limbourg (Limbourg, 2004).

La AUI se describe en términos de *Objetos de Interacción Abstractos* (AIO) y de relaciones abstractas que a su vez se transforman en *Objetos de Interacción Concretos* (CIO) y re relaciones concretas. Estos dos modelos se corresponden con el modelo de presentación del sistema.

Antes de realizar una transformación desde un modelo a otro, la herramienta comprueba que los modelos estén bien formados y sean consistentes.

Finalmente, partiendo de la CUI, se obtiene la interfaz de usuario final en lenguaje XAML.

Hay que destacar que esta herramienta proporciona distintos editores gráficos que permiten la especificación y/o modificación de la IU en cada una de sus representaciones, tanto iniciales como intermedias.

Esta herramienta presenta una limitación importante debido a que se basa en las reglas de transformación de Limbourg para realizar las transformaciones entre modelos, estas reglas serán analizadas en el capítulo 4. En estos momentos, aunque surge dentro del contexto CIAM, sólo permite la obtención de interfaces que no incluyen aspectos propios de los sistemas colaborativos, tales como los *widgets* multiusuario o el soporte al *awareness*. Aun así el objetivo principal es que en futuras versiones si incluya este tipo de aspectos. Como punto fuerte cabe destacar que además del código de la interfaz, genera automáticamente el modelo de datos y parte de la lógica de negocio de la aplicación.

Podemos afirmar entonces, que la herramienta CIAT da soporte computacional a la metodología de desarrollo de IU colaborativas que propone CIAM (ver Capítulo 3), esta metodología nos permite una mejora en el diseño de IU (Paredes et al 2010) frente a otras metodologías. Se trata por tanto de la herramienta más completa de las analizadas ya que la representación CTT obtenida para la generación de las IU parte de un análisis metodológico de las necesidades de interacción del sistema. Por lo tanto

creemos necesario mejorar la última parte de dicha herramienta, CIAT(GUI) para que ésta funcione con todos los tipos de operadores que se pueden representar en CTT.

Capítulo 3 Marco de trabajo: CIAM

3.1 Introducción

CIAM (Molina et al. 2008) es una de las pocas aproximaciones relacionadas con entornos de trabajo cooperativo soportado por ordenador (CSCW, por sus siglas en ingés) para el desarrollo de la interfaz de usuario. Se trata de un marco metodológico para el diseño de la interfaz de usuario en aplicaciones colaborativas.

CIAM implica la adopción de diferentes puntos de vista a la hora de abordar la creación de los modelos conceptuales de las aplicaciones *groupware*. El objetivo de este enfoque es servir como una guía para el diseñador al crear una especificación conceptual de los principales aspectos que caracterizan a los sistemas de trabajo en grupo y conducir al diseño de su arquitectura software y al diseño de la interacción que estos sistemas soportan. La información especificada en cada una de las etapas sirve como base para la elaboración de modelos a crear en la etapa siguiente, de modo que esta información se extiende, se relaciona o se especifica con mayor nivel de detalle.

3.2 Contenido metodológico

En esta sección se presentan las fases que componen la propuesta CIAM (Figura 2). Se trata de una propuesta mixta, ya que implica la adopción de distintos puntos de vista a la hora de abordar la creación de modelos conceptuales. Las primeras etapas abordan un modelado centrado en el grupo, pasando en fases posteriores a un modelo más centrado en el proceso (cooperativo, colaborativo y de coordinación), acercándose, a medida que se baja en el nivel de abstracción, hacia un modelado centrado en el usuario, en el que se modelan las tareas interactivas (es decir, el diálogo que se da entre un usuario individual y la aplicación).

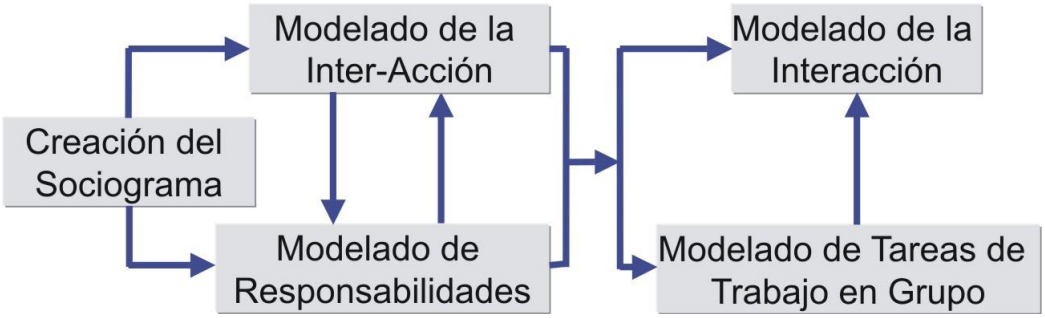


Figura 2: Etapas de la propuesta metodológica CIAM.

A continuación se describen brevemente cada una de las etapas de CIAM:

- **Creación del Sociograma*:*** Esta actividad se encarga de la especificación del *sociograma* (modelo que representa la estructura de la organización, así como las relaciones que existen entre los distintos integrantes de la misma).
- **Modelado de la responsabilidad*:*** Esta actividad presta atención a la perspectiva individual de cada uno de sus integrantes (roles) para ampliar el conjunto de responsabilidades que tiene asignado cada rol, añadiendo responsabilidades individuales que no forman parte del trabajo en grupo. Para cada tarea se especifica el objeto que maneja y los prerrequisitos que permiten su correcta ejecución (prerrequisitos en cuanto a las tareas que deben haberse completado antes de que comience la actual, así como los datos que deben estar creados antes de ejecutarla). La información detallada en esta fase se complementa con el modelado de la inter-acción, siendo necesario que ambos modelos sean coherentes entre sí.
- **Modelado de la Inter-Acción*:*** En esta actividad se definen las principales tareas (o procesos orientados a objetivos específicos) que definen el trabajo en grupo que se desarrolla en el seno de la organización.
- **Modelado del Trabajo en Grupo*:*** Esta actividad define, con un mayor nivel de detalle, las tareas cooperativas y colaborativas. Para el modelado de las tareas cooperativas se emplea el *grafo de descomposición de responsabilidades*, en el que se detallan las subtareas que componen una tarea cooperativa, de forma que, en el nivel más bajo de abstracción se contará exclusivamente con tareas individuales. En este caso, al especificar los datos manipulados por una determinada tarea podremos manejar distintos niveles de granularidad (Conjunto

de Objetos/Objeto/Atributo), ya que se pueden especificar distintas situaciones de manejo de información. Para el modelado de las tareas colaborativas se incluye la especificación de los roles participantes en su ejecución y de los objetos del modelo de datos manejados por el equipo de trabajo. La notación empleada incluye la especificación del *contexto compartido*.

- **Modelado de la Interacción*:*** Esta actividad modela los aspectos puramente interactivos de la aplicación, refiriéndonos exclusivamente a aspectos de interacción persona-ordenador. Se crea un modelo de interacción para cada tarea de naturaleza individual identificada en las distintas fases del proceso de refinamiento gradual que propone CIAM. Para cada una de ellas se crea un árbol de descomposición de tareas interactivas en notación CTT. En cuanto a las tareas colaborativas, el modelo de interacción se obtiene directamente a partir de la definición del contexto compartido.

3.3 Artefactos del sistema

A continuación se describen brevemente los distintos artefactos a generar durante la aplicación de la metodología CIAM. Estos artefactos se describen mediante la notación CIAN (*Collaborative Interactive Application Notiation*).

En la Figura 3 se presenta el conjunto de artefactos de CIAM.

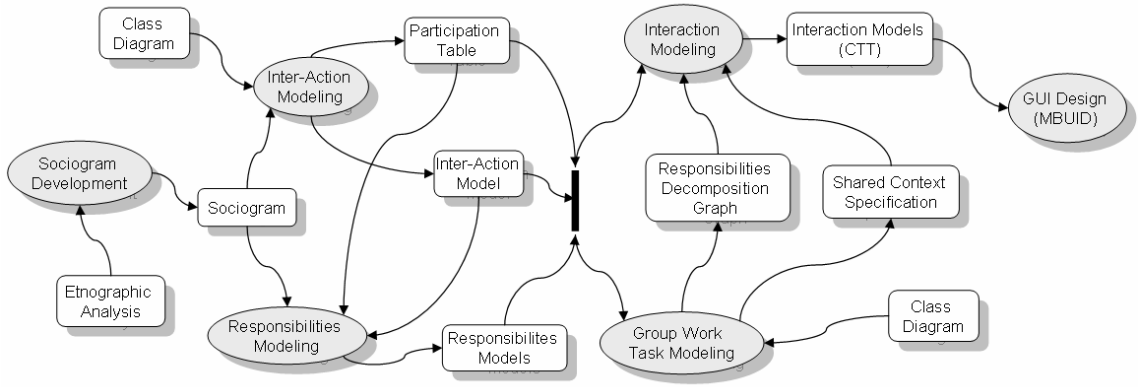


Figura 3: Mapa de fases y artefactos de CIAM (Molina et al. 2009)

Sociograma: Es un modelo gráfico que representa la estructura de la organización, así como las relaciones que existen entre los distintos integrantes de la misma. Los integrantes que forman la organización podrán entrar en una de estas

categorías: roles, actores, agentes software; o agrupaciones de los anteriores, dando lugar a grupos, agrupaciones de personas con responsabilidades homogéneas o equipos de trabajo, formado por varios roles. Los elementos de estos diagramas podrán relacionarse mediante tres tipos de relaciones básicas:

- Relación de herencia: Permite que se puedan heredar responsabilidades, siempre que se dé una determinada precondición (que puede especificarse en el modelo).
- Relación de desempeño: Permite relacionar actores y roles. Esta relación puede estar anotada por la cardinalidad (número de actores que pueden desempeñar un determinado rol dentro de la organización).
- Relaciones de asociación: Estas relaciones permite asociar roles entre sí, indicando que existen situaciones en las que dichos roles cooperarán o colaborarán en la realización de una tarea conjunta.

Modelo de Inter-Acción*:* Es un modelo gráfico que especifica las principales tareas, que definen el trabajo en grupo, que se desarrollan en el seno de la organización. Cada una de las tareas indican los roles involucrados en la misma, así como los datos manipulados y el producto generado en cada una de ellas (indicando los modos de acceso a los objetos como lectura, escritura y creación). Cada tarea deberá ser clasificada en una de las siguientes categorías: tarea cooperativa, tarea colaborativa o tarea individual. Las tareas están relacionadas entre sí mediante distintos tipos de relación que, en ciertos casos, pueden ser entendidas como dependencias: dependencia temporal (de orden), dependencia de datos (cuando las tareas necesitan datos manejados por tareas anteriores), notificación (es necesario que se genere un determinado evento de notificación para que el flujo de trabajo continúe).

Tabla de Participación: Es una descripción textual en forma de tabla que permite al diseñador tener una primera idea de la división del trabajo al nivel más elevado de abstracción. La tabla relaciona tareas de mayor nivel de abstracción y roles.

Tabla de Responsabilidades: Es una descripción textual en forma de tabla que agrupa para un rol el conjunto de responsabilidades que posee.

Grafo de Descomposición de Responsabilidades: Es una descripción gráfica de la descomposición de las tareas en grupo en tareas individuales. Este modelo permite bajar el nivel de granularidad en el modelado de la actividad permitiendo la definición

de un conjunto estructurado de modelos de interacción de mediano tamaño haciendo más fácil el diseño de la interfaz de usuario.

Especificación del Contexto Compartido: Es una especificación gráfica del conjunto de objetos que tienen importancia en el contexto de una actividad colaborativa. El modelado de tareas colaborativas implica el conocimiento de los roles implicados en su ejecución y de los objetos del modelo de datos que son manejados de forma compartida. Se representan los modificadores de acceso y los tipos de visualización que se llevan a cabo sobre los objetos.

Modelo de Interacción CTT: Es un diagrama de tareas que se basa en la notación CTT y que está asociado a las tareas individuales. Este diagrama se ha enriquecido con modificadores de visualización para enriquecer la especificación de la interfaz de usuario. Estos modificadores indican si una interfaz es de visualización colaborativa, visualización individual o de edición exclusiva.

Capítulo 4 Propuesta de Limbourg

4.1 Introducción

Limbourg (Limbourg, 2004) propone un modelo de desarrollo para la automatización de la generación de IU. Se definen cuatro modelos que representan las IU en distintos niveles de abstracción junto con varios tipos de relaciones entre los componentes que forman parte de los modelos propuestos, estas relaciones se utilizan para definir del comportamiento de la IU. Por último, Limbourg propone una serie de reglas de transformación que permiten pasar de la IU de un modelo de presentación a otro. A continuación definimos los modelos, las relaciones y las reglas de transformación.

4.2 Definición de modelos

Los cuatro modelos propuestos son, de mayor a menor abstracción, los siguientes:

1. *Modelo de Tareas*. Describe las tareas que son llevadas a cabo cuando el usuario interacciona con un sistema. Para describir esta interacción son necesarios además una serie de conceptos que representan el dominio del sistema. Estos conceptos son representados mediante otro modelo, llamado *Modelo de domino*, en el cual se describen los conceptos del mundo real y las interacciones del usuario con el mismo, así como las operaciones que se pueden realizar con estos conceptos. El *Modelo de domino* se corresponde con el modelo de datos del sistema, que representa las clases, atributos, métodos, objetos y relaciones del dominio, de manera que nos permite relacionar cada tarea presente en el Modelo de Tareas con los datos del sistema. El modelo de Tareas se representa mediante notación CTT, además, Limbourg propone un modelo de apoyo que es denominado Modelo de Mapping, este modelo sirve para poder relacionar efectivamente las tareas con el dominio.

2. *Interfaz de Usuario Abstracta.* Es un modelo de la interfaz de usuario que se define mediante una expresión regular que interpreta los conceptos de dominio y de tareas, de una manera que sea lo más independiente posible de las modalidades y las especificidades de la plataforma tecnológica que la soportará. Proporciona una especificación de la UI en términos de relaciones abstractas y

de *Objetos de Interacción Abstractos* (AIO). Estos últimos pueden ser de dos tipos: Componentes Abstractos Individuales (AIC) y Contenedores Abstractos (AC).Un AIC es una abstracción que permite la descripción de objetos de interacción de una forma que sea independiente del modo en el que se representarán en el mundo físico. Un AC, por su parte, es una entidad que permite la agrupación lógica de otros AC o AIC y que soportan la ejecución de un conjunto de tareas conectadas. En este modelo, aparece el concepto de faceta, de esta forma, un AIC puede estar compuesto por varias facetas. Cada faceta describe una función particular que puede representarse en el "mundo real". Se identifican cuatro facetas principales:

a. *Input,* describe una función de entrada soportada por una AIC.
b. *Output,* describe qué dato puede ser representado por el usuario en una AIC.
c. *Navigation,* describe la posible transición entre contenedores de un AIC concreto.
d. *Control,* describe los links entre un AIC y las funciones del sistema, por ejemplo, métodos desde el modelo de domino si existen.

Entre los objetos de interacción ser pueden establecer relaciones, denominadas *Relaciones de Interfaces de Usuario Abstractas* (relaciones AUI). Limbourg identifica varios tipos de relaciones.

3. *Interfaz de Usuario Concreta.* Permite la especificación de la apariencia y del comportamiento de una IU con elementos que pueden ser percibidos por los usuarios, como pueden ser una ventana, un botón, etc. En esta etapa, la representación sigue siendo independiente de la plataforma. De forma análoga a la AUI, también se compone de objetos de interacción. En este caso *Objetos Concretos de Interacción* (CIO), que pueden ser: *Contenedores Concretos* (CC), tales como ventanas, tablas, etc., o *Componentes Individuales Concretos* (CIC), tales como campos de texto. También hay relaciones concretas.

4. *Interfaz de Usuario Final* Se corresponde con la interfaz de usuario (IU) operacional, es decir, cualquier IU corriendo sobre una determinada plataforma o su representación (por ejemplo en un navegador Web) o por ejecución (por

ejemplo, después de la compilación del código en un entorno de desarrollo interactivo). La IU final tiene dos posibles representaciones el código y la representación gráfica. El código se ocupa de la representación de la IU como un conjunto de instrucciones (en lenguaje procedural) o como un conjunto de afirmaciones (en lenguaje imperativo) o en una mezcla de ambos. La representación gráfica del sistema es la representación de la IU perceptible por el usuario.

Además de estos modelos se introduce el Modelo de Contexto que permite asociar a los elementos del modelo con el contexto para los cuales son válidos, es decir, para el tipo de plataforma donde se visualizará la IU, como podría ser una PDA, un móvil o un monitor de un ordenador.

Si nos fijamos en los modelos propuestos, se puede realizar una comparación entre el enfoque MDA (MDA)y el de Limbourg que se puede observar en la siguiente tabla.

Model Driven Architecture	Propuesta de Limbourg
Computing Independent Model	Modelo de Tareas
Platform Independent model	IU Abstracta IU Concreta
Platform Specific Model	
Implementation	Final UI
Platform Model	Context Model

Tabla 1: Comparativa entre MDA y propuesta de Limbourg

4.3 Relaciones

Se definen una serie de relaciones entre los distintos elementos que componen un modelo. El propósito de estas relaciones es proporcionar al desarrollador un conjunto de relaciones predefinidas que permitan relacionar elementos desde modelos heterogéneos. Se pueden definir estas relaciones según la siguiente figura:

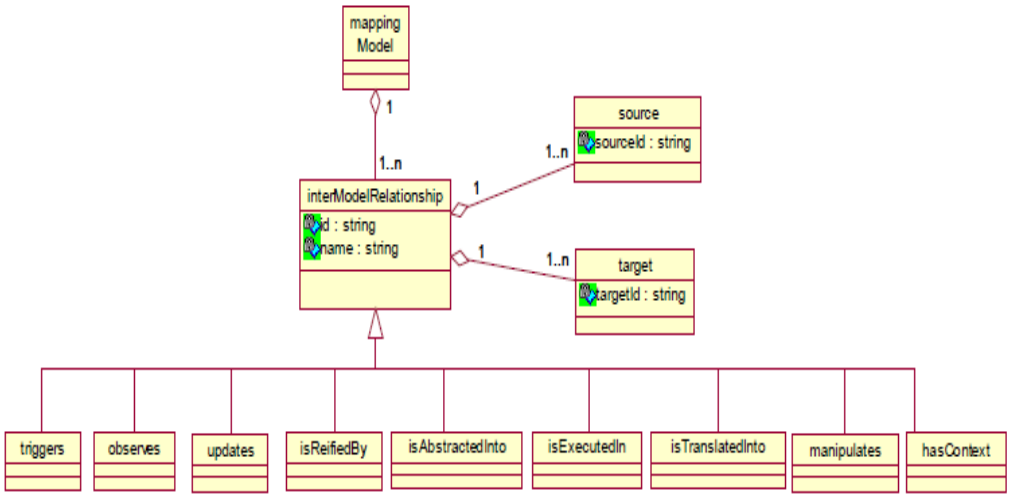


Figura 4: Relaciones intermodales

Estas relaciones pueden ser clasificadas según su tipo:

1. Relaciones entre el modelo de dominio y los modelos de IU. Todas estas relaciones están definidas ente un objeto de interacción y un concepto del modelo de dominio.

- *Observes*, se puede interpretar como que el contenido de un objeto de la UI debe ser sincronizado cuando:
    - Un atributo relacionado es modificado. El nuevo estado que resulta de esta modificación debe ser presentado en la IU.
    - Un método relacionado es ejecutado. Sus parámetros de salida son mostrados en la IU.
- *Updates*, en este caso el concepto del modelo de domino es concretamente un atributo. "*Updates*" describe la situación donde un atributo de un objeto del modelo de dominio debe ser sincronizado con el contenido de un objeto de la IU.
- *Trigger*, en este caso el concepto del modelo de dominio es concretamente una operación. Esta relación describe que un objeto de la IU es capaz de desencadenar un método desde el modelo de dominio.

2. Relaciones para asegurar la trazabilidad en el ciclo de desarrollo. Este tipo de relaciones se utiliza para asegurar la trazabilidad, debido a que se van realizando transformaciones en los modelos, con este tipo de relaciones podemos controlar la

ejecución de dichas transformaciones. Por ejemplo, podría resultar interesante saber cómo un objeto concreto del CUI se abstrae para convertirse en un objeto abstracto de la AUI.

- *Is Executed In*, relaciona una tarea con un objeto de interacción (un contenedor o un componente individual) permitiendo su ejecución. Este tipo de relación es útil para asegurar que el sistema pueda soportar de forma apropiada todas las tareas.
- *Is Reified By*, indica que un objeto abstracto es reificado desde uno abstracto mediante una transformación de reificación. Entendemos como reificación el proceso de refinar un elemento representado con un nivel de abstracción determinado a un elemento de una forma más concreta o tangible, lo que supone un menor nivel de abstracción.

3. Otras relaciones

Otra relación útil es *Manipulates*, que relaciona tareas con conceptos del dominio. Puede tratarse de un atributo, un conjunto de atributos, una clase (o un objeto), o un conjunto de clases. Esta relación es útil para encontrar la interacción más apropiada de un objeto para que soporte una determinada tarea.

4.4 Reglas de transformación

Limbourg propone una serie de reglas que permiten la transformación entre los distintos modelos. Si nos fijamos en la siguiente figura, estas reglas nos definen los pasos a seguir 1 y 2. Sin embargo, las reglas para la transformación de IU concretas a IU finales no están definidas, ya que estas reglas no se pueden generalizar porque dependen del lenguaje de programación en el que se desee realizar la aplicación y de las plataformas que se utilicen para la visualización de las interfaces.

Figura 5: Modelo de Limbourg

A continuación se detallan estas reglas de transformación.

4.4.1 Estrategia de aplicación de los sistemas de transformación

Un sistema de transformación está compuesto de varias reglas, por este motivo surge el problema de cómo aplicar estas reglas garantizando dos propiedades importantes: terminación y confluencia, es decir, que independientemente de cómo se apliquen estas reglas el resultado tiene que ser el mismo.

Limbourg utiliza una estrategia de transformación de una gramática de grafo para definir las distintas transformaciones, no entramos a definir cómo es dicha gramática de grafo ya que es irrelevante para nuestro estudio. Dicha estrategia se define como el orden en el cual las reglas de transformación se aplican a un grafo inicial. Las reglas se pueden aplicar concurrentemente, de una forma independiente del orden, o controladas de una forma secuencial.

La estrategia usada se representa con la siguiente figura:

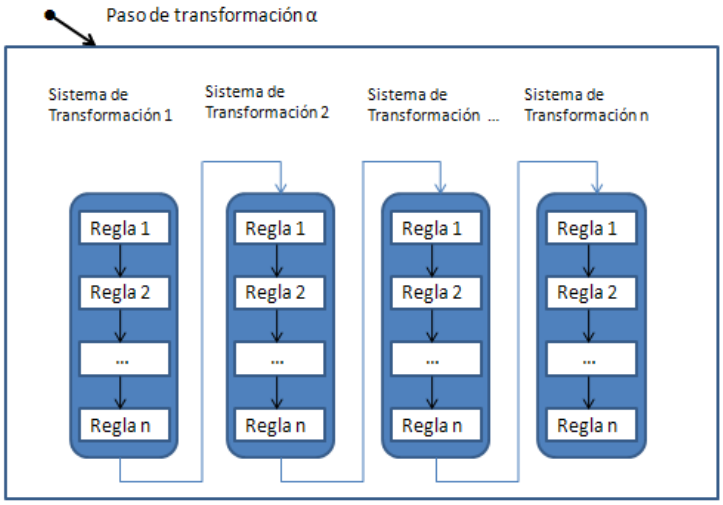


Figura 6: Estrategia de aplicación ordenada

El primer paso de desarrollo se inicializa de forma externa, luego, se ejecuta la primera transformación, una vez que ésta termine se aplica la segunda transformación y así sucesivamente hasta que la última transformación acaba. Un paso termina si cada uno de sus sub-pasos acaba. Un sistema de transformación termina, si cada una de las reglas que lo componen termina.

