Universidad
Politécnica
de Cartagena

Universidad Politécnica de Cartagena

E. T. S. Ingeniería de Telecomunicaciones
Ingeniería de Telecomunicación

Lenguajes específicos de dominio gráficos y textuales: Un estudio comparativo

Proyecto fin de carrera

realizado por

Ángel López Moya

Director del proyecto:	Cristina Vicente Chicote
	Rafael Verdú Monedero
Co-Director:	Juan Francisco Inglés Romero

Índice general

Capítulo 1

Introducción

En la actualidad, el Desarrollo de Software Dirigido por Modelos (DSDM) [PGP10] es uno de los paradigmas de desarrollo software más en boga en el ámbito de la Ingeniería del Software. Las tecnologías disponibles en torno a este nuevo enfoque ofrecen una aproximación prometedora para superar las limitaciones expresivas de los lenguajes de programación de tercera generación, permitiendo a los diseñadores describir sistemas cada vez más complejos de manera más simple, gracias a la utilización de conceptos propios de sus dominios de aplicación. El DSDM busca, por lo tanto, elevar el nivel de abstracción utilizado durante las distintas etapas del ciclo de vida software. Hoy en día, es posible diseñar nuevos lenguajes de modelado soportados por editores, gráficos o textuales. El propósito de este Proyecto es proporcionar una posible forma de evaluar, de manera cuantitativa, tanto las herramientas disponibles para crear estos editores (tiempo necesario para su aprendizaje, flexibilidad a la hora de permitir introducir cambios, etc.), como los propios editores desarrollados con ellas (tiempo de desarrollo, usabilidad, escalabilidad de los modelos, etc.).

1.1. Objetivos

El primer objetivo que se plantea al inicio de este proyecto es aprender los fundamentos del DSDM, así como el manejo de varias herramientas que den soporte a este paradigma de desarrollo de software, en particular, algunas de las soportadas por la plataforma Eclipse[1] (EMF [FB03], GMF[2], Xtext[3]).

El segundo objetivo a realizar, es diseñar un lenguaje de modelado cuya sintaxis abstracta (meta-modelo) sea relativamente sencilla de implementar, y a partir de él, sendos editores de modelos: uno gráfico y otro textual. Posteriormente modificar la sintaxis abstracta del lenguaje para observar la repercusión que esto supone en los editores desarrollados.

Finalmente se proponen varias pruebas para cuantificar de manera empírica aspectos como el coste asociado al aprendizaje, la expresividad y la flexibilidad de las herramientas utilizadas

[1] Eclipse Project. `http://www.eclipse.org/`

[2] Graphical Modeling Project. `http://www.eclipse.org/modeling/gmp/`

[3] Xtext. `http://www.eclipse.org/Xtext/`

para implementar los editores. De la misma forma se proponen diferentes medidas para, de cara al usuario final, cuantificar la usabilidad de cada uno de los editores desarrollados, así como la estabilidad de los modelos que se pueden diseñar con cada uno de ellos.

1.2. Fases del Desarrollo del Proyecto

El desarrollo de este Proyecto se lleva a cabo siguiendo las etapas que se resumen a continuación:

1. **Fase de documentación y experimentación.**

   Estudio de la bibliografía relacionada con el DSDM, así como de las diferentes herramientas Eclipse que se van a utilizar durante el desarrollo del presente proyecto. Estudio de las técnicas para la evaluación empírica y comparación de lenguajes.

2. **Especificación de un lenguaje de modelado sencillo y de los editores de modelos.**

   Desarrollo de un meta-modelo sencillo, así como la creación de los editores gráfico y textual.

3. **Modificación del lenguaje de modelado y adaptación de los editores de modelos anteriores.**

   Se realiza alguna modificación en el meta-modelo creado anteriormente y se procede a adaptar los editores gráfico y textual. Esto permitirá comparar el impacto que tienen los cambios en el lenguaje en el proceso de desarrollo de cada editor.

4. **Definición de las métricas a utilizar para la comparativa de los editores.**

   Se diseñan las diferentes medidas que se utilizarán para comparar los editores que se han diseñado.

5. **Conclusiones del trabajo realizado en el proyecto y redacción de la memoria.**

   Se procede a evaluar el trabajo realizado durante el proyecto y se redacta la memoria.

1.3. Organización de la Memoria

El resto de la memoria se organiza como se indica a continuación:

- **Capítulo 2:** Lenguajes de modelado y técnicas para su evaluación.
- **Capítulo 3:** Desarrollo de un lenguaje de modelado para especificar arquitecturas de componentes.
- **Capítulo 4:** Planificación del Estudio para Evaluar Editores Gráficos vs Editores textuales.