Una regla termina cuando no haya más coincidencias en el grafo resultante.

4.4.2 Ingeniería directa

La ingeniería directa se pude ver como una secuencia de refinamientos progresivos aplicados sobre una especificación de alto nivel para obtener el código de la aplicación o una especificación de más bajo nivel.

Si nos fijamos en la Figura 7, se pueden distinguir 3 pasos en el modelo de desarrollo propuesto por Limbourg. A continuación de tallamos en qué consiste cada uno de los paso especificados.

4.4.2.1 Paso 1: De Modelo de tareas (Task & Domain) a Interfaz de usuario Abstracta (AUI).

En este primer paso se modifica la representación de la IU partiendo de los modelos de tareas y dominio, para conseguir una AUI. Este paso se descompone a su vez en varios subpasos:

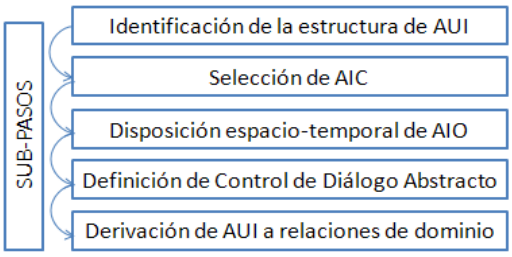


Figura 7: Paso de desarrollo: de Modelo de Tareas a AUI

Si nos fijamos en la Figura 7, podemos observar que aparecen los conceptos de componentes abstractos individuales (AIC) y objetos de interacción abstractos (AIO) que se han descrito con anterioridad.

Se propone la descomposición en 5 pasos ordenados, en primer lugar, se identifican contenedores abstractos y componentes abstractos individuales, luego, se ordenan dichos elementos y finalmente se fijan dentro de los contenedores previamente identificados. Esto completa el diseño de la representación abstracta. Luego, se añade el comportamiento dinámico de presentación de elementos (diálogo).

1.1. Sub-paso: Identificación de la estructura de AUI.

Consiste en la definición de grupos de interacción de objetos abstractos. Cada grupo corresponde con un grupo de tareas fuertemente acopladas. "Para cada grupo de tareas, hijo de la misma tarea, generar un espacio de interacción".

Regla 4-1: Para cada hoja de tarea del árbol de tareas, crear un Componente Abstracto Individual. Para cada tarea, padre de una hoja de tareas, crear un Contenedor Abstracto. Unir el contenedor abstracto y el Elemento Individual Abstracto por una relación de contención.

Regla 4-2: Cuando una tarea se descompone en otra(s) tarea(s), crear un contenedor abstracto que contenga todas las tareas en las que se descompone la tarea principal.

1.2. Sub-paso: Selección de componentes abstractos individuales.

En este paso se fusiona la información contenida en el modelo de tareas y en el modelo de dominio para producir la especificación de componentes abstractos individuales.

Para poder llevar a cabo esta transformación, hay que añadir información al modelo propuesto, en concreto, a las hojas que forman parte del modelo de tareas. Necesitamos definir una serie de propiedades nuevas, que enriquecen la información que nos proporciona el modelo de dominio, de tal forma que, podamos identificar el comportamiento que tiene que tener dicha hoja, por ejemplo, identificar si la interacción del usuario con el sistema se produce pulsando un botón o introduciendo un texto en una caja, etc.

Para conseguir este objetivo, identificamos cada hoja del árbol de tareas mediante dos nuevas etiquetas: *actionType*, que indica una acción genérica como puede ser *Select* o *Start/go*, e *itemType* que indica el objeto genérico sobre el cual la acción es llevada a cabo como puede ser un elemento o una colección de elementos, etc. De esta forma se definen las distintas facetas del modelos, que no son otra cosa que la especificación del comportamiento de las tareas hojas que forman parte del modelo de tareas, y que servirán para especificar el tipo de elementos que tienen que ser representados en la AUI, CUI y FUI. A continuación se muestra una tabla en la que se identifican según los valores de las propiedades *itemType* y *actionType*, el valor de la faceta resultante.

Task [actionType] +[actionItem]	AIC Facet type + [actionType] + [actionItem]
[Start/go] + [Operation]	[Control]
[Stop/exit] + [Operation]	[Control]
[Start/Go] + [Container]	[Navigation]
[Stop/exit] + [Container]	[Navigation]
[Select] + [Element]	[Input] + ([Select] + [Attribute Value] OR [Select] [Object])
[Select] + [Collection]	[Select] + [Collection] [Input] + ([Select] [Attribute Value Set] OR [Choose] [Object Set])
[Create] + [Element]	[Input] + ([Create] [Attribute Value] OR [Create] [Object])
[Create] + [Collection]	[Input] + ([Create] [Attribute Value Set] OR [Create] [Object Set])
[Delete] + [Element]	[Input] + ([Delete] [Attribute Value] OR [Delete] [Object])
[Delete] + [Collection]	[Input] + ([Delete] [Attribute Value Set] OR [Delete] [Object Set])
[Modify] + [Element]	[Input] + ([Update] [Attribute Value] OR [Update] [Object])
[Modify] + [Collection]	[Input] + ([Update] [Attribute Value Set] OR [Update] [Object Set])
[View] + [Element]	[Output] + ([View] [Attribute Value] OR [view] [object])
[View] + [Collection]	[Output] + ([View] [Attribute Value Set] OR [View] [Object Set])
[Monitor] + [Element]	[Output] + ([Monitor] [Attribute Value] OR [Monitor][Object])
[Monitor] + [Collection]	[Output] + ([Monitor] [Attribute Value Set] OR [Monitor] [Object Set])
[Move] + [Element]	[Input] + ([Move] [Attribute] OR [Move] [Object])
[Move] + [Collection]	[Input] + ([Move] [Attribute Value Set] OR [Move] [Object Set])
[Duplicate] + [Element]	[Input] + ([Duplicate] [Attribute] OR [Duplicate] [Object])
[Duplicate] + [Collection]	[Input] + ([Duplicate] [Attribute Value Set] OR [Duplicate] [Object Set])

Tabla 2: Asociación de tipos de acciones de tareas con facetas AUI

Regla 4-3: para cada elemento abstracto individual asignado a una tarea, tal que, la naturaleza de la tarea consiste en la activación de un método, es decir, el *actionType* de la tarea es de tipo *Start/Go*, y esta tarea es asignada a una clase, asignar al componente abstracto individual que contiene a dicha clase, una faceta de acción que active el método de asignación.

1.3. Sub-paso: Disposición espacio-temporal de objetos de interacción abstractos.

El objetivo de este sub-paso es el de establecer una relación espacio-temporal, entre los distintos elementos que componen la interfaz, es decir, establecer el orden en el cual las distintas tareas tienen que ser llevadas a cabo. Para ello, se utiliza el término "relación de adyacencia abstracta", que nos indica que las tareas que están unidas mediante este tipo de relación están secuencialmente ordenadas.

Regla 4-4: Si existen dos tareas en el mismo nivel (hermanas) secuenciales (">>") y estas tareas tienen asignado un componente abstracto individual, crear una relación de "adyacencia Abstracta" entre esos AIOs.

1.4. Sub-paso: Definición de Control de Diálogo Abstracto. (ADC)

El modelo de tareas define restricciones temporales entre las tareas. Estas restricciones tienen que poder ser representadas a nivel de AUI. Es en este sub-paso donde se añaden estas restricciones a la AUI.

Regla 4-5: para cada pareja de tareas hermanas que tengan asignados AIC y que estén unidas por una relación temporal, crear una relación temporal entre los AIC que las contienen con la misma semántica que la relación temporal que las une en el modelo de tareas.

1.5. Sub-paso: Derivación de AUI a relaciones dominios

En este sub-paso hay que se utiliza el concepto de relación de tipo "*manipulates*", que ha sido descrito con anterioridad.

Regla 4-6: por cada tarea que está unida a un método mediante una relación de tipo "*manipulates*", la AIC que representa esta tarea desencadena el método.

Regla 4-7: si dos tareas hermanas unidas por una relación de tipo "*manipulates*" con el mismo atributo, y estas tareas están unidas en el modelo de tareas mediante una relación temporal ">>", y además cada una de esas tareas ha sido mapeada en un AIC, entonces la AIC que se corresponde con la primera tarea se une mediante una relación de "*updates*" con el atributo "*manipulates*" por las tareas. La segunda AIC se relaciona con este atributo mediante una relación de observación.

Regla 4-8: si dos tareas hermanas unidas por una relación de tipo "*manipulates*" con el mismo atributo, y estas tareas están unidas en el modelo de tareas mediante una relación temporal ">>", y además cada una de esas tareas ha sido mapeada en un AIC, entonces ambas AIC están relacionadas con el atributo mediante relaciones "*update*" y "*observe*".

4.4.2.2 Paso 2: De Interfaz de usuario Abstracta (AUI) a Interfaz concreta de usuario (CUI)

Este paso consiste en generar una interfaz concreta de usuario desde una interfaz abstracta de usuario. Este paso del desarrollo puede tener los siguientes subpasos:

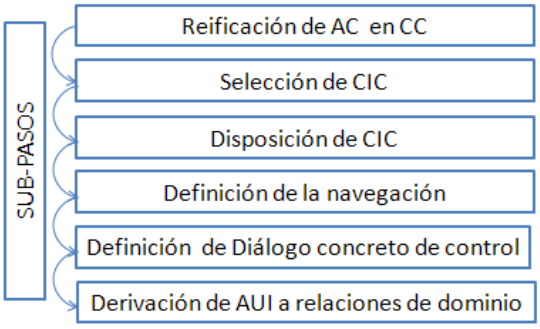


Figura 8: Paso de desarrollo de AUI a CUI

Para generar la CUI se siguen los siguientes pasos: en primer lugar, hay que identificar los contenedores (ventanas, cajas…), después, los elementos individuales son identificados y organizados en los contenedores que se han identificado previamente.

De esta forma, se completa el diseño de la presentación. Después de esto, se añade el diálogo en tres fases: la definición de la navegación, el control de esta navegación y otros comportamientos. Finalmente se establecen las relaciones de domino de CUI transformando las relaciones del modelo AUI.

2.1 Sub-paso: Reificación de contenedores abstractos en contenedores concretos

Un contenedor abstracto puede ser reificado en diferentes tipos de contenedores concretos. Las variables que influyen en esta transformación son: modalidad, contexto de uso, estilo de interacción y las preferencias del diseñador. La mayor dificultad de este paso reside en el problema de elegir un nivel apropiado para agrupar contenedores abstractos en contenedores concretos.

Regla 4-9: Cada contenedor abstracto en cada nivel de hoja que se esté procesando es transformado en una ventana. Observar que un contenedor abstracto es siempre reificado en una caja del nivel concreto. Esta caja está entonces incrustada en una ventana.

Regla 4-10: Cada contenedor abstracto contenido en un contenedor abstracto que es reificado en una ventana es transformado en un área de contenido del tipo horizontal box o caja horizontal e incrustada en una ventana.

2.2 Sub-paso: Selección de componentes individuales concretos.

Las funcionalidades de componentes abstractos individuales son identificadas según sus facetas. La selección de componentes concretos individuales consiste en elegir el elemento concreto que soportará todo o parte de las facetas asociadas con un componente abstracto individual.

Se introduce un nivel intermedio entre el modelo de tareas y la selección de herramientas concretas. Reglas como las siguientes pueden encontrarse en este método: “Para cada atributo en el modelo de dominio generar un campo de entrada cuya etiqueta es el nombre del atributo”.

Regla 4-11: Cada faceta de un componente abstracto individual es reificado por un componente gráfico individual del tipo "componente de texto editable" (ej. text box).

2.3 Sub-paso: Disposición de componentes individuales concretos

Las relaciones entre objetos de interacción abstractos pueden interpretarse para proveer información concreta de la distribución de los objetos. Un esquema abstracto se define con relaciones abstractas de adyacencia. Una relación de adyacencia ya sea concreta (*concreteAdjacency*) o abstracta (*abstractAdjacency*) nos indica que dos objetos de interacción (abstractos o concretos dependiendo del nivel en el que estemos trabajando) son lógicamente adyacentes. Contamos con *concreteAdjacency* para especificar una distribución concreta a nivel de CUI. Combinado con nuestro sistema de cajas, estas relaciones nos permiten definir distribuciones no ambiguas.

Regla 4-12: para cada pareja de componentes abstractos individuales relacionados con una "*abstractAdjacency*" y reificados en componentes concretos individuales, generar una relación "*concreteAdjacency*" entre los componentes concretos individuales.

2.4 Sub-paso: Definición de la navegación

La navegación es definida por un conjunto de transiciones entre contenedores que forman parte de una UI. La navegación es sólo un efecto de la reificación de contenedores abstractos en contenedores concretos.

Regla 4-13: para cada contenedor relacionado con otro contenedor perteneciente a diferentes ventanas, y que sus respectivos contenedores abstractos estén relacionados por "una relación de anterioridad", generar un botón de navegación en el que el contenedor de la fuente apunta a la ventana del contenedor objetivo.

2.5 Sub-paso: Definición de diálogo concreto de control

Este sub-paso consiste en una simple trasposición de relaciones de diálogo abstractas en el mundo concreto. Para ello definimos la relación de diálogo de control,

este tipo de relación permite una especificación del flujo de control entre los distintos objetos de interacción (abstractos o concretos como en anteriores casos).

Regla 4-14: por cada pareja de contenedores abstractos con una relación de diálogo de control, trasponer esta relación a la pareja de contenedores concretos que reifican a los primeros.

2.6 Sub-paso: Derivación de CUI a relaciones de dominio

Este paso consiste en la trasposición de "AUI a relaciones de dominio" a nivel concreto. Se supone una simple propiedad de transacción entre un concepto de dominio, un concepto abstracto y un concepto concreto.

Regla 4-15: por cada AIC unido mediante una relación *Update* con un concepto de dominio, si un CIC deifica esta AIC, entonces el CIC actualiza este mismo concepto de dominio.

4.5 Valoración de la propuesta

Esta solución ha sido implementada de forma computacional por el grupo de investigación CHICO[1] en la herramienta CIAT(GUI).

Una de las ventajas que presenta es que permite la interacción del diseñador entre una fase y otra de las transformaciones, lo que permite hacer cambios en la IU si el diseñador lo considera conveniente sin la necesidad de modificar el árbol CTT de entrada.

Sin embargo, esta propuesta presenta varias limitaciones, la principal es que sólo define las transformaciones para el operador de secuencia ">>" de los árboles de partida. Esto hace que la propuesta no se pueda utilizar para la generación de interfaces que contengan tareas concurrentes. Además, según están definidas las reglas, nos proporciona en cualquier caso sólo una ventana en la que aparecerá toda la IU. Con esta limitación es de esperar que no permita la representación de tareas colaborativas aunque la notación CTT sí permita representarlas. En el siguiente capítulo proponemos una solución a estas limitaciones.

[1] http://chico.inf-cr.uclm.es.

Capítulo 5 Una propuesta basada en ETS

5.1 Introducción

Luyten (Luyten, 2004) propone otra solución a la generación automática de Interfaces de Usuario, en este caso, en lugar de basarse en reglas de transformación entre modelos, se utiliza el concepto de *Enabled Task Sets* (ETS o conjunto de tareas habilitadas) de Paternò (Paternò, 2000). Esta solución sí que contempla tipos de relaciones concurrentes, en concreto, propone un algoritmo para calcular conjuntos de ETS, de esta forma, se consiguen las tareas que tienen que ser ejecutadas en el mismo instante de tiempo, es decir que son mostradas en el mismo momento al usuario.

A continuación vamos a describir la propuesta de Luyten, en primer lugar definimos los modelos que utiliza y a continuación analizamos de forma detallada el algoritmo propuesto para la generación de ETS.

5.2 Propuesta de Luyten

5.2.1 Definición de modelos

Se proponen tres modelos que se consideran el *core* del enfoque, estos modelos son el *Modelo de Tareas*, el *Modelo de Diálogo* y el *Modelo de Presentación*. Además de estos tres modelos también están presentes el modelo de diálogo y de presentación cuya utilidad es la de unir la lógica de la aplicación y de la interfaz por lo que no se consideran modelos principales y cuya definición general se ha analizado en capítulos anteriores.

Los tres modelos *core* son:

1. *Modelo de Tareas*. Describe las actividades y tareas que se dan cuando el usuario interacciona con un sistema de forma estructurada, se expresa en notación CTT. Para describir esta interacción se hace uso de los llamados ETS, que se definen como conjuntos de tareas que están habilitadas para comenzar su funcionamiento en el mismo periodo de tiempo. Para identificar cuáles son estas

tareas, hay que inspeccionar los operadores temporales que relacionan las tareas. Hay que tener en cuenta que una misma tarea puede pertenecer a dos ETS distintos, esto es debido a que la especificación permite tareas concurrentes.

2. *Modelo de Diálogo*. Describe las relaciones que existen entre los ETS y los conceptos representados por este conjunto de tareas. Estos conceptos son análogos a los conceptos del modelo de dominio propuesto por Limbourg y añaden información sobre el comportamiento dinámico de las tareas, como puede ser el orden de ejecución de éstas.
3. *Modelo de Presentación (PM)*. Es la representación más concreta de la IU, es decir, la representación física de ésta. Una *unidad de representación (PU)* agrupa la realización concreta de la IU que puede ser manipulada por el usuario en un instante de tiempo concreto y bien definido. El *Modelo de Presentación* se define por tanto como la notación que describe un conjunto de unidades de presentación que ocurren durante el ciclo de vida de la aplicación.

   La definición de PU puede relacionarse fácilmente con la de ETS: existe una relación uno a uno entre un modelo de presentación y una ETS.

5.2.2 Descripción del método

Partiendo de la representación CTT del modelo de tareas se genera el conjunto de ETS, lo que nos indica que la visualización de las tareas que están en un mismo ETS se tiene que producir en el mismo instante de tiempo. Para calcularlo se realizan dos pasos. Primero se genera el *árbol de prioridad* [59], que se define mediante una especificación CTT, donde todas las relaciones temporales del mismo nivel del árbol en la jerarquía de tareas tienen la misma prioridad de acuerdo a su orden predefinido. Este árbol se obtiene descendiendo recursivamente la especificación CTT del Modelo de Tareas e insertando un nuevo nivel con tareas abstractas donde los operadores temporales de un mismo nivel no tengan la misma prioridad (Tabla 3).

Elección	([])	Mayor prioridad ↕ Menor prioridad
Composición paralela	(|||) (|[]|)	
Interrupción	([>)	
	(|>)	
Habilitación	(>>), ([]>>)	

Tabla 3: Prioridad de operadores

De esta forma se evita la ambigüedad que pueda existir en la representación CTT según los operadores que relacionan las tareas. Hay que destacar en este punto que aunque se utiliza la definición de árbol de prioridad de Paternò, en el caso de Luyten cuando se localizan relaciones ambiguas y hay que insertar un nuevo nivel, el operador que se queda con una profundidad mayor es el que tiene mayor prioridad según la figura anterior. En el caso de Paternò sería el de menor prioridad.

Por ejemplo, en la Figura 12 podemos interpretar la especificación de dos formas: (T1 [] T2) ||| T3 o T1 [](T2 ||| T3).

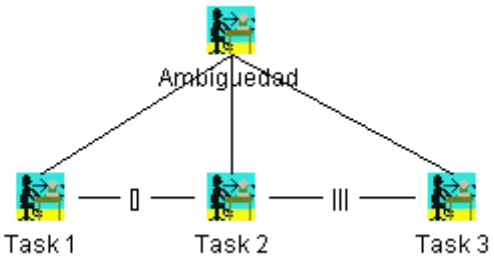


Figura 9: Representación ambigua

Se resuelve creando una tarea abstracta nueva que evita la ambigüedad, según se puede observar en la Figura 13:

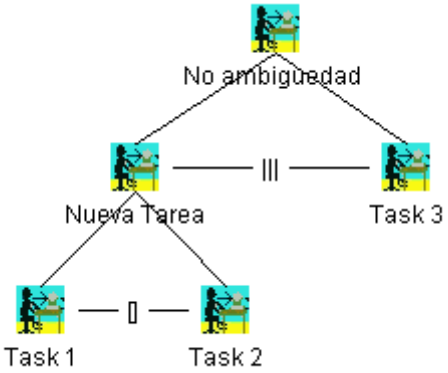


Figura 10: Representación no ambigua

Una vez que obtenemos el árbol de prioridad, el siguiente paso es calcular el conjunto de ETS. Para ello Luyten propone un algoritmo. Este algoritmo se basa en una serie de funciones que proporcionan la información necesaria de las tareas de la especificación CTT.

A continuación se detallan dichas funciones, donde t se corresponde con la tarea que está siendo procesada en un instante de tiempo por el algoritmo:

first(t): devuelve el conjunto de subtareas de t que deberían ejecutarse primero.

body(t): devuelve el conjunto de subtareas de t que no están incluidas en *first*(t).

parent(t): devuelve la tarea padre de t.

children(t): devuelve las tareas hijo de t. Se puede definir como: *children*(t) = *first*(t) U *body*(t).

necessaryTask(S,t), donde S es un ETS y t pertenece al Modelo de Tareas, devuelve un ETS actualizado que contiene las tareas de S que, estando relacionadas mediante una operación temporal "|||" ó "[>", o tienen un padre común con t (el padre de t es hermano de la tarea del ETS), o t es una tarea del mismo nivel que la tarea del ETS.

El ETS se inicializa con la tarea principal del árbol de prioridad; a continuación se usan las funciones descritas anteriormente, dependiendo de los operadores que relacionen cada tarea, y se va completando el ETS. Cuando se llega a la condición de parada, se eliminan del ETS todas las tareas que hayan sido especificadas en el árbol CTT como Tareas de Usuario, ya que éstas no se muestran en la IU.

El algoritmo se corresponde con el siguiente pseudocódigo:

```
1. El conjunto inicial de tareas habilitadas ETS contiene un solo
ets: ets1 = {tr}, que sólo contiene la tarea raíz tr.
```

2. WHILE exists(d) € ets_i, WHERE 1 <*i < #ETS; and (children(d)<> Ø) V first(d) V body(d)*

(a) IF *(d* is a leaf) do:

If*(lfirst(d)):* remove the *lfirst* label from d

Else If(*lbody*(d)): ETS ← ETS \ ets_i

(b)If *(children(d)*<> Ø), take the next level in the tree in which *children(d) = {*n_1,n_2,…,n_m*}* participate:

If $(n_1 \overset{[]}{\rightarrow} n_2)$ (choice):

ets_k ← (ets_i \ {d}) U {n_1, n_2,...,n_m} and ETS ← (ETS \ ets_i) U ets_k

For all node € {n_1, n_2,...,n_m} make a new ETS and add it to the set of enabled task sets:
ETS ← ETS U { {*lbody*(n_1) U ets_i } U ... U {*lbody*(n_m) U ets_i } }

If $(n_1 \overset{|||}{\rightarrow} n_2 \vee n_1 \overset{|[]|}{\rightarrow} n_2)$ (parallell composition):

Replace d with children(d) in ets_i: ets_i ← (ets_i \ {d}) U {n_1, n_2,...,n_m}

If $(n_1 \overset{[>}{\rightarrow} n_2 \vee n_1 \overset{|>}{\rightarrow} n_2)$ (interrupt):

ets_k ← *lbody*(n_2) U (ets_i \ {d})

ets_i ← {n_1} U *lfirst*(n_2) U ets_i \ {d}

ETS ← ETS U ets_k

If $(n_1 \overset{>>}{\rightarrow} n_2 \vee n_1 \overset{[]>>}{\rightarrow} n_2)$ (enabling):

- If(*lbody*(d))

  ETS ← ETS U {{n_1 U necessaryTasks(ets_i,d)} U...U { {n_m U necessaryTasks(ets_i, d)}

  ETS ← ETS \ ets_i

  - Else If (*lfirst*(d))

  ets_k ← lfirst(n1) U (ets_i \ {d})

  ETS ← ETS U {{*lbody*(n_1)} U necessaryTasks(ets_i,d)}

  ETS ← ETS \ etsi

  Else

  ETS ← ETS U {(ets_i \ {d}) U {n_1}} U... U {(ets_i \ {d}) U {n_m)}

```
ETS ← ETS \ {ets_i}
Eliminar todas las tareas de usuario porque no se muestran en la IU
```

Cuando el algoritmo acaba y se obtienen los ETS, éstos se pueden transformar al modelo de diálogo, lo que corresponde con encontrar todas las posibles combinaciones de unidades de presentación. Cada ETS puede ser representado por una o varias unidades de presentación, tal que cada ETS presenta una UI integrada.

A continuación se define el comportamiento de de la IU a nivel de *Modelo de Diálogo*. Para ello utiliza el concepto de *State Transition Network* (STN), se trata de un grafo dirigido donde cada nodo se corresponde con un ETS y los nodos que los unen proporcionan la navegación entre los ETS (navegación entre ventanas). Propone un método para el cálculo de dicha navegación. Este modelo de diálogo presenta un inconveniente para nuestra propuesta debido a que sólo es válido en el caso de que no haya tareas colaborativas. Por este motivo no vamos a entrar en más detalle. En resumen, una vez definido el STN quedaría definido el comportamiento de la IU.

Una vez obtenido el Modelo de Diálogo el siguiente paso es la obtención del Modelo de Presentación o Interfaz de Usuario final, que, como en la propuesta descrita anteriormente, depende de la plataforma y el lenguaje concretos de la IU.

5.3 Obtención de Interfaces de Usuario Concretas a partir de ETS

La generación de IU que propone Luyten, resuelve la limitación más importante que presentaba la propuesta de Limbourg. En la propuesta de Luyten, sí se contemplan todos los operadores disponibles de la especificación CTT y se definen claramente las distintas agrupaciones de elementos que tienen que existir en la IU. Sin embargo, consideramos los modelos AUI y CUI propuestos por Limbourg, útiles para la generación automática de las IU. Estos modelos y las transformaciones entre ellos ya han sido probados de forma computacional en la herramienta CIAT(GUI), por lo que podemos afirmar que, en la forma en la que están definidos, funcionan.

Por lo tanto, proponemos una solución a las limitaciones que en nuestro estudio hemos detectado en la propuesta de generación de IU mediante reglas de transformación

de Limbourg, implementadas en la herramienta CIAT(GUI). Nuestra solución propone el uso del algoritmo de generación de ETS elaborado por Luyten.

Proponemos en primer lugar la modificación de la definición del modelo de Interfaz de Usuario Abstracto de Limbourg descrito anteriormente. Tendremos entonces dos tipos nuevos de objetos abstractos:

- El objeto *Abstract Task Subset* (ATS), que representa un conjunto de tareas que se visualizan en el mismo instante de tiempo. Cada uno de estos objetos se corresponde con los ETS calculados en mediante el algoritmo y contendrá una o varias tareas.
- El objeto *Task*, que representa una tarea del sistema.
- El objeto *Abstract Enabled Task Set* (AETS), que representa la agrupación de uno o varios ATS.

De esta forma la especificación IU a nivel abstracto podría definirse bien en términos de relaciones abstractas y de objetos de interacción abstractos, en el caso de que el árbol de entrada sólo tenga relaciones de tipo “>>” o en términos de *Enabled Task Sets* en el caso de que el árbol de entrada tenga otro tipo de relaciones.

En concreto, las transformaciones a llevar a cabo se dividirían en dos pasos generales:

1. Generación de la Interfaz de Usuario Abstracta

   Partiendo del árbol de tareas en notación CTT los pasos a seguir serían los siguientes: en primer lugar se comprueban las relaciones entre las tareas representadas. Si sólo existen relaciones de tipo “>>” se utilizan las reglas de transformación ya implementadas. Si no, habría que generar el árbol de prioridad. En el caso de que existan en el mismo nivel del árbol tareas relacionadas mediante operadores con distinta prioridad (Tabla 8), se genera el árbol de prioridad, si esta situación no se da el árbol de prioridad coincidiría con el árbol original de entrada. Una vez hemos obtenido el árbol de prioridad se calculan los ETS, mediante el algoritmo propuesto por Luyten, que formarán parte del modelo AUI. De esta forma, tendríamos las tareas representadas en el modelo de tareas que tienen que ser mostradas al usuario en el mismo instante o también denominada interfaz agrupada.

2. Generación de la Interfaz de Usuario Concreta

Partiendo del modelo AUI, en concreto del modelo ETS generaríamos la CUI. Para realizar esta transformación hemos definimos tres nuevas reglas, basadas en as propuestas por Limbourg) para transformar los Task, ATS y AETS definidos anteriormente en objetos de tipo concreto, como pueden ser ventanas, campos de texto, etc.

Las reglas que proponemos son:

Regla 1. Un objeto AETS se transformará en un Contenedor Concreto de tipo ventana. Este objeto concreto sirve para agrupar los elementos que se visualizan a la vez por el usuario.

Regla 2. Un objeto ATS se transformará en un Contenedor Concreto de tipo caja, que se incrustará dentro del Contenedor Concreto de tipo ventana correspondiente (obtenido por la Regla 1). Mediante esta regla se agrupan dentro del mismo contenedor de tipo caja los objetos Tarea de un ATS.

Regla 3. Un objeto Tarea se transformará en uno o varios componentes gráficos individuales (como por ejemplo un "componente de texto editable"). El tipo concreto del componente gráfico individual dependerá de los valores *Action Type* y *Action Item* y del tipo de datos asociados al objeto Tarea. Por ejemplo, si *Action Item* tiene el valor de Element y *Action Type* el valor de Select el objeto Tarea se transforma en un "componente de texto editable".

Con esta última regla identificamos los elementos individuales que forman parte de la CUI. Existirá al menos un elemento concreto individual por cada una de las tareas del Modelo de Tareas (excepto tareas de tipo usuario).

Con la aplicación de estas tres reglas de transformación que proponemos obtenemos la interfaz de usuario a nivel concreto.

Para definir componentes gráficos individuales, se utiliza la información del modelo de *mapping*, de esta forma, según este modelo podemos definir si un componente se traba de un campo de texto, un botón, etc. Se generarían además relaciones contenedoras entre los componentes individuales y los componentes que los contienen.

Capítulo 6. Aplicación de la propuesta: mejora de CIAT(GUI)

6.1 Introducción

Para la implementación de nuestra propuesta hemos tenido que estudiar las distintas tecnologías que utiliza CIAT(GUI) para poder integrar los cambios en ella. En este capítulo realizamos una breve introducción a dichas tecnologías. Además abordamos las modificaciones hechas en CIAT(GUI) teniendo en cuenta que éstas pueden dividirse en dos bloques, por una lado el diseño e implementación del algoritmo propuesto por Luyten y por otro la modificación del editor gráfico para que pueda representar los ETS. Concretamos además el escenario del problema con el fin de aclara los requisitos que tiene que tener la nueva herramienta.

6.2 Tecnologías

A continuación describiremos brevemente las tecnologías utilizadas en el presente trabajo.

6.2.1 Eclipse

Eclipse[2] es una herramienta que permite integrar diferentes aplicaciones para construir un entorno integrado de desarrollo (IDE o *Integrated Development Environment*). Es una plataforma de desarrollo de software abierto (*open-source*), que está dividida en tres partes: *Eclipse Project*, *Eclipse Tools Project* y *Eclipse Technology Project*. El componente principal de *Eclipse Project* es JDT (Java Development Tools). Éste permite crear aplicaciones en Java. Además, Eclipse proporciona mecanismos para integrar otras aplicaciones en forma de *plug-ins*. Estos plug-ins son reconocidos automáticamente por Eclipse cuando se inicia. Dado que Eclipse está desarrollado en Java, para su funcionamiento se debe tener instalado el JRE (*Java Runtime Enviroment*). Una vez instalado, Eclipse detecta automáticamente la ubicación del JRE.

[2] http://www.eclipse.org.

6.2.2 Eclipse Modeling Framework

El *Eclipse Modeling Framework*[3] (EMF) es un entorno de desarrollo para el desarrollo de herramientas y aplicaciones basadas en un modelo de datos estructurado. Proporciona una estructura de modelado de tal forma que permite la especificación de los datos de la aplicación mediante un Diagrama de Clases subconjunto de UML.

Los modelos EMF se pueden definir de tres formas: mediante Java Interfaces, Diagramas de clases UML y XML Schema. Una vez definido el modelo mediante una de las tres formas, se pueden generar de las demás. En la Figura 16 podemos observar la definición del model EMF.

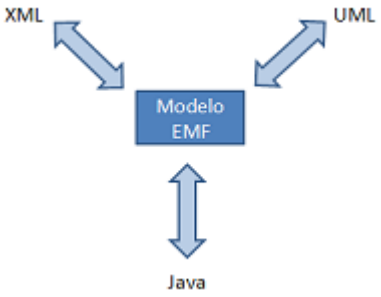


Figura 11 Definición del modeo EMF

Estas tres formas proporcionan la misma información, lo que varía es la forma de visualización o representación de los datos.

Para la realización de la herramienta CIAT(GUI) así como para la ampliación llevada a cabo en los modelos que forman parte de ella, se ha utilizado la representación mediante diagramas UML, usando el llamado *Ecore Tools graphical editor*, de tal forma que se crea el denominado metamodelo de los datos o ECore. Una vez que se han definido los modelos del sistema, EMF proporciona un motor de generación del código del objeto, de tal forma que se producen una serie de clases Java en base a ese modelo, un conjunto de clases Adapter, que permiten su visualización y edición basándose en comandos del modelo, y un editor básico.

La arquitectura de EMF puede observarse en la siguiente figura:

[3] http://www.eclipse.org/modeling/emf/.

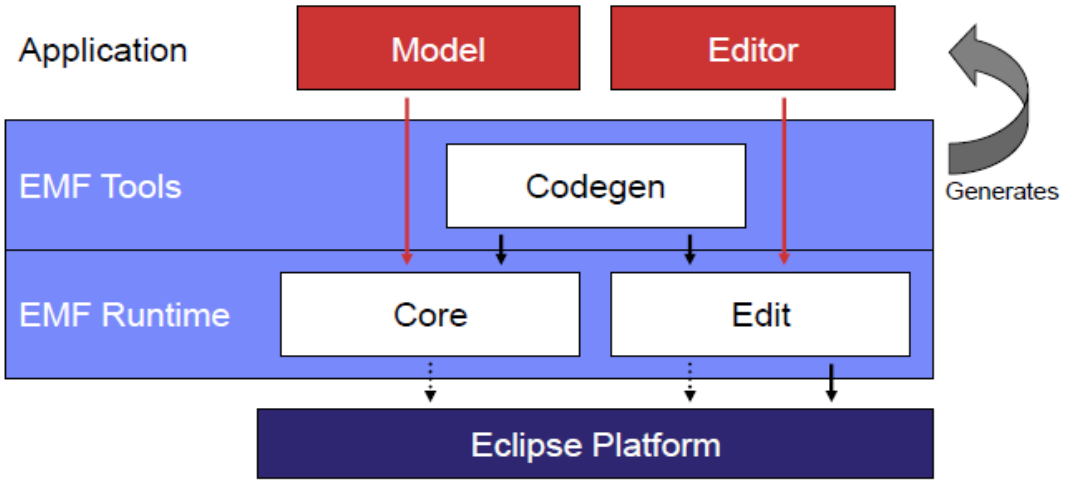


Figura 12: Arquitectura EMF

EMF permite importar modelos que han sido previamente especificados usando documentos Ecore.

Una de las características más importantes de EMF es que suministra mecanismos de interoperabilidad con otras herramientas y aplicaciones basadas en EMF.

EMF utiliza XMI (*XML Metada Interchange*) que es un estándar de la OMG que mapea MOF a XML. XMI define cómo deben ser usadas las etiquetas XML que son usadas para representar modelos MOF. Los metamodelos MOF son convertidos en DTD (*Document Type Definitions*) y los modelos son convertidos en documentos XML que son consistentes con su DTD correspondiente. XMI resuelve muchos de los problemas encontrados cuando intentamos utilizar un lenguaje basado en etiquetas para representar objetos y sus asociaciones, y además el hecho de que XMI esté basado en XML significa que tanto los metadatos (etiquetas) y las instancias que describen (elementos) pueden ser agrupados en el mismo documento, permitiendo a las aplicaciones entender rápidamente las instancias por medio de los metadatos.

6.2.3 Graphical Modeling Framework

Graphical Modeling Framework[4] (GMF) proporciona una herramienta para la generación de editores gráficos basados en EMF y *Graphical Editing Framework* (GEF). Mediante GEF el desarrollador puede crear un editor gráfico completo de forma rápida a partir de un

[4] http://www.eclipse.org/modeling/gmp/.

modelo de una aplicación. GMF proporciona un editor gráfico e intuitivo para la creación de objetos conformes a los modelos creados mediante EMF.

Una característica destacable en GMF es la reutilización de la definición gráfica para diferentes dominios y aplicaciones, esto es, se pueden reutilizar las metáforas gráficas ya definidas para conceptos de diferentes dominios y aplicaciones. Esta característica se consigue modelando por separado las componentes gráficas que se corresponden con cada uno de los elementos del dominio y la definición de la paleta de herramientas, la cual tendrá una herramienta por cada primitiva. Para completar el proceso de generación de un editor gráfico de dominio, GMF proporciona una definición de *mapping* o correspondencia mediante la que se asocia cada primitiva de modelado con su componente gráfica y con su herramienta dentro del editor que se está generando.

Se describirá más detalladamente las funciones de GMF en el capítulo 3 de este documento.

6.2.4 Atlas Transformation Language

Atlas Transformation Language[5] (ATL) es un lenguaje y una máquina virtual para la transformación entre modelos. Se trata de un componente del proyecto Eclipse Model-to-Model (M2M) dentro del *Eclipse Modeling Project* (EMP). ATL es un lenguaje de transformación de modelos híbrido que permite, en su definición de transformaciones, especificar construcciones declarativas e imperativas. La propuesta es del ATLAS Group del INRIA & LINA, de la Universidad de Nantes y fue desarrollado como parte de la plataforma AMMA (ATLAS *Model Management Architecture*).

Otros detalles de la Definición del lenguaje:

- Es compatible con los estándares de la OMG: es posible describir transformaciones modelo a modelo y se basa en QVT.
- Permite definir pre y post condiciones.

En cuanto a las reglas de transformación, ATL define un dominio (metamodelo) para el modelo origen (*source*) y otro dominio para el modelo destino (*target*). *Source* y *target* pueden tener iguales o diferentes dominios (aunque sean iguales, ambos deben ser claramente identificados). Si bien las transformaciones son unidireccionales, ATL

[5] http://www.eclipse.org/atl/.

permite la definición de transformaciones bidireccionales como la implementación de dos transformaciones, una para cada dirección.

Como características particulares del lenguaje, se pueden mencionar las estructuras que define este lenguaje. En primer lugar, la definición de transformaciones forman módulos (modules) que contienen las declaraciones iniciales y un número de *helpers* y reglas de transformación. Los *helpers*, son una estructura intermedia dentro de las transformaciones que facilitan la navegación, la modularización y el reuso. Permiten definir operaciones. Existe también una construcción llamada *called rule,* y que representa a un *procedure*. Estas pueden contener argumentos y pueden ser invocadas por su nombre.

La aplicación de las reglas se realiza de forma no determinística, no se ha provisto ninguna construcción o claúsula que permita aplicar en forma condicional las reglas. Cabe mencionar que la invocación de *called rules* es determinística. Esta invocación, junto con la utilización de parámetros permite soportar recursión.

ATL provee modularización por sus procedimientos o called rules, además de que sus módulos pueden incluir a otros; y mecanismos de reuso ya que las reglas se pueden heredar.

6.2.5 Meta-Object Facility

Meta-Object Facility[6] (MOF) es un estándar creado por la OMG, que extiende UML para que éste sea aplicado en el modelado de diferentes sistemas de información. El estándar MOF define diversos metamodelos, esencialmente abstrayendo la forma y la estructura que describe los metamodelos. MOF es un ejemplo de un meta meta modelo o un modelo del metamodelo orientado a objetos por naturaleza. Define los elementos esenciales, sintaxis y estructuras de metamodelos que son utilizados para construir modelos orientados a objetos de sistemas. Además, proporciona un modelo común para los metamodelos de CWM y UML.

El gran potencial de MOF reside en que permite interoperar entre metamodelos muy diferentes que representan dominios diversos. Las aplicaciones que utilizan MOF no tienen por qué conocer los interfaces del dominio específico de algunos de sus

[6] http://www.omg.org/mof/.

modelos, pero pueden leer y actualizar los modelos utilizando las operaciones genéricas y reflexivas ofrecidas en los interfaces.

6.2.6 ConcurTaskTrees

ConcurTaskTrees (CTT) (Paternò et al. 1997) es una notación que permite realizar la especificación del modelado de tareas. Se ha desarrollado con el objetivo de mejorar las limitaciones de las notaciones existentes a la hora diseñar aplicaciones interactivas. Este modelo es ampliamente aceptado y se ha convertido en un estándar de facto en cuando al modelado de tareas de usuarios. CTT aporta una notación expresiva y flexible capaz de representar actividades concurrentes e interactivas, además, ofrece la posibilidad de apoyar las cooperaciones entre varios usuarios y posibles interrupciones.

Su objetivo principal es ser una notación fácil de usar y que pueda apoyar el diseño de aplicaciones industriales reales, es decir, aplicaciones de dimensiones medianas y grandes. Las principales características que presenta son: estructura jerárquica, lo que hace que sea muy intuitivo; proporciona una sintaxis gráfica, lo que lo hace más fácil de interpretar; es una notación concurrente y está enfocada a las actividades.

6.3 Modificación de CIAT(GUI)

A continuación detallamos las modificaciones realizadas en CIAT(GUI).

6.3.1 Escenario e identificación del problema

Se requiere ampliar la funcionalidad CIAT(GUI) mediante una herramienta que implemente el algoritmo de Luyten. El desarrollador de IU interactivas dispondrá de un editor gráfico, mediante el cual especificará la interacción del usuario de la aplicación con la propia aplicación en notación CTT, el modelo de dominio del sistema y el modelo de mapping. Una vez especificada la IU mediante estos tres modelos, se obtendrá la Interfaz de Usuario Abstracta y la Interfaz de Usuario Concreta simplemente pulsando un botón. El sistema debe tener la capacidad de detectar las relaciones que existen entre las tareas que forman parte del Modelo de Tareas, de tal forma que, en caso de que sólo aparezcan relaciones de tipo “>>” se utilizará la implementación de la propuesta de Limbourg (ya implementada por CIAT(GUI)) y en el caso de que existan otro tipo de relaciones se utilizará la propuesta que aquí se presenta.

Tal y como se ha comentado con anterioridad, la modificación a realizar se puede dividir en dos partes, por un lado tenemos la implementación del algoritmo propuesto por Luyten para la generación de interfaces de usuario, y por otro la ampliación del editor gráfico que forma parte de la herramienta CIAT(GUI) para que ésta pueda representar de forma visual la representación de IU que conseguimos mediante el algoritmo de Luyten.

En primer lugar analizaremos el modelo y la representación gráfica que debe ser usada por la herramienta y en segundo lugar analizaremos el algoritmo propuesto por Luyten.

6.3.2 Implementación del algoritmo

El algoritmo que propone Luyten, consta en realidad de dos algoritmos independientes: por un lado, tenemos el algoritmo que comprueba si en el modelo de tareas de entrada existe ambigüedad y por otro lado el algoritmo que calcula los ETS a partir de este árbol de prioridad.

Si analizamos el algoritmo que genera el árbol de prioridad notaremos que se realiza un recorrido en profundidad dentro del árbol y que a la vez que se realiza dicho recorrido se van analizando las hojas a las que está unido cada nodo. Parece entonces que el diseño puede llevarse a cabo mediante técnicas de diseños de algoritmos denominadas “backtracking” o “vuelta atrás”. En su forma básica, la idea de backtracking se asemeja a un recorrido en profundidad dentro de un grafo dirigido. El grafo en cuestión suele ser un árbol, o por lo menos no contiene ciclos. Siguiente esta técnica de diseño podremos recorrer el árbol y comenzando por el nivel de profundidad mayor se irá resolviendo la ambigüedad entre las tareas que componen el árbol. Por lo tanto la estructura para la representación del modelo de tareas se corresponderá con un grafo dirigido.

El algoritmo que propone Luyten está especificado en pseudocódigo por lo que nuestro trabajo consiste en implementarlo utilizando la estructura de datos que hemos elegido. La estructura de paquetes de nuestra aplicación se puede observar en la siguiente figura:

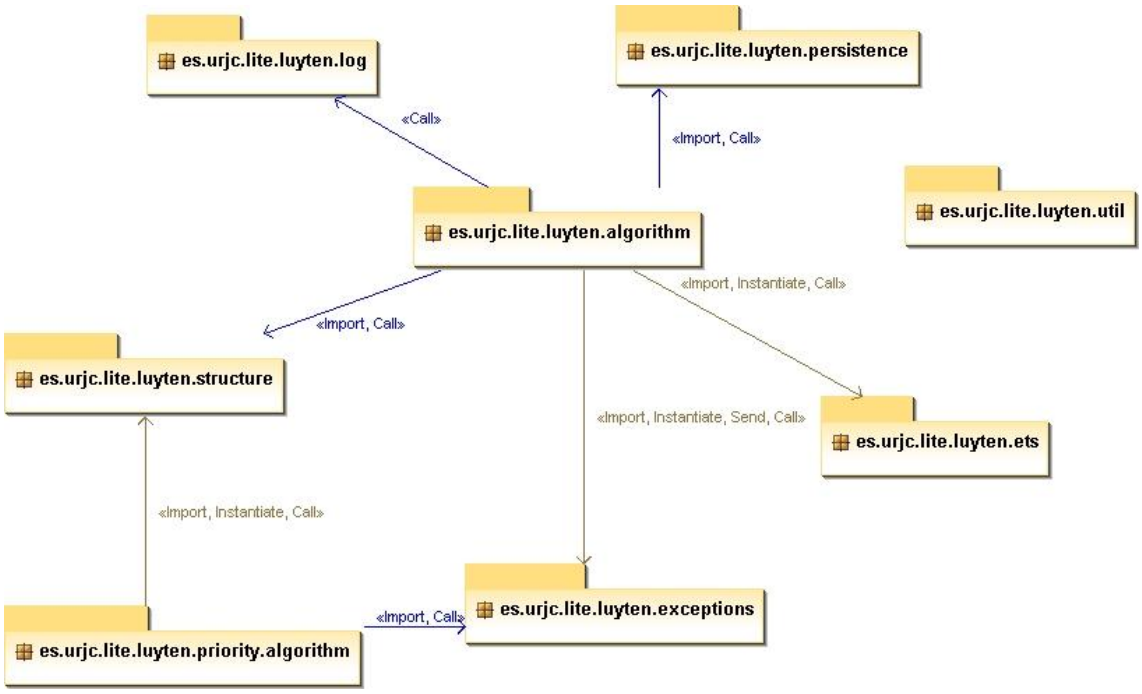


Figura 13: Estructura lógica de paquetes

6.3.3 Modelo y representación gráfica

En el problema a resolver el modelo de datos a desarrollar es el de *Enabled Task Set*, la definición de ETS es el de conjuntos de tareas habilitadas, esto es, conjuntos de tareas que tienen que estar habilitadas para el usuario en el mismo instante de tiempo.

La herramienta que desarrollaremos debe disponer de un editor gráfico que permita la visualización de los conjuntos de tareas resultantes. Por lo tanto, se debe desarrollar como parte de la herramienta, el editor gráfico para la visualización de los ETS de tal forma que quede integrado con el editor gráfico de CIAT(GUI). Si analizamos nuestra herramienta, los ETS resultantes formarán parte de la denominada Interfaz de usuario Abstracta, por lo tanto, a la hora de implementar el editor gráfico de ETS hay que tener en cuenta, que esta representación debe aparecer dentro de las herramientas del grupo AUI, tal y como se puede observar en la Figura 18:

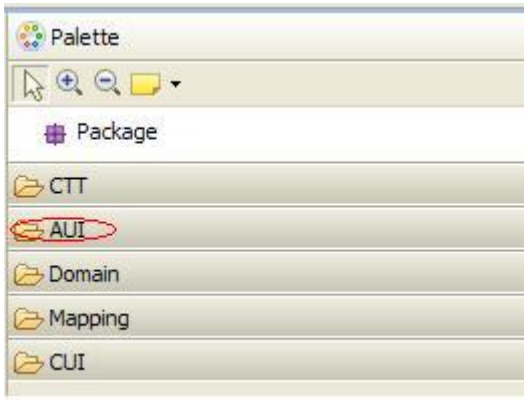


Figura 14: Paleta del editor gráfico de la herramienta

Para añadir la representación de los ETS tenemos que modificar el metamodelo ya existente. Es decir añadir a la estructura que ya existe una nueva representación. Ampliamos entonces el metamodelo chico para que pueda soportar la representación de datos de los ETS.

La representación mediante diagramas de clase del modelo ETS se puede observar en la siguiente figura:

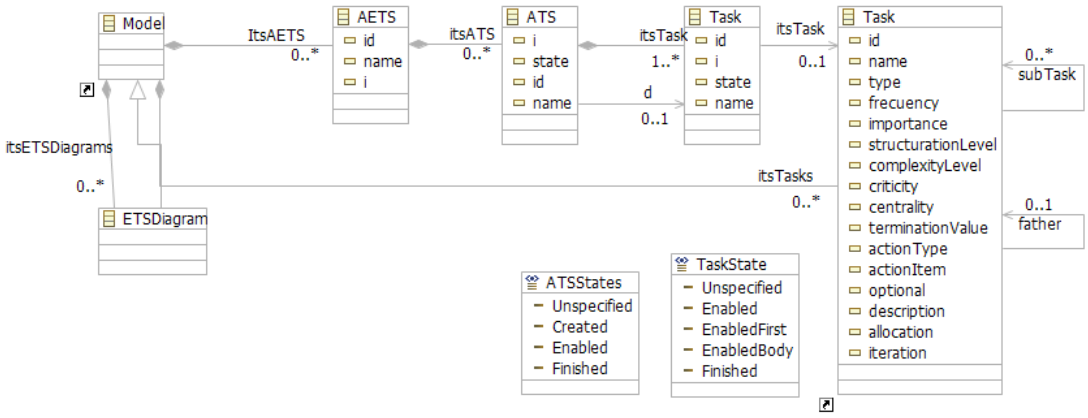


Figura 15: Diagrama de clases del modelo ETS

Una vez que hemos definido el metamodelo del sistema, el siguiente paso se corresponde con la generación del código del sistema. Gracias al framework EMF con el que estamos trabajando, la generación del código del modelo de datos que utiliza nuestra aplicación se hace de forma automática. Además del código fuente del modelo de datos, generamos el código de los Editores no Gráficos de la herramienta.

Una vez definido el código del modelo de datos de la herramienta, así como los editores no gráficos de los ETS, el siguiente paso consistiría en el diseño de la herramienta gráfica para la representación de ETS. Aunque los ETS se generarán de forma automática por la herramienta y no directamente por el usuario de la aplicación, consideramos importante que el usuario disponga de una representación gráfica de los ETS que permita visualizar de forma esquemática el resultado del algoritmo.

El diseño y creación del editor gráfico se realizará mediante la tecnología GMF la cual permite la creación de una herramienta para la edición gráfica. Se representarán los ETS de forma que, cada ETS estará formado por un subconjunto ETSi que tendrá una forma rectangular y dentro de cada uno de estos ETSi se mostrarán el nombre de la tarea que forma parte de él.

6.3.4 Transformaciones entre modelos

Si nos fijamos en primer lugar, en el algoritmo de generación de ETS, se trata en de lo que en ingeniería basada en modelos se denomina *Transformación entre modelos*, ya que a partir de un o unos modelos de entrada se genera un modelo de salida, según una serie de reglas de transformación preestablecidas. Además si nos fijamos en el pseudocódigo de éste (Sección 5.2.2) al tratarse de un algoritmo iterativo (no recursivo) todo parece indicar que en lugar del diseño del algoritmo propiamente dicho, se debería traducir a una serie de reglas de transformación que se pudieran implementar mediante algún tipo de lenguaje QVT como es ATL. Sin embargo, nos encontramos con varios problemas, el principal de todos ellos es que mediante el lenguaje ATL, que es utilizado por CIAT(GUI), resulta imposible indicar por cuál de los elementos del sistema tiene que comenzar a realizar las transformaciones, es decir, no podemos indicar el elemento por el cual tiene que empezar a operar, esta operación es fundamental ya que el conjunto de ETS se inicializa con la tarea raíz del árbol en notación CTT y va proporcionando una serie de resultados dependiendo de los operadores que unan a los hijos de éste. Además, el algoritmo que genera el árbol de prioridad, es un algoritmo que también necesita conocer a priori el orden de cada tarea. Además como se ha comentado en el apartado anterior parece lógico, que utilizará recursividad para recorrer la estructura de datos de entrada, lo que imposibilita también de alguna forma el uso de lenguajes declarativos como ATL para realizar este tipo de operaciones.

Por lo tanto, se ha decidido implementar la solución mediante el lenguaje de programación Java, con este lenguaje se puede realizar prácticamente cualquier tipo de programa, además el framework EMF, nos proporciona una serie de funcionalidades que nos permiten acceder a los modelos declarados, así como modificarlos y guardar un nuevo modelo o modificar el existente, en forma de documento XMI.

Capítulo 7 Conclusiones y trabajos futuros

7.1 Logros alcanzados

En este último capítulo se expone un resumen de los puntos más destacados del proyecto, así como las conclusiones obtenidas en los estudios que se han llevado a cabo. Además se propondrán líneas futuras de cara a la ampliación del estudio.

El objetivo del proyecto es encontrar una solución total o parcial al diseño y desarrollo de interfaces de usuario mediantes herramientas MBUID, centrándonos en CIAT(GUI). Para ello hemos estudiado diferentes tecnologías y metodologías existentes en la literatura. Nos hemos centrado en una de ellas, para poder proponer posibles soluciones o mejoras para conseguir nuestro objetivo.

El estudio realizado sobre herramientas MBUID nos ha permitido comprobar que a día de hoy no existe ninguna herramienta que permita el desarrollo de IU interactivas y colaborativas de manera automática. A pesar de esto, cabe destacar, en primer lugar, la herramienta Teresa que parece ser la más extendida y aceptada en estos momentos y que permite la generación de IU para distintos dispositivos. Se trata de una herramienta de libre distribución disponible en http://giove.isti.cnr.it/tools/TERESA/.

Por otro lado tenemos la herramienta CIAT(GUI), cuyo punto fuerte es que además de tratarse de una herramienta MBUID propiamente dicha, proporciona soporte a todo el proceso de desarrollo y especificación de sistemas colaborativos. Es por este motivo por el que nuestro estudio se ha centrado es resolver las limitaciones de ésta, que presenta un potencial que no tienen otras de las herramientas. Esta herramienta genera la IU en código XAML, cuyas cualidades ya han sido comentadas, y la representación interna de los modelos está implementada en UsiXML. Este hecho parece lógico ya que si nos fijamos en los modelos con los que trabaja CIAT(GUI) y los modelos que se pueden especificar con UsiXML podemos ver que son los mismos (presentación, tareas, diálogo y dominio). Además de generar el aspecto visual de la aplicación, se genera el esqueleto del modelo de datos de la aplicación en C#, algo novedoso en cuanto a este tipo de herramientas ya que, en general, sólo se encargan de generar el código de la IU.

En este sentido podemos destacar que mediante esta herramienta se unifica el proceso de diseño y desarrollo de IU con las aplicaciones, cuya importancia se ha comentado en el Capítulo 2 del presente proyecto.

Sin embargo la herramienta presenta varias limitaciones importantes. La principal es que sólo permite generar aplicaciones a partir de IU cuya representación mediante el modelo de tareas sólo contenga el operador de tipo secuencial (“>>”). Esta situación hace que la propuesta no se pueda utilizar para la generación de interfaces que contengan tareas concurrentes, por lo que no permite la representación de tareas colaborativas. En este punto, hemos analizado otra de las propuestas existentes, en concreto la propuesta de Luyten, que proporciona un algoritmo que permite calcular *Enabled Task Sets.* Estas representaciones permiten agrupar las tareas existentes en el modelo de tareas que tienen que ser presentadas al usuario en un mismo instante de tiempo. Hemos modificado dicha herramienta con la solución propuesta que se ha integrado perfectamente en ella.

Finalmente, no se ha implementado el algoritmo de generación del árbol de prioridad por falta de tiempo, aunque se ha realizado un análisis minucioso de éste así como su diseño por lo que se puede establecer el desarrollo de éste como un hito a corto plazo.

Cabe destacar también en este punto, que aunque no era uno de los objetivos de este trabajo, se ha conseguido a través de la herramienta desarrollada, la generación de interfaces de usuario finales en notación XAML y la estructura de datos de ésta en C#. Al integrar el nuevo módulo de la herramienta con CIAT(GUI), se puede utilizar la funcionalidad existente en esta herramienta para obtener interfaces de usuario finales partiendo de la interfaz de usuario concreta que se obtiene con nuestra herramienta

Como conclusión de nuestra investigación, proponemos que combinando ambas propuestas se puede solucionar la limitación más importante de la herramienta CIAT(GUI), ya que ampliando el modelo de representación abstracta de la interfaz con los ETS mencionados se consiguen interfaces representadas como IU concretas que contemplan tareas concurrentes.

De esta forma, aunque no se haya conseguido la generación automática de IU colaborativos, podemos afirmar que se ha dado un paso más para lograrlo y por lo tanto valoramos que se han conseguido los objetivos generales del proyecto marcados en el capítulo 1.3.

7.2 Posibles trabajos futuros

Una vez analizados los resultados, una futura línea de investigación sería encontrar una solución a la navegación entre las pantallas resultante mediantes la propuesta de Luyten que contemple aspectos colaborativos de las interfaz. Luyten propone una navegación *"inter-window"* en su propuesta, de forma experimental y que no contempla aspectos colaborativos, por lo tanto, analizar esta solución para lograr que contemple aspectos colaborativos y poder establecer un método sistemático para lograrlo podría ser uno de los hitos futuros.

También cabe destacar la implementación del algoritmo que calcula el árbol de prioridad del sistema. Además de este punto estaría la posibilidad de estudiar una forma para realizar las transformaciones entre modelos mediante transformaciones ATL en lugar de mediante transformaciones en código Java de forma que todas las transformaciones que realice la herramienta CIAT(GUI) se realicen mediante el mismo tipo de transformaciones.

Por último, si nos fijamos en el manual de usuario podemos apreciar que el uso de la herramienta puede resultar un poco complejo para usuarios sin experiencia, hemos podido comprobar que el tiempo de aprendizaje de uso de la herramienta no es corto, por lo que un posible trabajo futuro podría ser la ampliación de la herramienta de cara a que el usuario tenga que realizar menos acciones o que éstas sean más sencillas.

Referencias

(Berti et al, 04) Berti, S., G. Mori, F. Patern and C. Santoro, "A transformation-based environment for designing multi-device interactive applications," presented at Proceedings of the 9th international conference on Intelligent user interfaces, Funchal, Madeira, Portugal, (2004), 352-253, publisher: ACM, 1-58113-815-6.

(Clerckx et al, 04) Clerckx, T., Luyten**,** K., Coninx, K. "The Mapping Problem Back and Forth: Customizing Dynamic Models while preserving Consistency".TAMODIA 2004: 33-42.

(Coninx et al, 2003) Karin Coninx, Kris Luyten, Chris Vandervelpen, Jan Van den Bergh, Bert Creemers: Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems. Mobile HCI 2003:256-270

(Díaz, 2010) Díaz García, L. "Problemas en la transformación de interfaces abstractos a interfaces concretos". Seminario de Investigación en Tecnologías de la Información Aplicadas a la Educación (SITIAE).

(Díaz, Paredes, 2010) "Generación automática de interfaces de usuario para aplicaciones colaborativas". Actas de Jornadas de Innovación y TIC educativas 2010. Pg 67-70.

(García, G. 2010) García Pérez, G. CIAT-GUI: Generación Automática de Interfaces Gráficas de Usuario en el Contexto de CIAM. Universidad de Castilla-La Mancha. 2010

(Giraldo et al. 2009) Giraldo, W., Molina A. I., Collazos C.A., Ortega M. and Redondo M. A., "CIAT, A Model-Based Tool for Designing Groupware User Interfaces Using CIAM". Computer-Aided Design of User Interfaces VI 2009, 201-212,

(Gooma et al, 2005) Gooma, M.,Salah, S. y Rahman, S."Towards A Better Model Based User Interface Development Environment: A Comprehensive Survey" Proceedings of the 38th Annual Midwest (2005).

(Limbourg, 2004) Limbourg, Q., *Multi-Path Development of User Interfaces. Université catholique de Louvain.* 2004

(López Jaquero, 2005) López Jaquero, "Adaptive User Interfaces Based on Models and Software Agents" Universidad de Castilla-La Mancha, 2005.

(Luyten, 2004) Luyten, K., "Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development". Universiteit Limburg. 2004

(MDA) http://www.omg.org/mda/.

(MDE) http://en.wikipedia.org/wiki/Model-driven_engineering.

(Molina et al. 2008) Molina, A. I., M. A. Redondo, M. Ortega and U. Hope, "CIAM: A methodology for the development of groupware user interfaces," Journal of Universal Computer Science(JUCS), 14, issue. 9, (2008c), 0948-695x.

(Molina et al. 2009) Molina, A. I., M. A. Redondo and M. Ortega, "A methodological approach for user interface development of collaborative applications: A case study," Sci. Comput. Program., 74, issue. 9, 754-776, (2009), 0167-6423

(Paredes et al, 2010) Paredes, M., Molina, A. I., Redondo, M. A., González, J.J., Díaz, L., Ezzaid, K. "CIAM Extendido con Generación Automática de IU Frente a Metodologías no Guiadas: Evaluación de una Experiencia con COFARCIR (L)". XI Congreso Internacional de Interacción Persona-Ordenador 2010.

(Paternò et al. 1997) Paternò, F., Mancini, C., Meniconi, S. "ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models". Interantional Conference on Human-Computer Interaction. 1997

(Paternò, 2000) Paternò, F. "Model-Based Design and Evaluation of Interactive Applications". Springer. 2000

(Paternò, 2003) Paternò, F., y Santoro, C., "A Unified Method for Designing Interactive Systems Adaptable to Mobile and Stationary Platforms, Interacting with Computers", Elsevier, 15, 2003, pp. 349-366.

(Paternò, 2011) Paternò, F., Santoro, C, Spano, L. D. "Engineering the authoring of usable service front ends". The Journal of Systems and Software 84 (2011) 1806– 1822

(Rigole, 2005) Rigole, P., "A Component-Based Infrastructure for Pervasive User Interaction" SOFTWARE TECHNIQUES FOR EMBEDDED AND PERVASIVE SYSTEMS, 2005

(Schlungbaum, 1996) Schlungbaum E., *Model-based user interface software tools - current state of declarative models*, Technical Report GIT-GVU 96-30, Georgia Institute of Technology, 1996.

(Shneiderman, 05) Shneiderman, B., *Diseño de Interfaces de Usuario. Addison Wesley, 2005.*

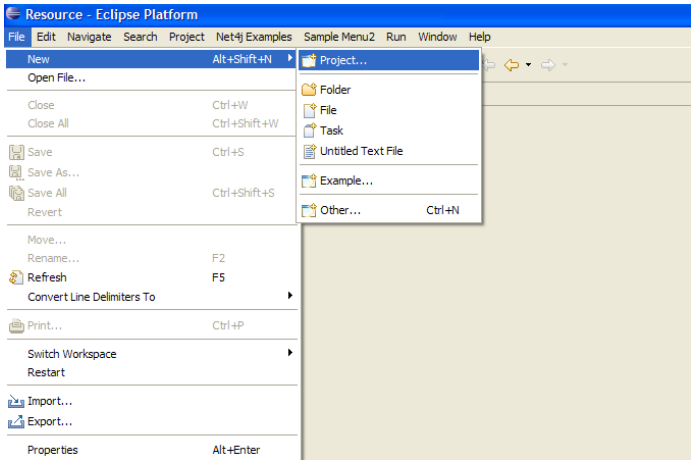
(Traetteberg , 2002)Traetteberg, H., Ph.D Thesis "Model-based User Interface Design", Noruega, Mayo 2002

Anexo

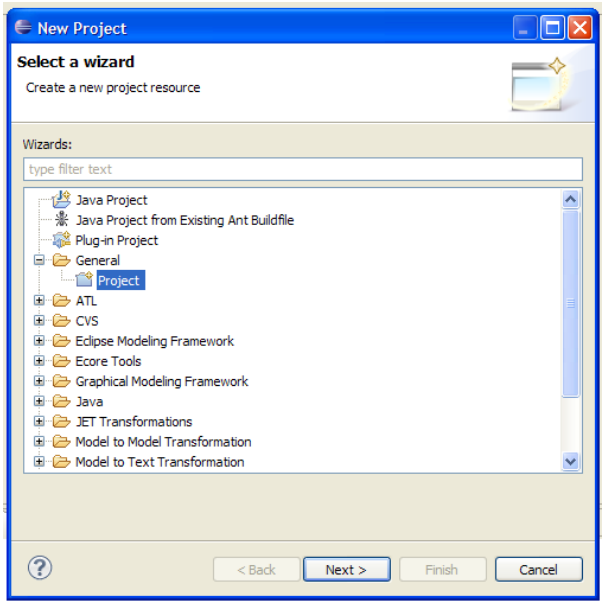
MANUAL DE USUARIO

Crear el proyecto

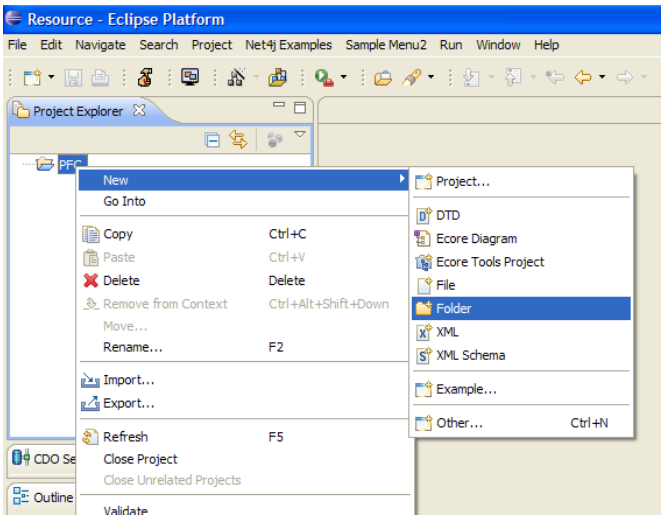
Seleccionamos File -> New -> Project



Dentro de la ventana seleccionamos General y lo llamados PFC.

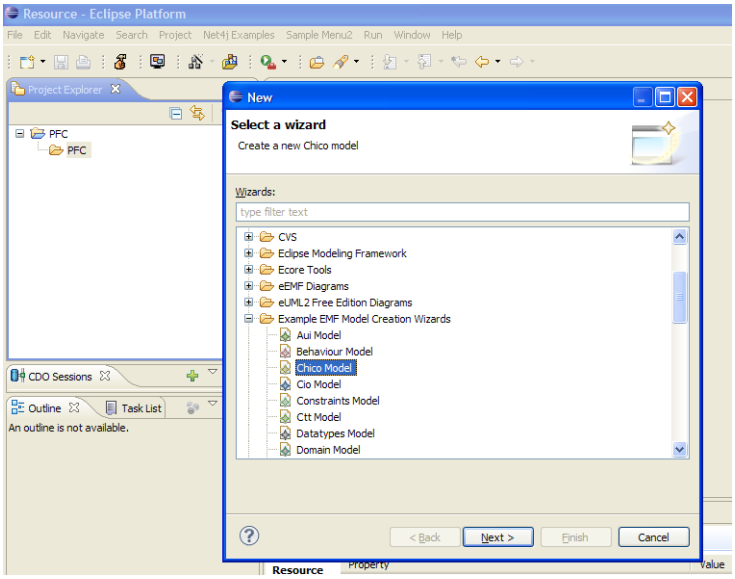


Una vez creado el proyecto, hay que crear una carpeta dentro de este, haciendo click derecho creamos una carpeta dentro del proyecto a la que también llamaremos PFC.

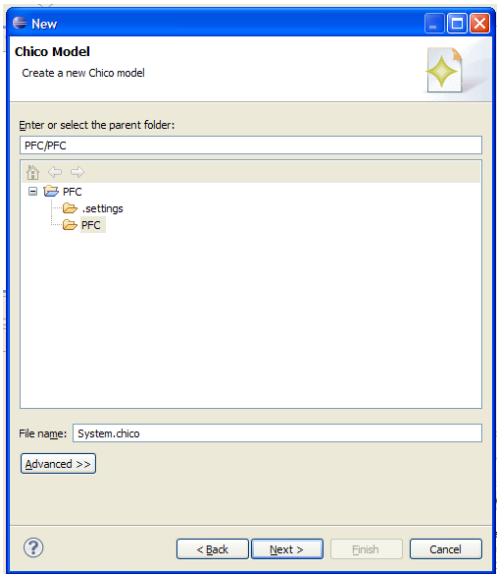


Crear los diagramas

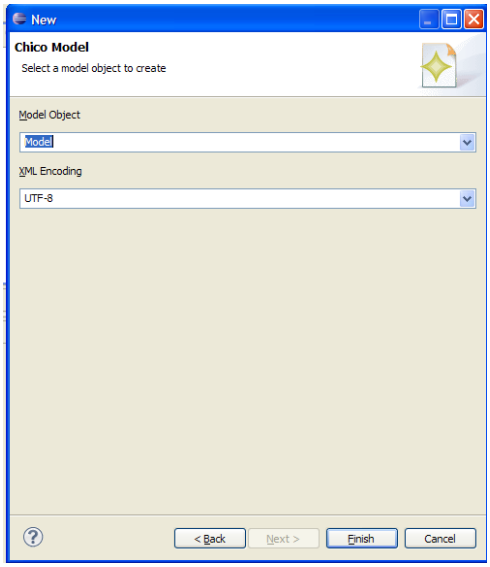
1. Se selecciona la carpeta PFC, una vez seleccionada presionar Ctrl+N y seleccionar chico Model en el asistente.



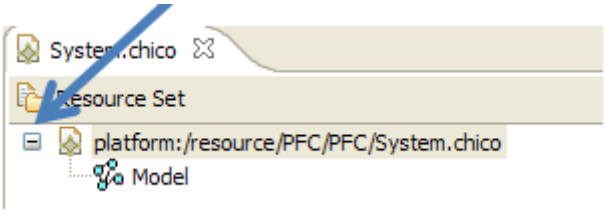
2. Nombrarlo System.chico



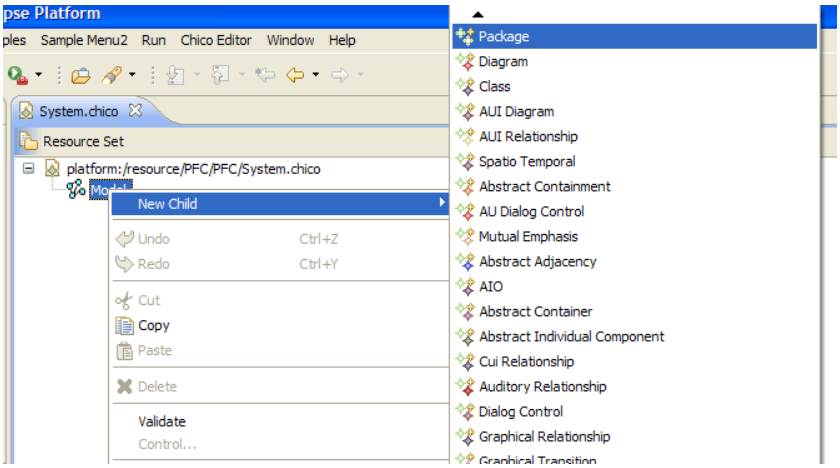
3. En Model Object seleccionar Model. Click en Finish



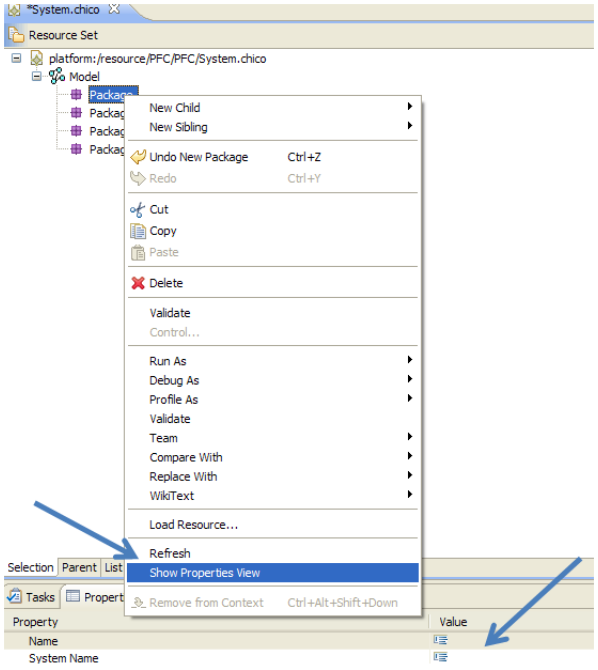
4. El archivo creado se abre automáticamente en el Workspace de Eclipse. Hay que hacer click en el icono de la izquierda. De esta forma se podrá ver la estructura del diagrama. En estos momentos sólo tenemos el Modelo principal.



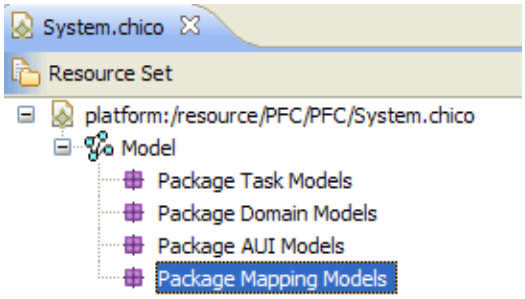
5. Seleccionamos el elemento Model y hacemos click con el botón derecho del ratón. De esta forma podremos añadir hijos al nodo principal. Hay que crear cuatro Packages.



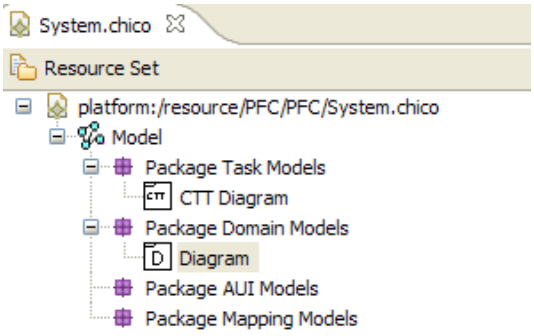
6. Los Packages son: Task Models, Domain Models, AUI Models y Mapping Models. Para nombrarlos hay que hacer click con el botón derecho sobre el paquete y seleccionar, Show Properties View, de esta forma se abrirá la pestaña en la parte inferior de Eclipse. Hay que rellenar el campo Name.



7. La jerarquía resultante sería:



8. Hacer lo mismo que en el paso 5, para crear un CTTDiagram hijo del paquete Task Models y un Diagram como hijo del paquete Domain Models. La jerarquía resultante debe ser como la siguiente:

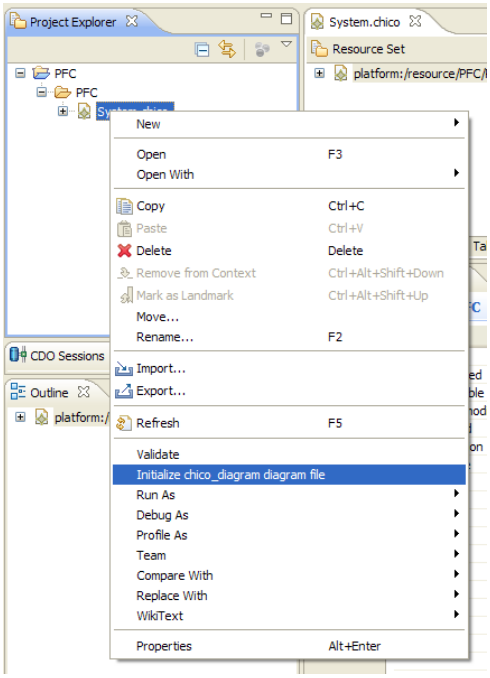


Creación del diagrama CTT

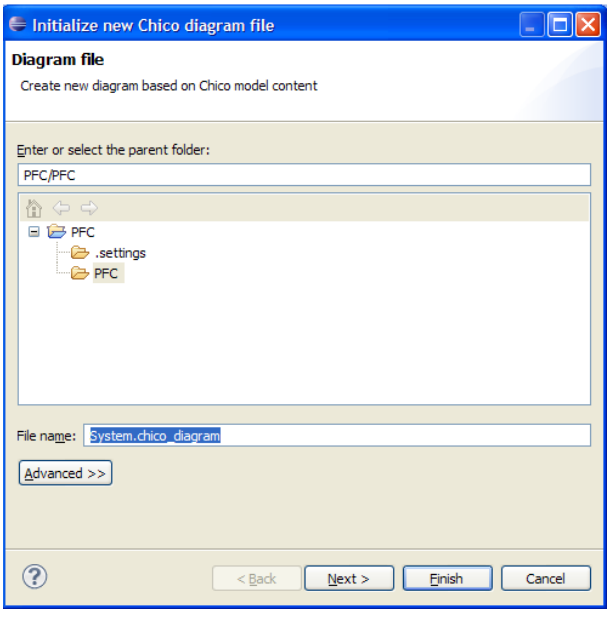
1. Pulsar Ctrl+S para salvar el archivo

2. Cerrar System.chico

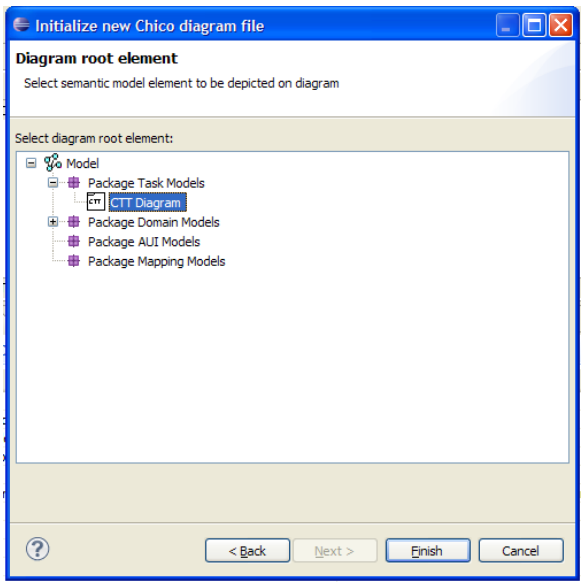
3. Sobre la representación del archivo System.chico en el Package Explorer, hacer click con el botón derecho y seleccionar *initialize chico_diagram file.*



4. Editar el nombre del archivo y pulsar Next. El nombre que sale por defecto es válido.

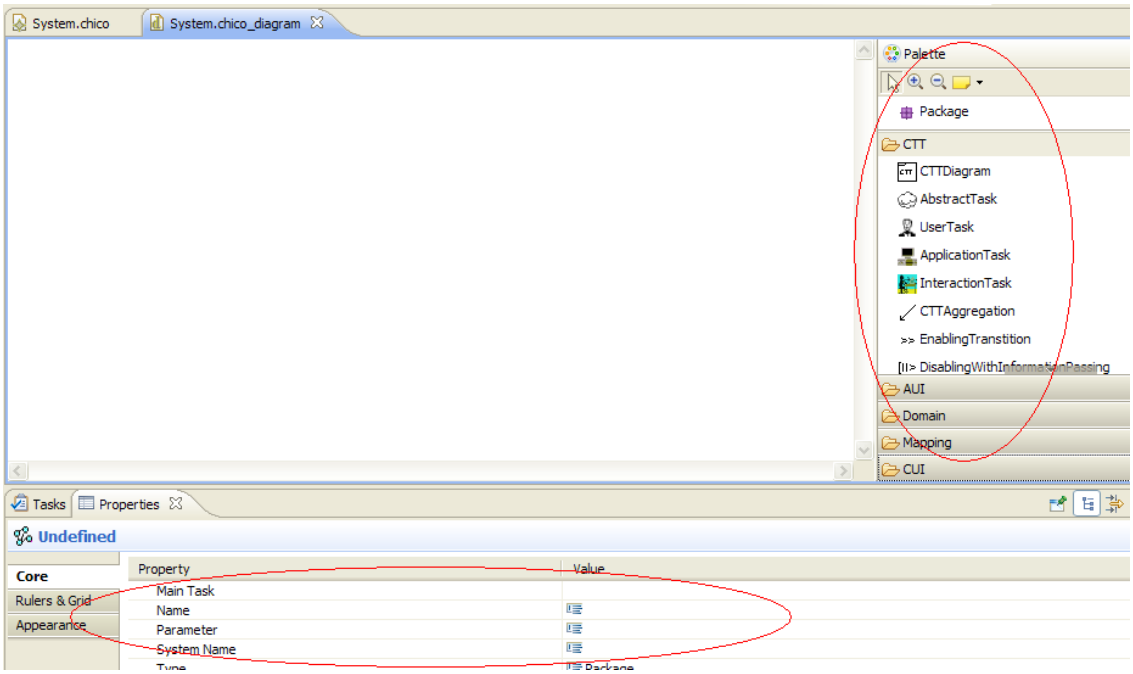


5. Seleccionar CTTDiagram y Finish.



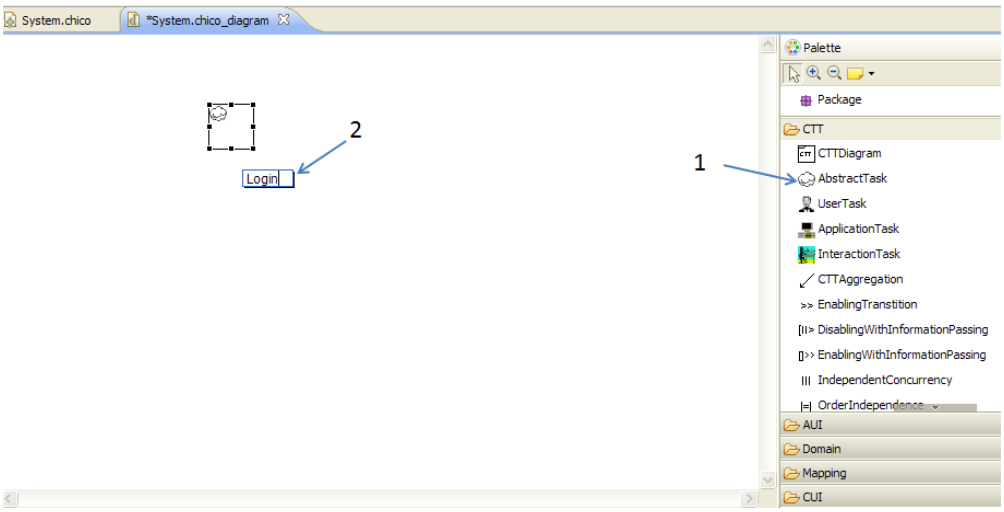
6. Se ha creado un nuevo archivo que Eclipse abre automáticamente.

7. El editor tiene el siguiente aspecto:

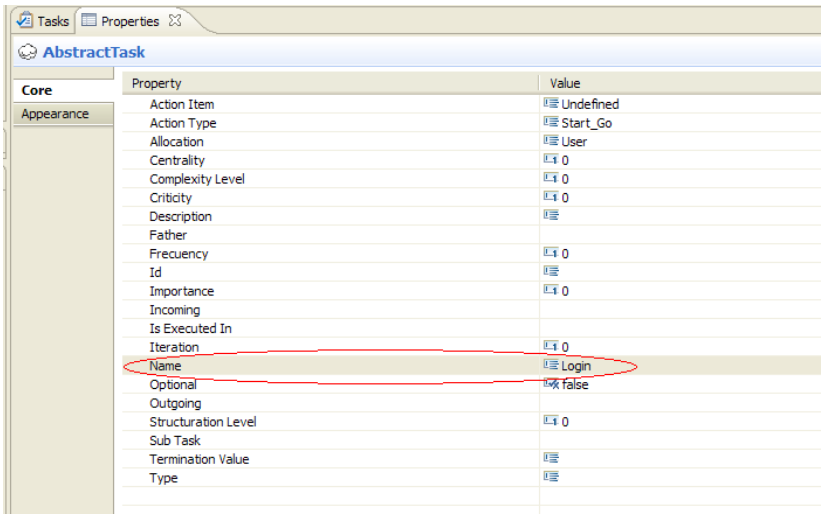


8. Sobre el área de edición se pueden poner diferentes elementos del área Palette, en el área de Properties se pueden cambiar las propiedades de los elementos.

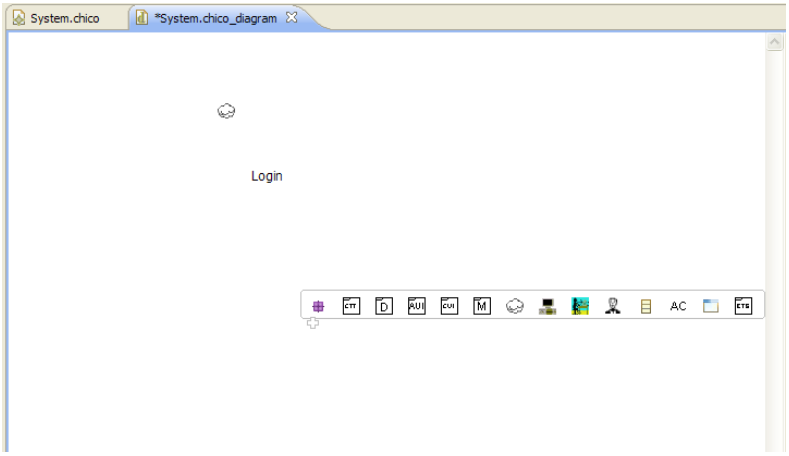
9. Para poner un elemento en el área de edición hay que seleccionar el elemento que se quiere editar (de la paleta de elementos de la derecha) y soltarlo sobre el área de edición.



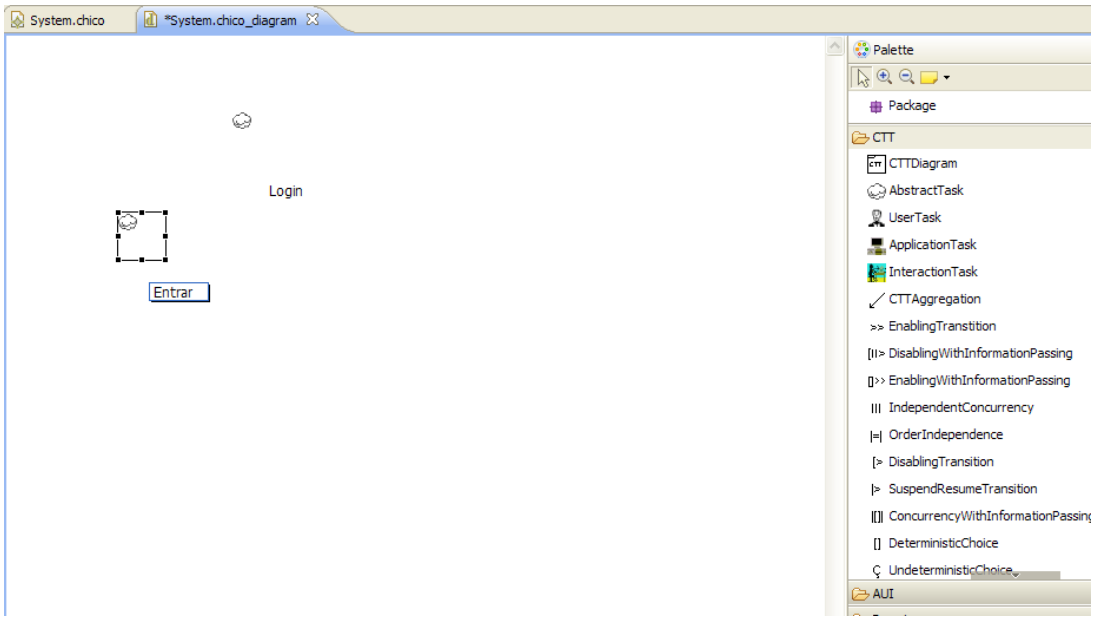
10. El editor te permite nombrar la tarea. Si se quiere cambiar más adelante se puede hacer en el área de Properties.



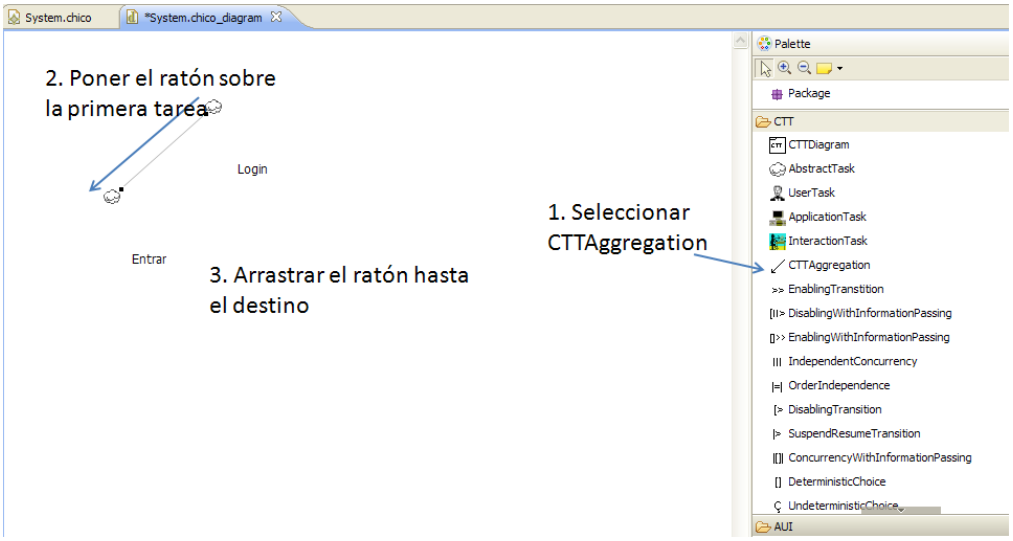
11. Otra forma de poner elementos en el área de edición es haciendo click en un área vacía y el editor abrirá un pop-up con un menú abreviado.



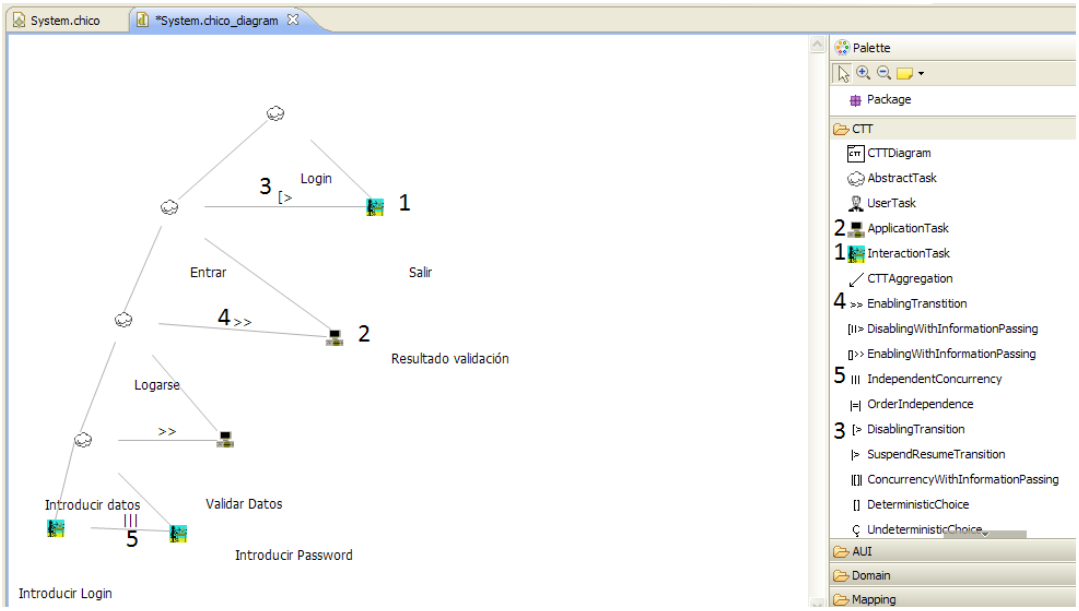
12. Seguidamente, poner una nueva tarea.



13. Queremos establecer una relación padre-hijo (o CTTAggregation) entre estas dos tareas. Proceder como se muestra:

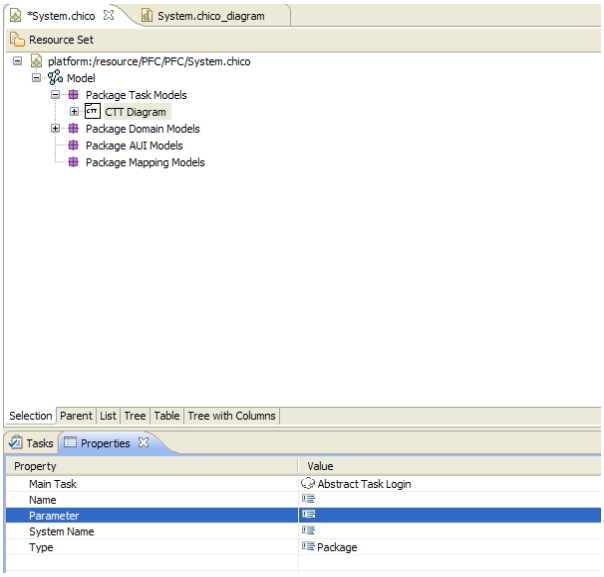


14. Hay que construir esta estructura, los números representan qué elementos de la Palette corresponden con la representación final de los elementos del modelo. Todas las relaciones de una tarea padre a una tarea hija son del tipo CTTAggregation.

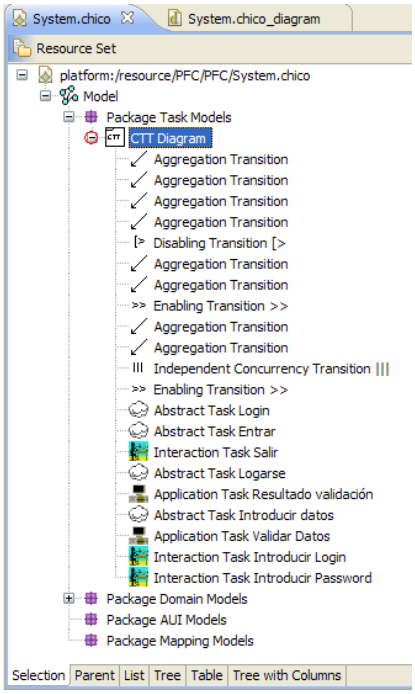


15. Salvar el documento presionando Ctrl+S.

16. Abrir System.chico de nuevo y rellenar el campo Name y el campo Main Task (Tarea Principal) del CTTDiagram. El campo Main Task también se puede rellenar desde la vista del CTTDiagram.

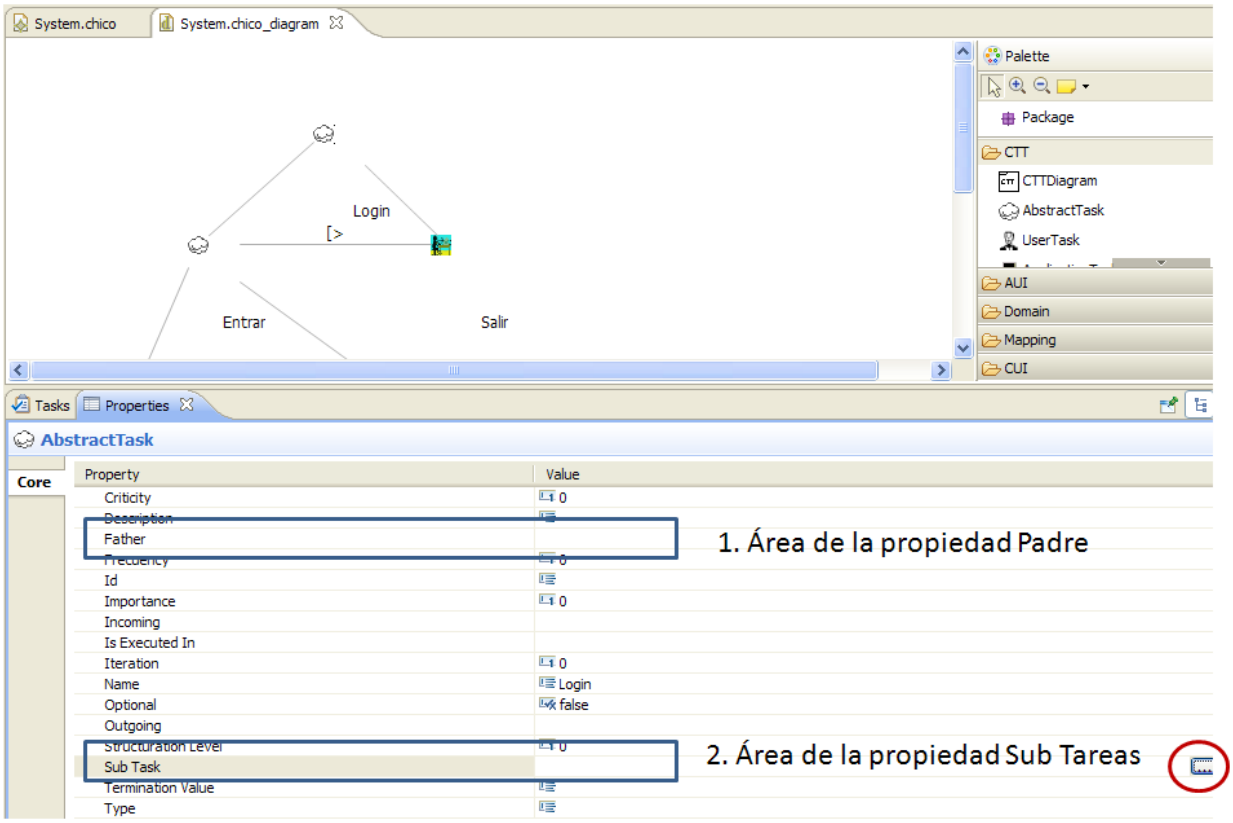


17. Tenemos que configurar cuáles son las tareas hijas de otras tareas y cuáles son los padres de las tareas. Hay que configurarlo desde el área de propiedades. Se puede configurar desde la jerarquía del CTTDiagram:

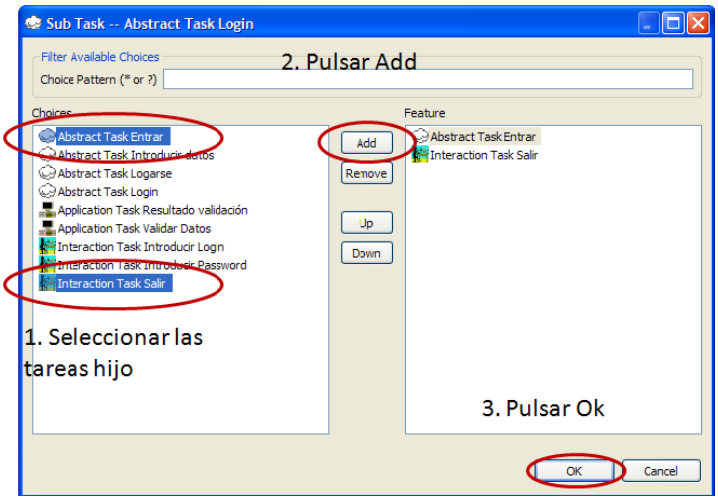


O desde el diagrama del árbol de las etapas anteriores. El área de propiedades de las tareas en ambos casos es el mismo.

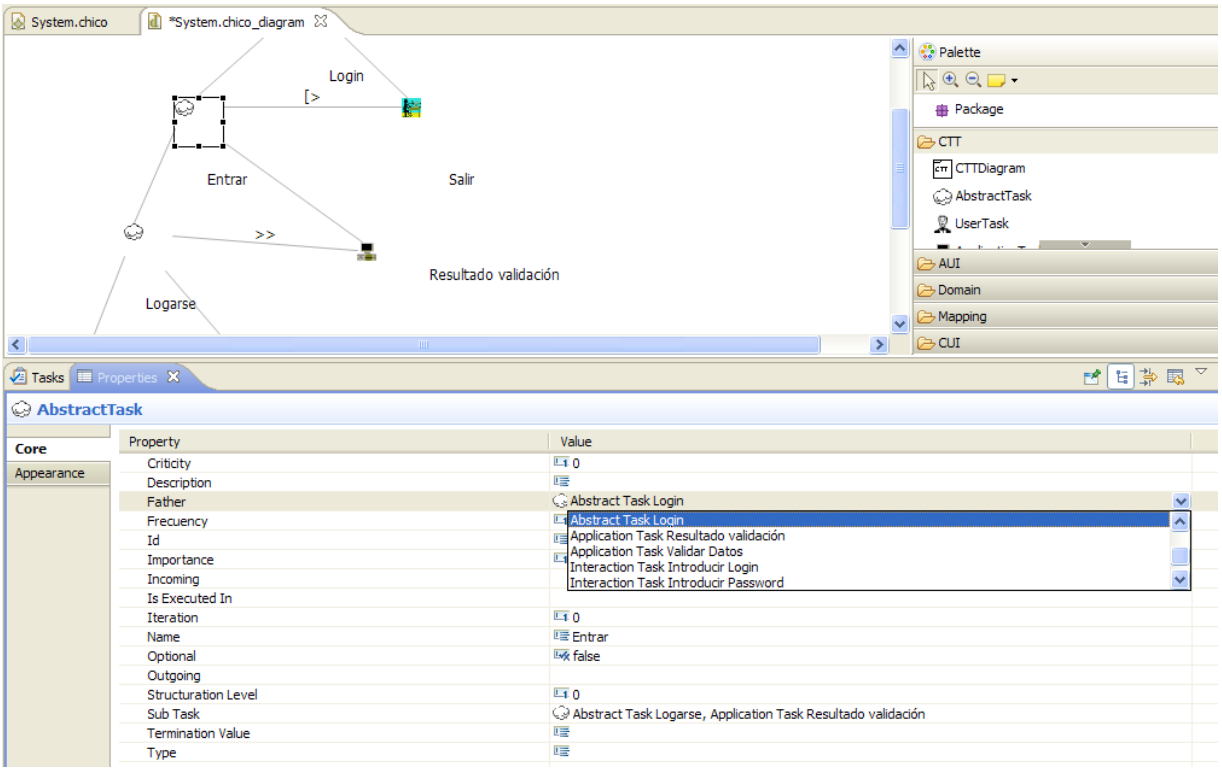
18. En nuestro ejemplo tenemos que la tarea Abstracta Login tiene 2 hijos: Entrar y Salir. Entrar tiene a su vez 2 hijos: Logarse y Resultado de la validación, etc. Si nos fijamos en el área de propiedades, hay dos campos donde se pueden configurar estos valores: Father y Sub Task



19. En el caso de la tarea principal, sólo hay que configurar la propiedad de Sub Tareas, para configurarlas se abre el menú de Sub Task (círculo rojo en la anterior figura).



20. Configurar también el padre de cada tarea.



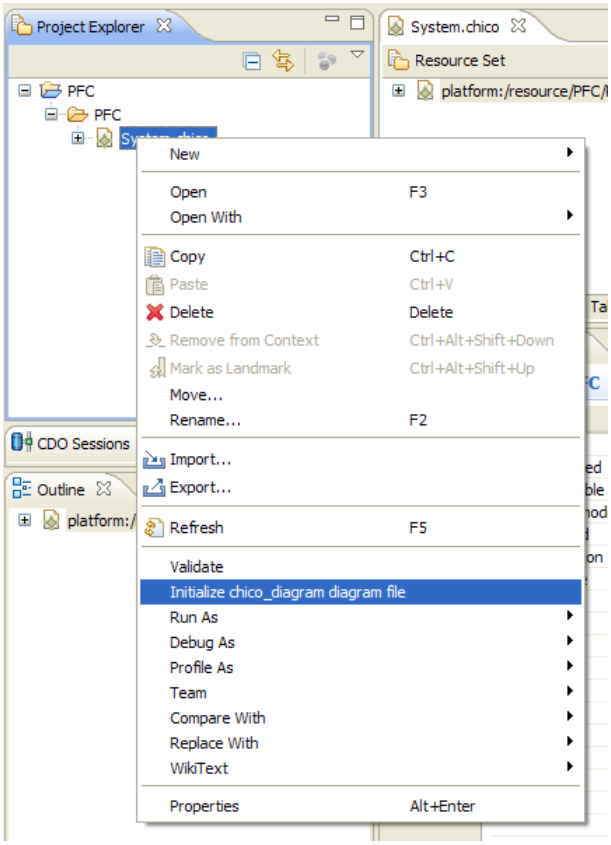
21. Repetir los puntos 19 y 20, por cada tarea que forma parte del diagrama.

22. Salvar el archivo. Ctrl + S.

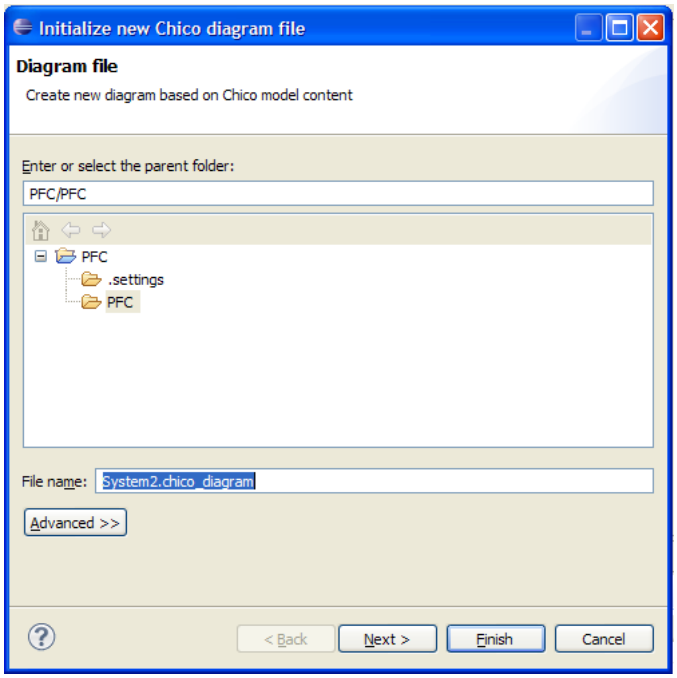
Creación del Diagrama de Dominio

1. Cerrar System.chico

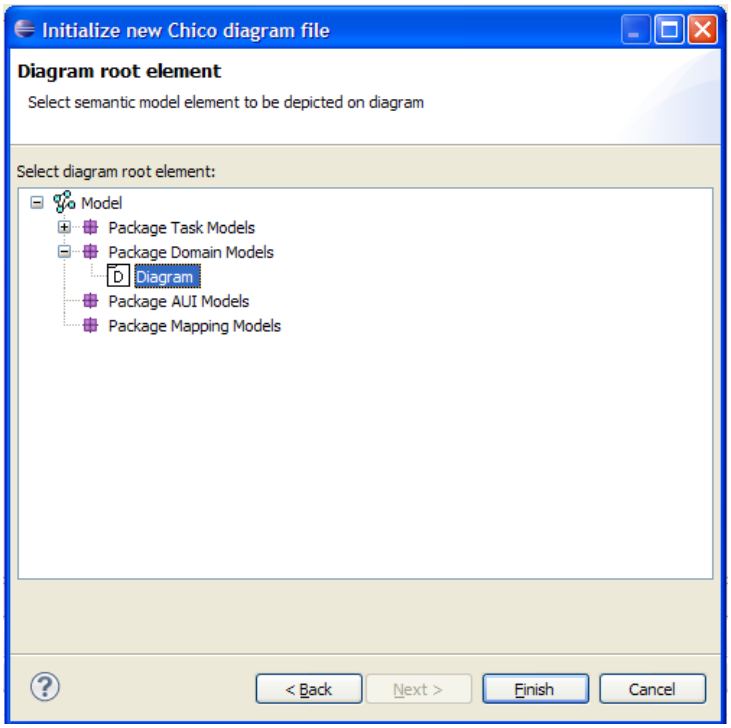
2. Sobre la representación del archivo System.chico en el Package Explorer, hacer click con el botón derecho y seleccionar *initialize chico_diagram file.*



3. Editar el nombre del archivo y pulsar Next. El nombre que sale por defecto es válido.



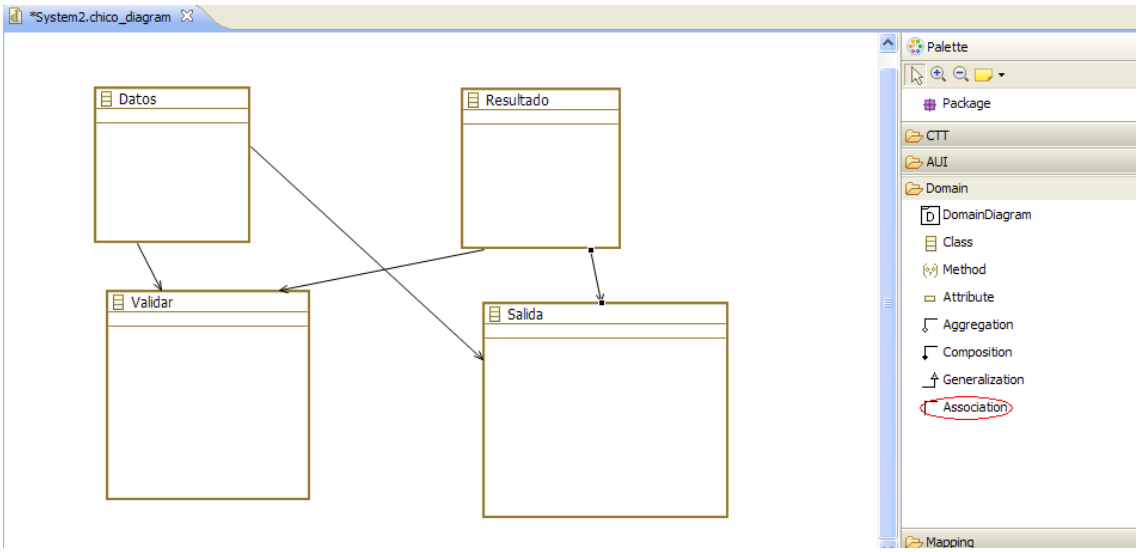
4. Seleccionar Diagram y Finish.

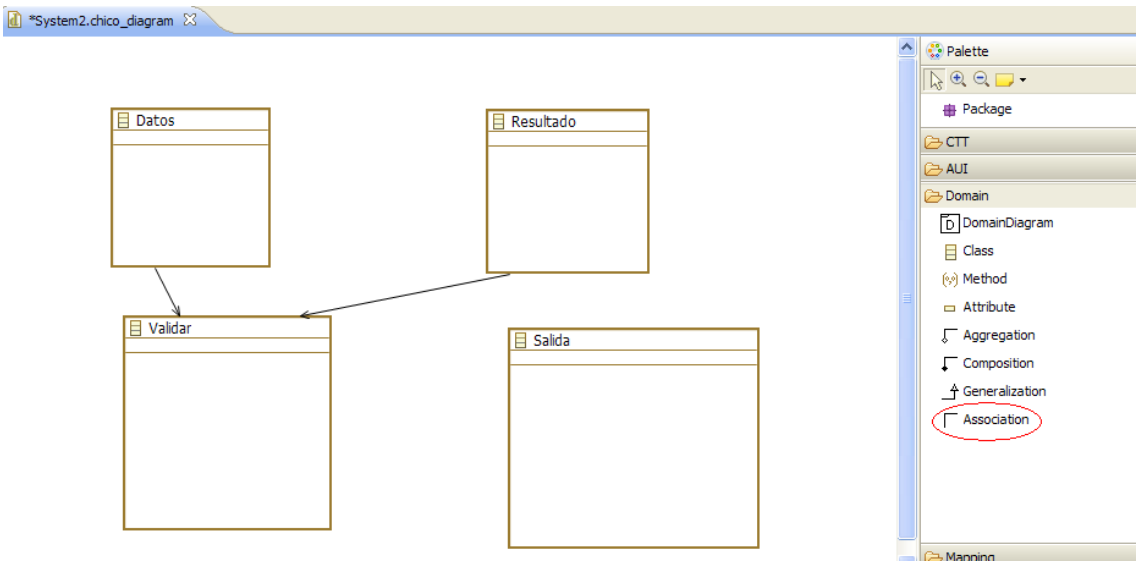


5. Se ha creado un nuevo archivo que Eclipse abre automáticamente.

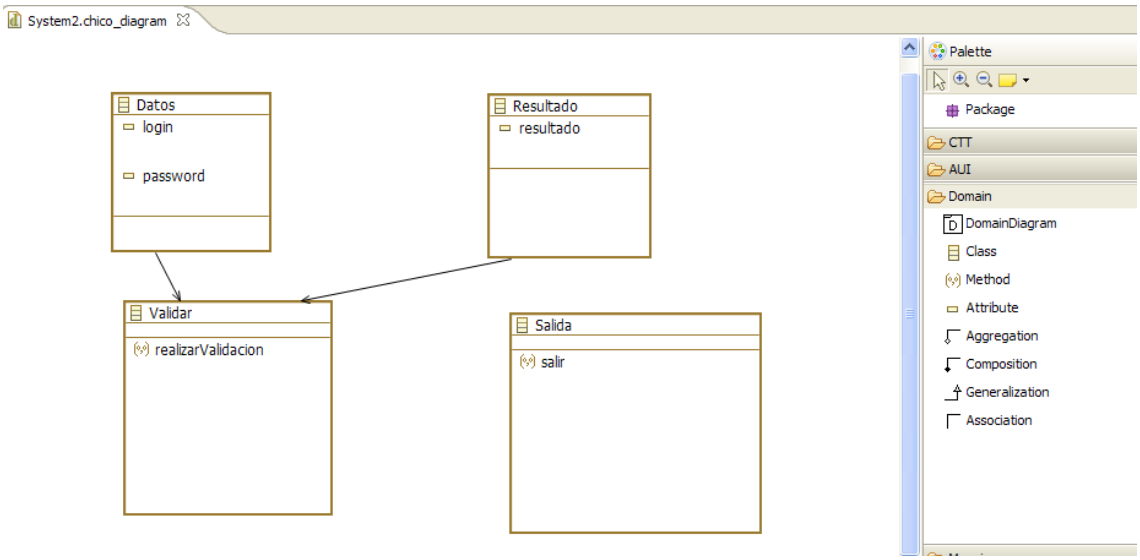
6. Comenzamos a crear el Diagrama de Dominio (Domain Diagram), todas las características que necesitaremos representar estarán dentro de la sección Domain del área Palette. Comenzamos con una clase llamada Datos.

7. Creamos el resto de clases necesarias para el ejemplo y añadimos la relación entre las clases. Para ello hay que poner cuatro relaciones Association tal y como se observa:





8. Añadir los atributos y los métodos a las clases creadas. Los atributos se insertar en el primer contenedor de las clases y los atributos en la segunda.



9. El siguiente paso es configurar los parámetros de los atributos:Login, password y resultado:

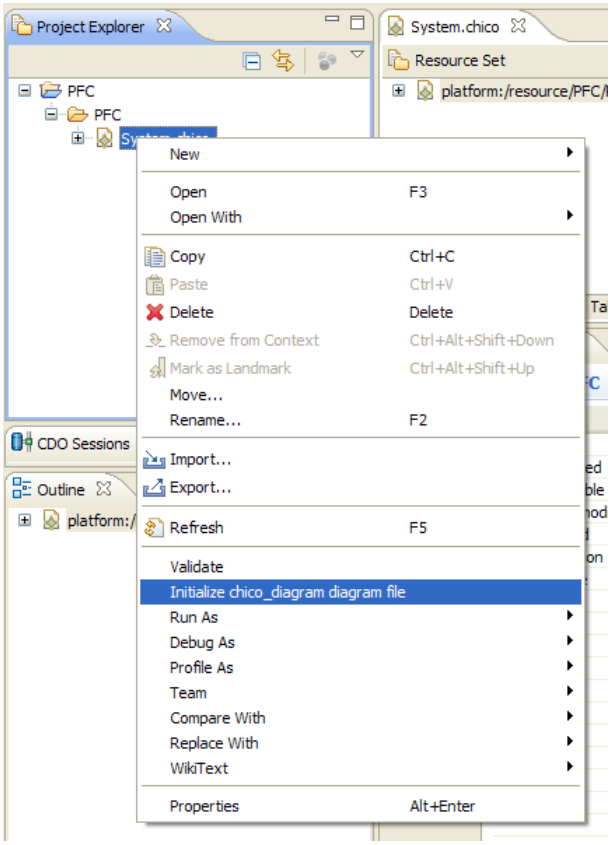
Property	Value
Attribute Card Max	1
Attribute Card Min	1
Attribute Data Type	String

10. Salvar Ctrl+S.

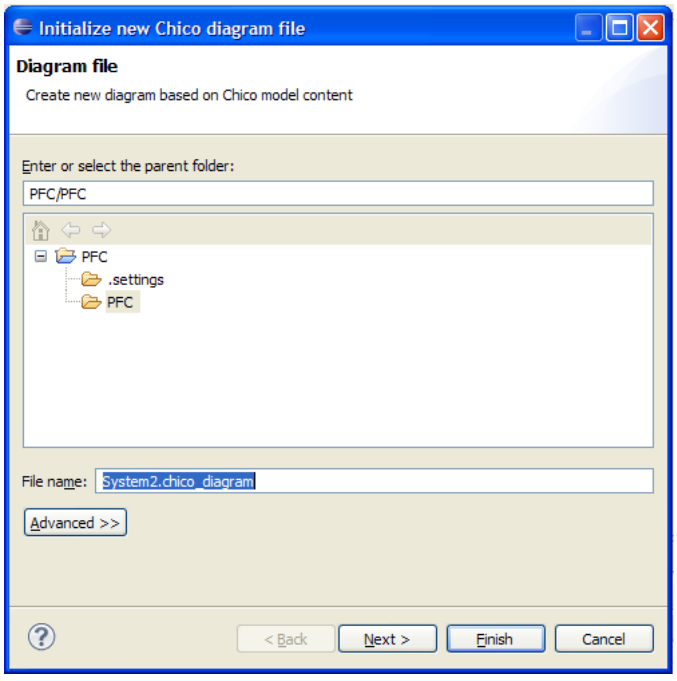
Creación del Diagrama de Mapping

1. Cerrar System.chico

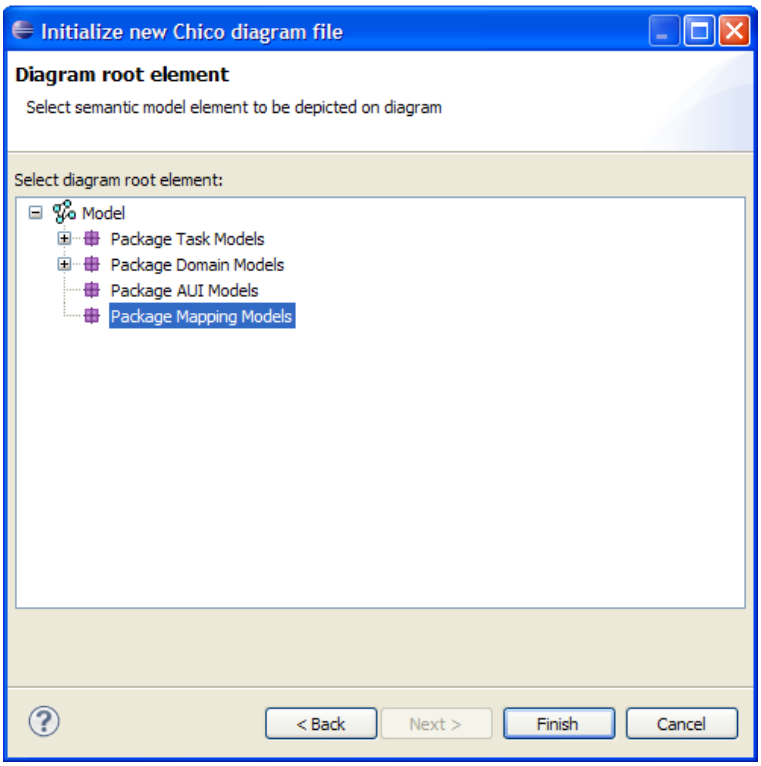
2. Sobre la representación del archivo System.chico en el Package Explorer, hacer click con el botón derecho y seleccionar *initialize chico_diagram file.*



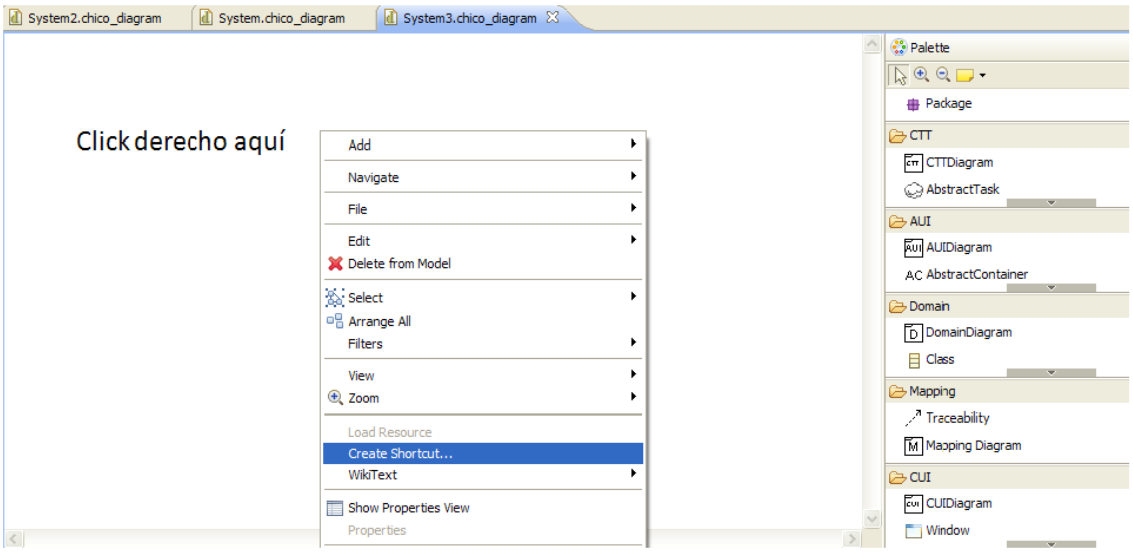
3. Editar el nombre del archivo y pulsar Next. El nombre que sale por defecto es válido.



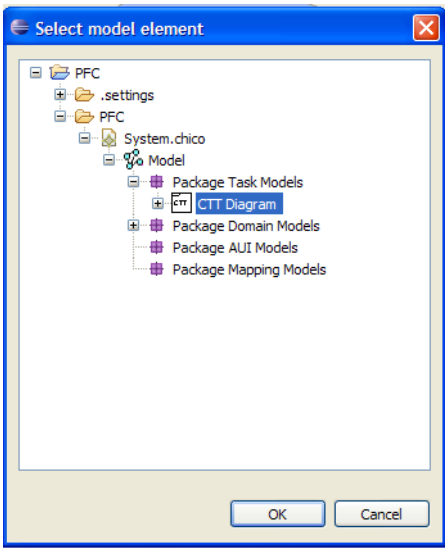
4. Seleccionar Mapping Models y Finish.



5. El archivo se abre por defecto. Hacer click derecho en la página en blanco y seleccionar *Create Shortcut*.

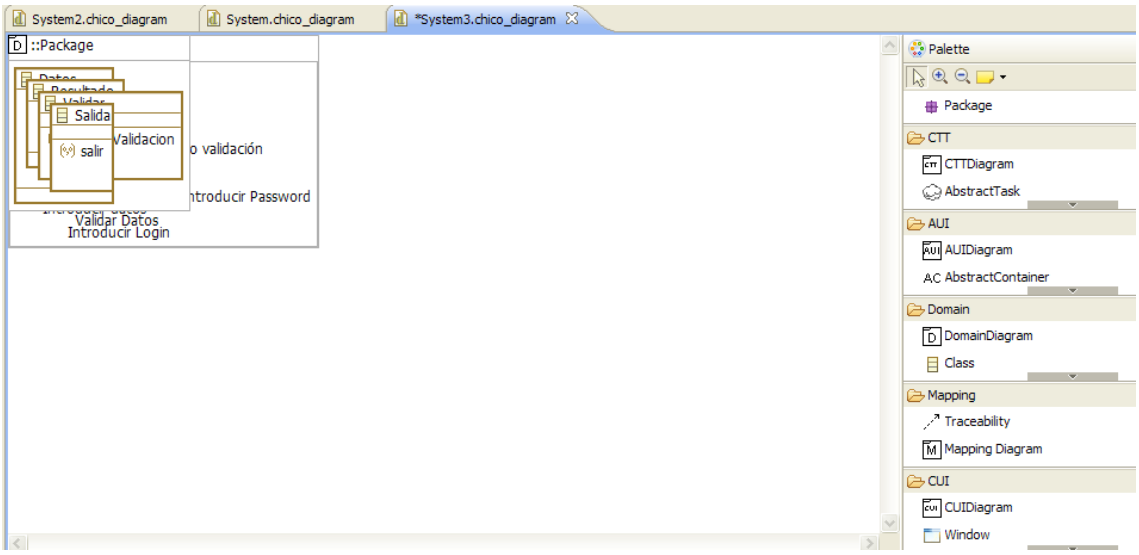


6. Seleccionar el CTTDiagram y pulsar Ok.

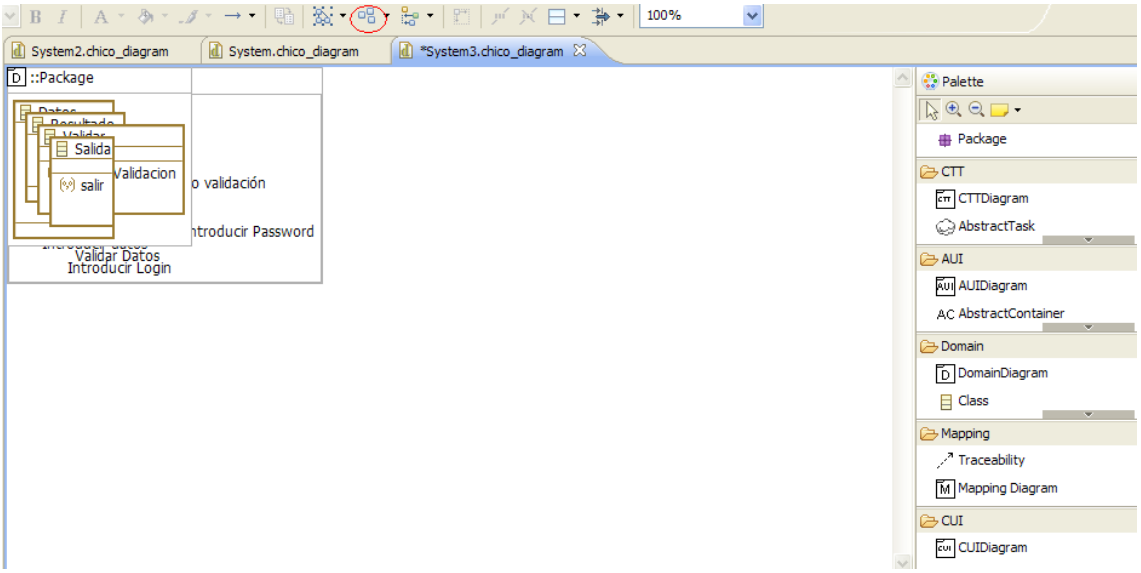


7. Realizar los pasos 5 y 6 pero seleccionando el Diagram.

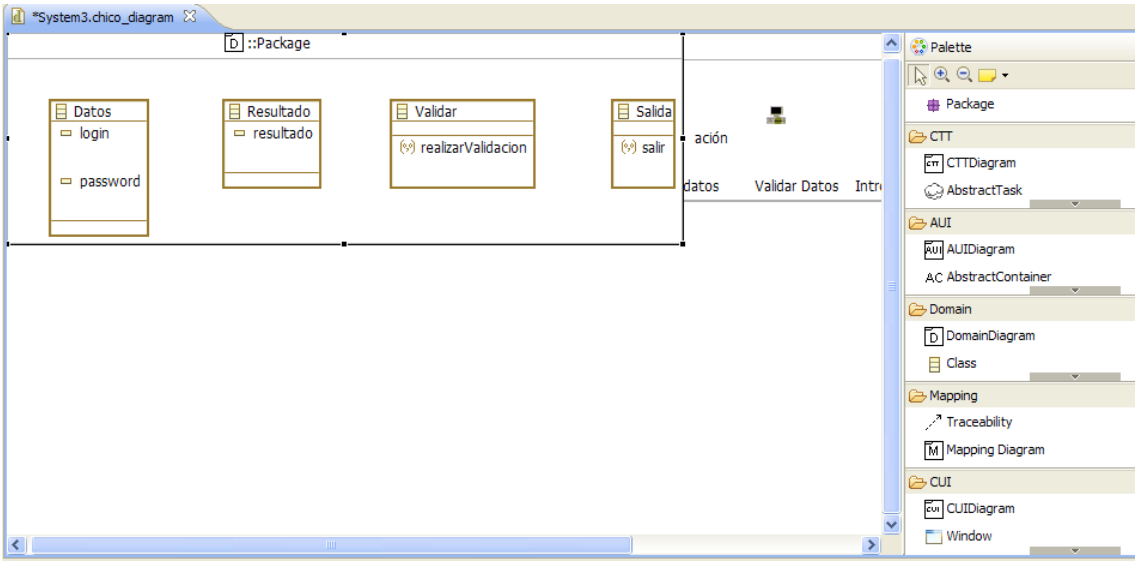
8. El resultado es el siguiente



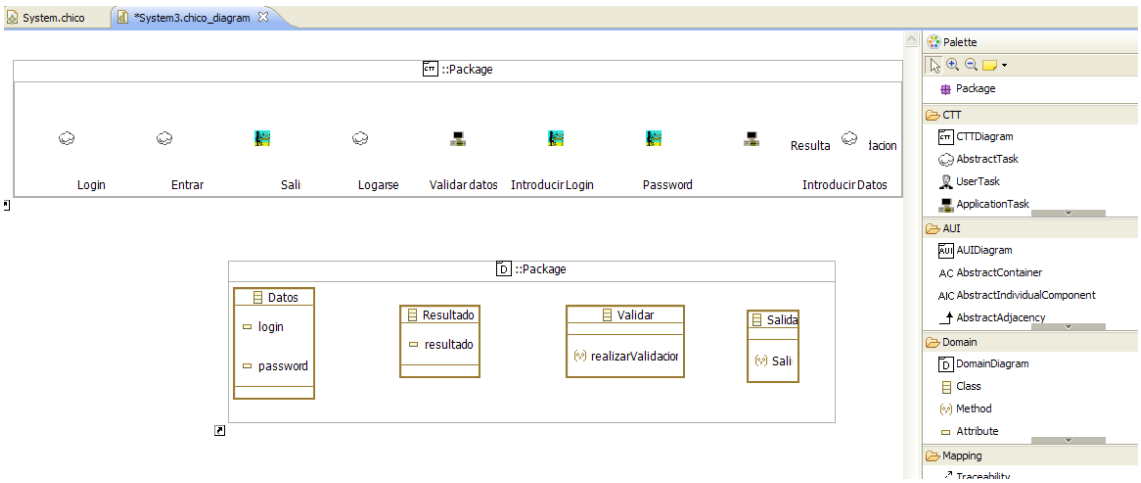
9. El botón que podemos observar dentro del círculo señalado permite reorganizar los componentes. Antes de utilizar este botón es necesario seleccionar todos los componentes.



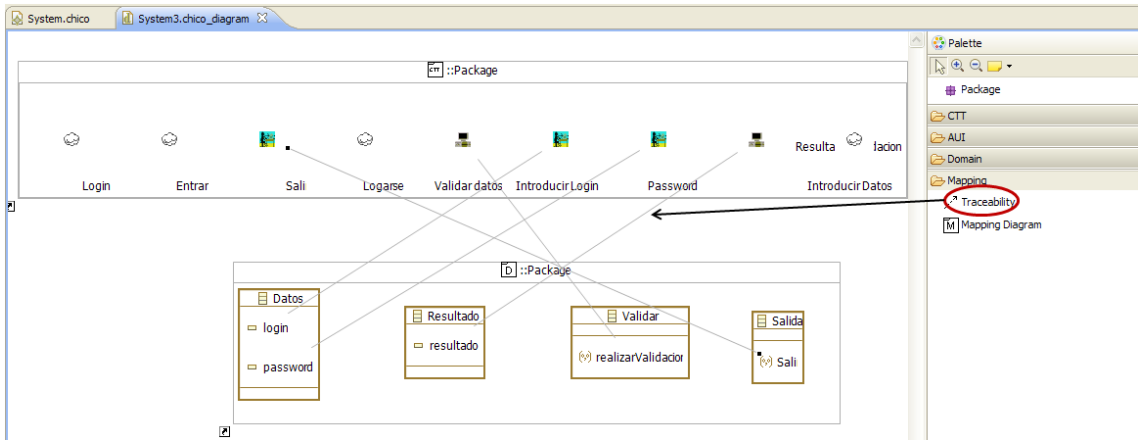
10. El resultado sería:



11. Se pueden recolocar los elementos si se quiere[7]



12. Añadir las relaciones de trazabilidad entre los elementos de los diagramas. La trazabilidad representa la correspondencia entre los elementos del diagrama CTT y los elementos del dominio descrito. Es importante que esta relación sea hecha desde los elementos tareas hacia los elementos de dominio. El resultado sería el siguiente:



13. A continuación hay que editar las propopiedades de las tareas denominadas Action Type y Action Item, para que la aplicación pueda utilizar la interfaz adecuada mediante la que tiene que representar los distintos elementos. Sobre el área de propiedades hay que configurar los parámetros de la siguiente forma:

[7] El grupo CHICO está trabajando en estos momentos para que se visualicen correctamente las relaciones entre las tareas.

13.1.:Introducir Login y Password:

Property	Value
Action Item	Element
Action Type	Select

13.2: Salir:

Property	Value
Action Item	Element
Action Type	Stop_Exit

13.3: Validar Datos:

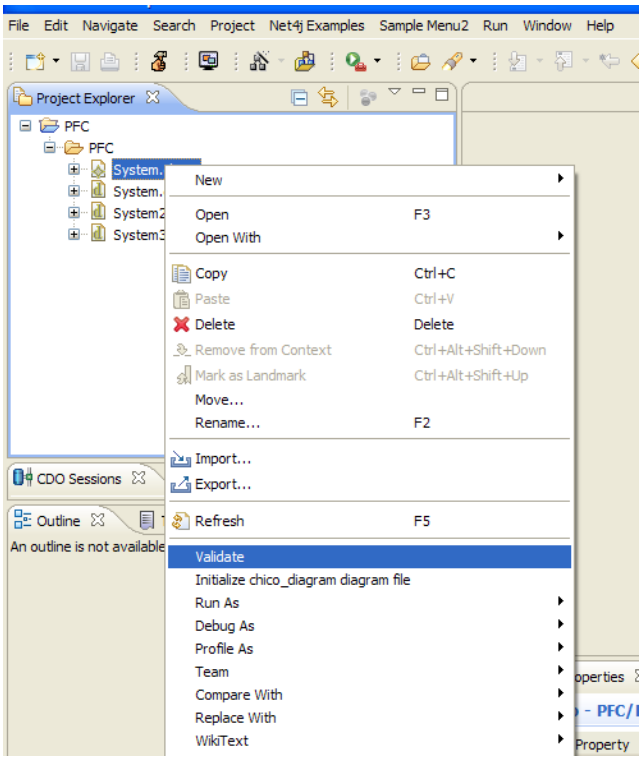
Property	Value
Action Item	Operation
Action Type	Start_Go

13.4: Resultado Validación:

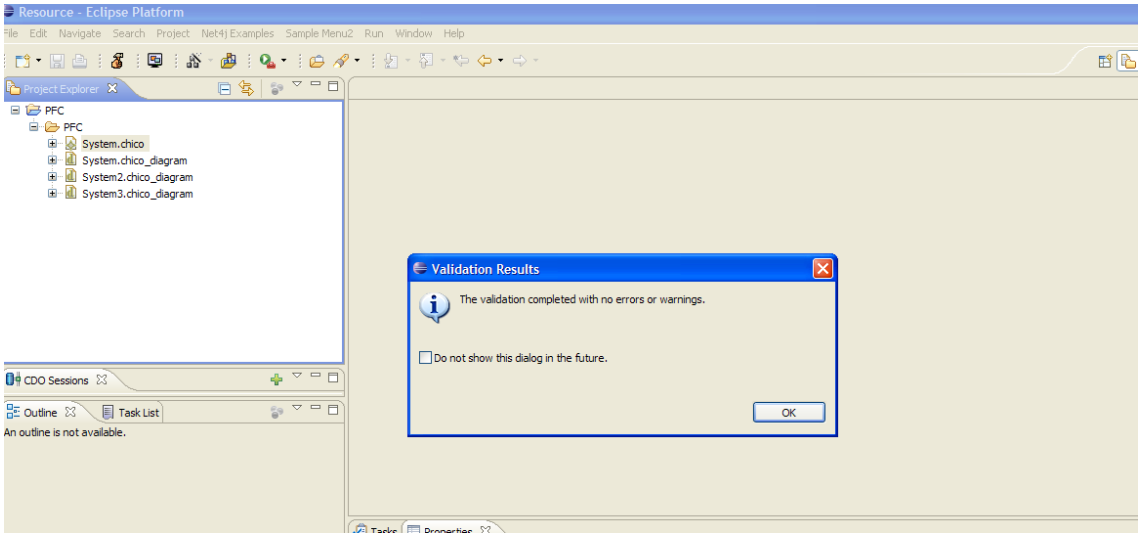
Property	Value
Action Item	Element
Action Type	View

14. Salvar el documentos Ctrl+S.

15. Llegados a este punto podemos comprobar si el modelo tiene errores. Se puede validar el archivo System.chico.



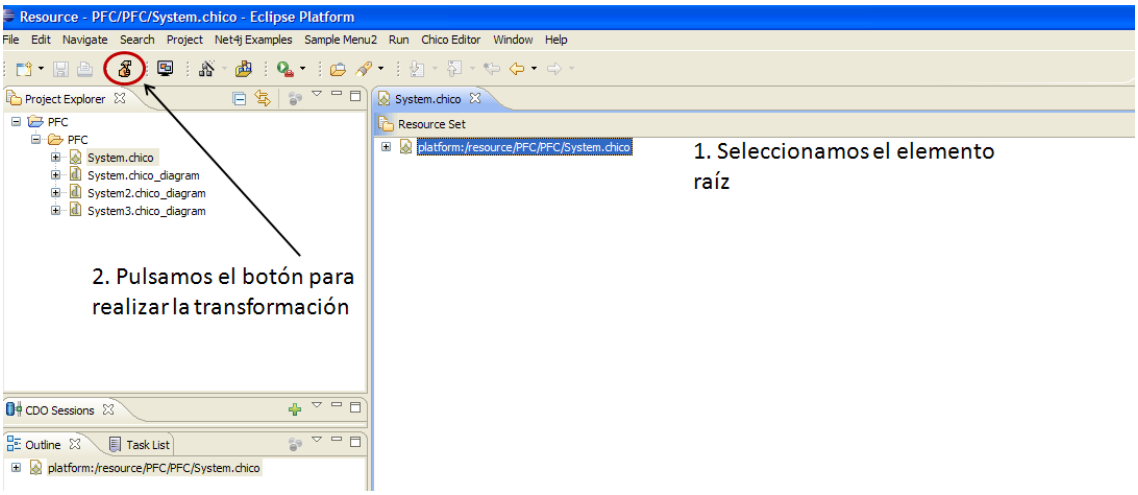
16. El resultado debe ser el siguiente.



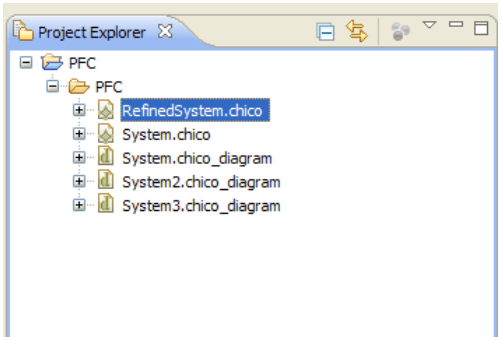
17. Este mensaje indica que el modelo está construido correctamente.

Obtención de la interfaces de usuario abstracta y concreta

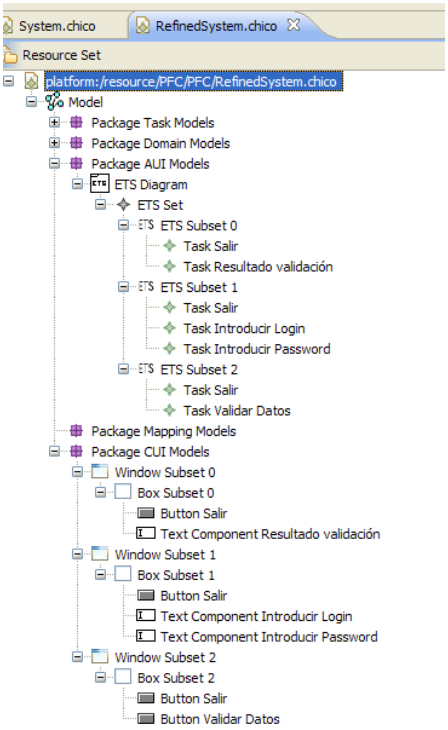
1. Una vez que hemos creado el modelo correctamente. El siguiente paso sería realizar las transformaciones para obtener las interfaces de usuario abstractas y concretas. Para ellos abrimos System.chico. Una vez abierto, seleccionamos el elemento raíz del modelo y pulsamos el botón para realizar la transformación:



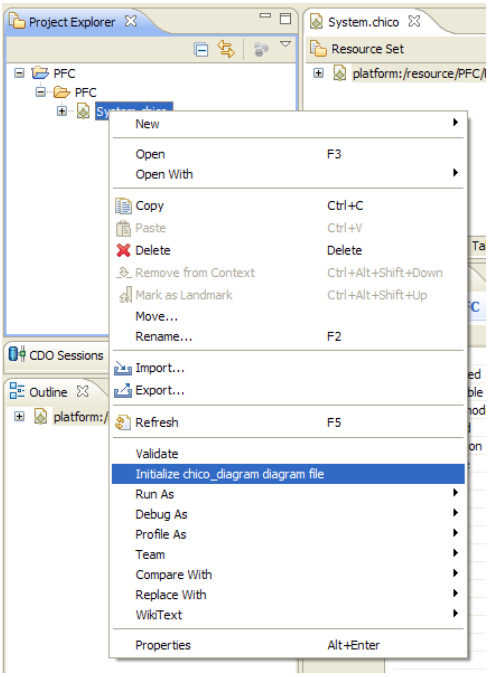
2. Cuando ejecutamos la transformación aparece un nuevo archivo llamado RefinedSystem.chico en el mismo nivel que está el archivo System.chico.



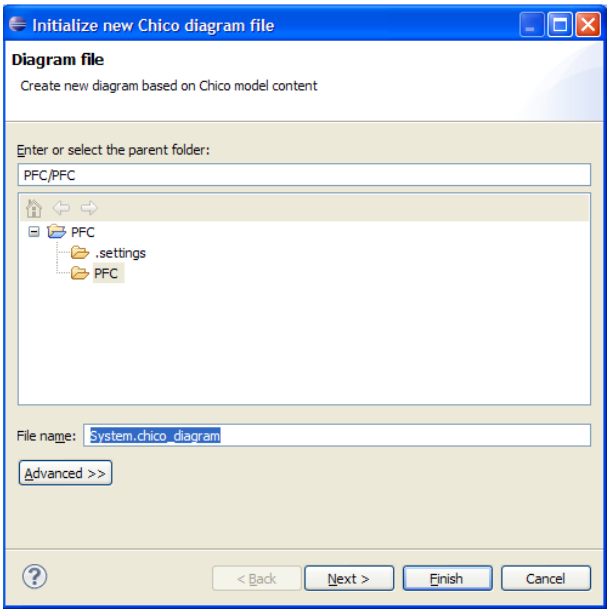
3. Abrimos el archivo resultante y si expandimos la estructura podemos observar que se han creado la interfaz de usuario abstracta y la interfaz de usurio concreta.



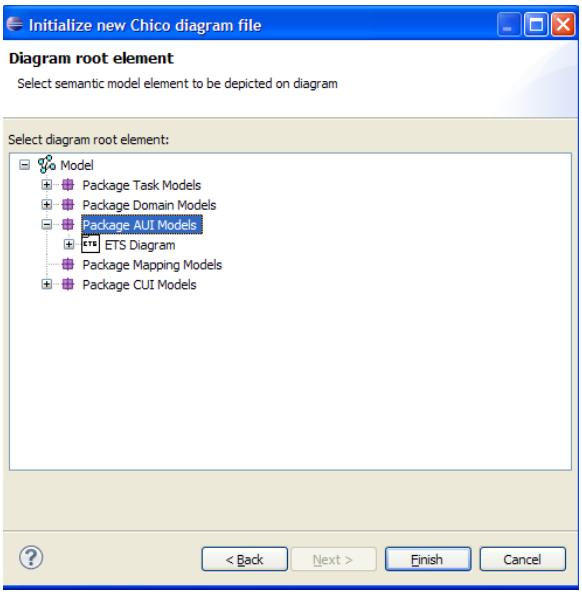
4. Para visualizar de forma gráfica la interfaz de usuario abstracta, sobre la representación del archivo RefinedSystem.chico en el Package Explorer, hacer click con el botón derecho y seleccionar *initialize chico_diagram file.*



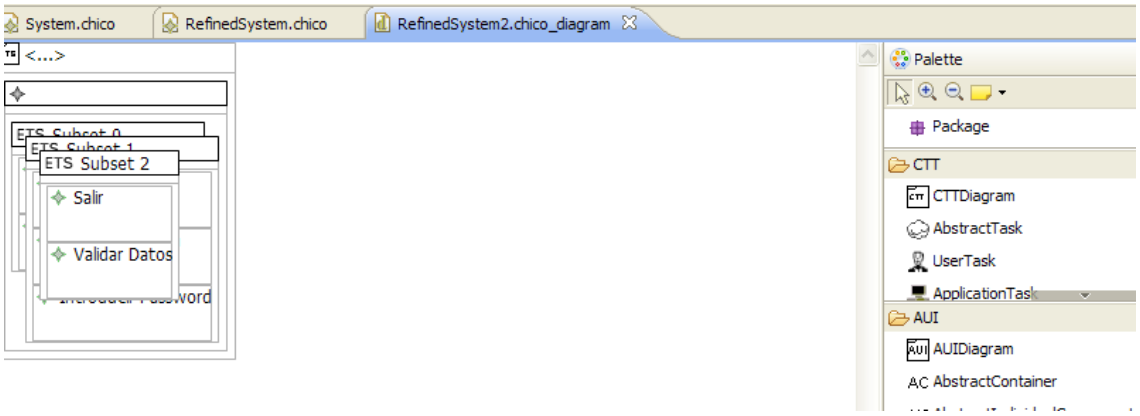
5. Editar el nombre del archivo y pulsar Next. El nombre que sale por defecto es válido.



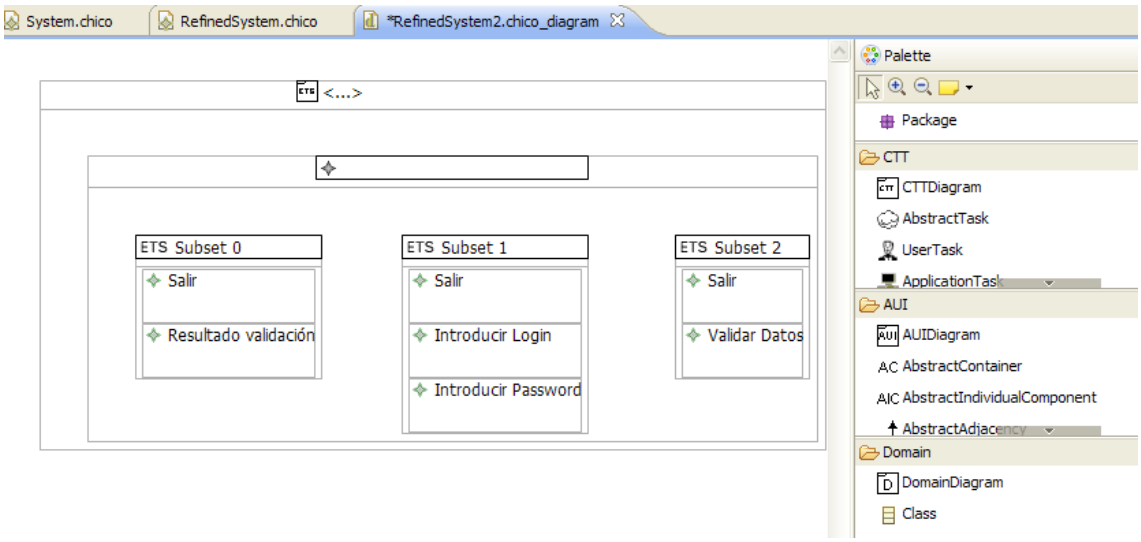
6. Seleccionar Package AUI Models y Finish.



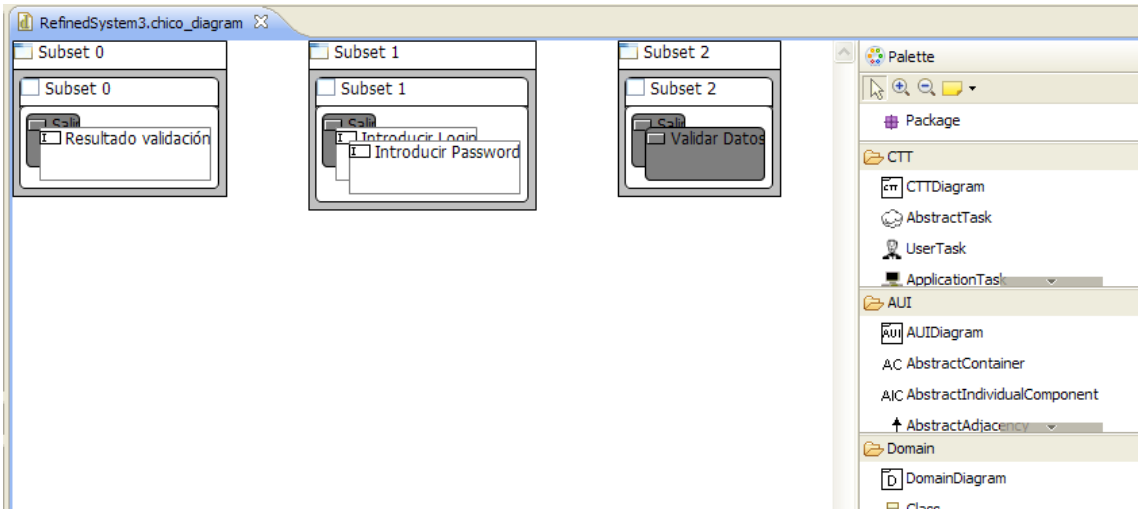
7. Obtenemos el siguiente resultado.



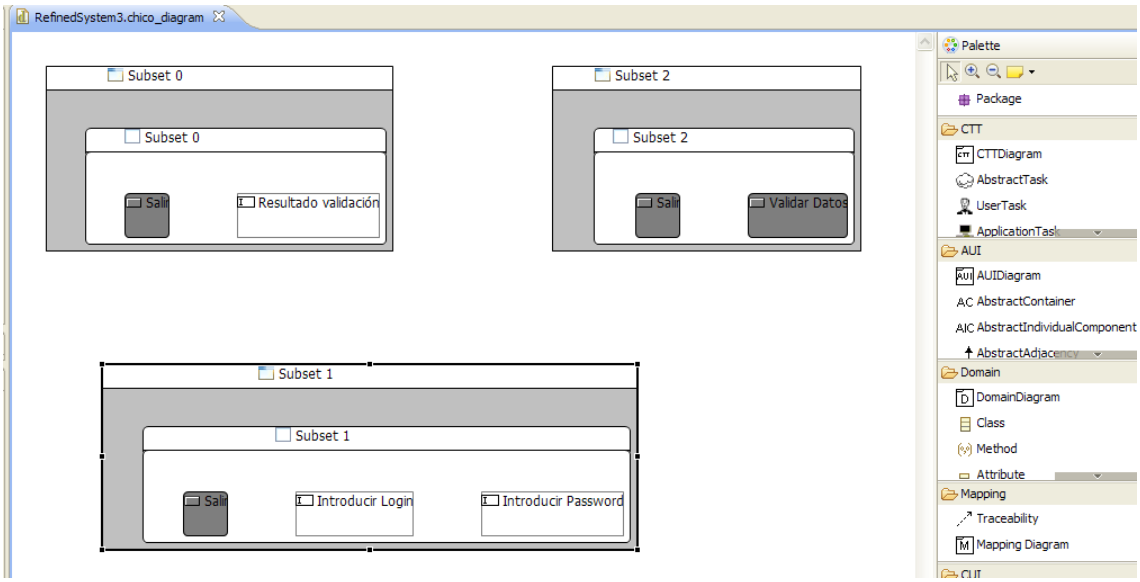
8. Si lo expandimos vemos la representación de la interfaz de usuario abstracta:



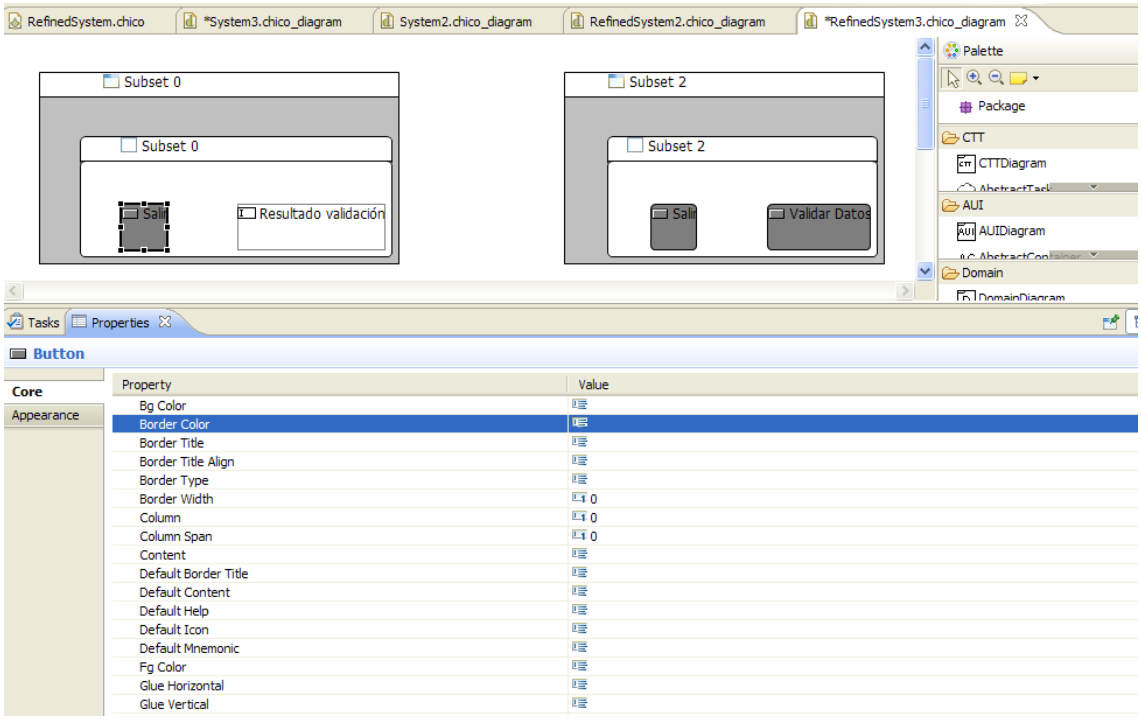
9. Para la visalizamos de forma gráfica de la interfaz de usuario concreta repetimos los pasos 4, 5 y 6 solo que en este último en seleccionamos Package AUI Models. El resultado sería:



10. Podemos recolar el diagrama para visualizar mejor el resultado.

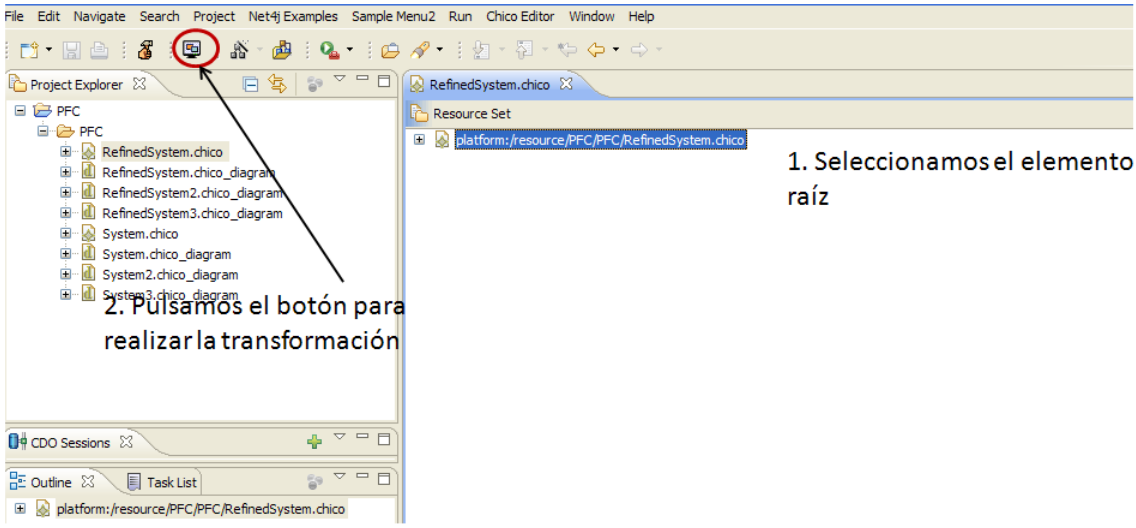


11. Podemos asignar valores a cada elemento de la interfaz concreta final, por ejemplo, seleccionamos el botón Salir del subset 0, y nos vamos a la sección de Properties. Podemos rellenar el ancho, el alto o el color que queremos que presente este elemento en la intefaz de usuario final.

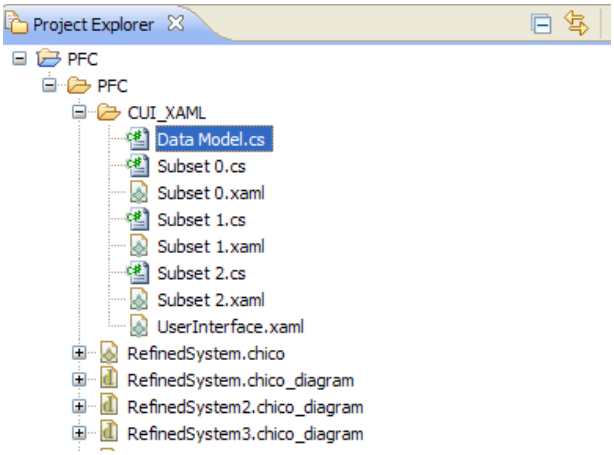


Obtención de la interfaz de usuario final

1. Abrimos RefinedSystem.chico. Una vez abierto, seleccionamos el elemento raíz del modelo y pulsamos el botón para realizar la transformación:



2. Cuando ejecutamos la transformación aparece una nueva carpeta llamada CUI_XAML en el mismo nivel que está el archivo System.chico. Si expandimos la estructura vemos que se han creado



3. De esta forma hemos obtenido el esqueleto de la interfaz de usuario para el problema de entrada y del modelo de datos del sistema.

Subset 0.cs:

```
using System;
using System.IO;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Navigation;


namespace SystemModels
{
 public partial class Subset0
 {
    public Subset0()
     {
       this.InitializeComponent();


       //Inserte el código necesario para la creación de objetos
debajo de este punto;


     }
    public void SetTextComponentResultadovalidación (String text)
     {

       Resultadovalidación.Text = text;
```

```
        //Inserte el código necesario para actualizar este atributo;

      }

     public void Salir (object sender, RoutedEventArgs e)
      {

        objeto.Salir( ); //introduzca aquí la clase respectiva como
parámetro

        //Inserte el código necesario para responder a este evento;

      }

  }
}
```

Subset 0.xaml

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        x:Class="SystemModels.Subset0"
        x:Name="Window"
        Title="Subset0"
        Width="0" Height="0">
        <GroupBox Header="Subset0" Canvas.Top="0" Canvas.Left="0"
Height="0" Width="0">
                <Canvas>
                <Label Width="0" Height="0"
Content="Resultadovalidación" Canvas.Left="0" Canvas.Top="0"
x:Name="ResultadovalidaciónLabel"/>
                <TextBox Width="0" Height="0" Canvas.Left="0"
Canvas.Top="0" x:Name="Resultadovalidación"/>
                <Button Content="Salir" Canvas.Top="0" Canvas.Left="0"
Height="0"Width="0" x:Name="Salir" Click="salir"/>
                </Canvas>
        </GroupBox>

</Window>
```