- **Capítulo 5:** Conclusiones y Lineas de Trabajo Futuras.
- **Anexo A:** Editor Textual. Guía de usuario y Ejercicios
- **Anexo B:** Editor Gráfico. Guía de usuario y Ejercicios

Capítulo 2

Lenguajes de Modelado y Técnicas para su Evaluación

En este capítulo se introducen los conceptos básicos empleados en el desarrollo de este proyecto. En la primera sección se aborda el Desarrollo de Software Dirigido por Modelos (DSDM), dando una visión general sobre los fundamentos de este paradigma, así como también sobre el Eclipse Modeling Project y las diferentes herramientas empleadas. En la segunda sección se introducen diferentes técnicas para evaluar de forma empírica los lenguajes.

2.1. Desarrollo de software dirigido por modelos

El DSDM [PGP10] es un conjunto de técnicas y herramientas que dan soporte al desarrollo de lenguajes de modelado y con ellos modelar sistemas que mediante transformaciones modelo a modelo (M2M) y transformaciones modelo a texto (M2T) permiten obtener el código final de la aplicación.

2.1.1. Conceptos básicos

Comenzaremos hablando del concepto de modelo, ya que el DSDM se fundamenta en el uso de modelos y transformaciones de éstos. Un modelo es una representación de la realidad. El modelo deberá darnos una representación de algún aspecto de un sistema. Los modelos permiten abstraer la información no necesaria del sistema para representar únicamente aquella que es de interés, sobre todo, si éste es de una gran complejidad.

Un sistema puede representarse por uno o varios modelos, centrados en representar distintos aspectos de interés, a distintos niveles de abstracción. Así, es posible crear modelos de análisis, de diseño e incluso modelos de implementación más próximos a la plataforma de desarrollo que se emplearán para la codificación del sistema final. Los modelos de más alto nivel pueden evolucionar a otros de más bajo nivel, mediante la aplicación de transformaciones automáticas de modelos, hasta obtenerse un modelo lo suficientemente detallado como para poder generar, a partir de él, el código final del sistema. Así, el modelo (o conjunto de modelos) que repre-

senta al sistema junto con las transformaciones modelo-a-modelo y modelo-a-código, permiten automatizar, en gran medida, el proceso de desarrollo de software.

Para definir cualquier modelo es necesario contar con un lenguaje de modelado. Los meta-modelos definen formalmente la sintaxis abstracta de los lenguajes de modelado, recogiendo sus conceptos (palabras del lenguaje) y relaciones, así como las reglas que indican cómo se pueden combinar dichos conceptos para formar modelos válidos. De este modo, como indica Bézivin y Gerbé “Towards a precise definition of the OMG/MDA framework"[Ger01], “un modelo solamente será válido si es conforme a su meta-modelo". Por otra parte, la representación (ya sea gráfica o textual) asociada a cada uno de los elementos del meta-modelo (conceptos y relaciones) constituye la sintaxis concreta del lenguaje. Por último, la semántica asociada a los elementos de cada meta-modelo, esto es, su significado o interpretación, tiene impacto fundamentalmente en las transformaciones de modelos, en las que se define cómo transformar (interpretar) cada concepto del modelo de partida en términos del otro lenguaje (ya sea de modelado o de programación).

Las figuras 2.1 y 2.2 sitúan los conceptos anteriormente abordados. El esquema de la figura 2.1 ayuda a introducir la noción de meta-meta-modelo, que permite definir un lenguaje para especificar otros lenguajes, así, los meta-modelos originados serían conformes a su meta-meta-modelo. No obstante, dado que esta dinámica de ir acumulando términos “meta” podría resultar infinita, en el DSDM, los meta-meta-modelos se definen de forma reflexiva, es decir, se emplea el propio meta-lenguaje para definirse a sí mismo (ver pirámide en la figura 2.2), a este nivel, podemos encontrar el estándar MOF (MetaObject Facility Specification) [mof05] del Object Management Group (OMG) [omg03].

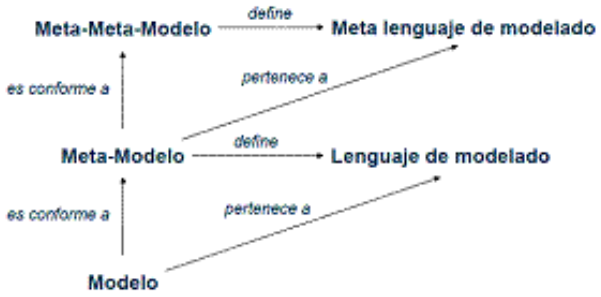

Figura 2.1: Relación modelo - lenguaje de modelado en DSDM.

Actualmente, el DSDM es uno de los paradigmas de desarrollo de software más en boga en el ámbito de la Ingeniería del Software. Los fundamentos sobre los que asienta el DSDM fueron establecidos hace ya un par de décadas. Sin embargo, el auge de este enfoque sólo ha sido posible en los últimos años gracias, por una parte, a la encomiable labor de estandarización que, en el ámbito del DSDM, ha llevado a cabo la OMG con su iniciativa MDA (Model-Driven Architectura) [omg03], y por otra, a la aparición en el mercado de las primeras herramientas que dan soporte a este nuevo enfoque, permitiendo explotar todo su potencial.

Entre las herramientas que actualmente dan soporte al DSDM, cabe mencionar las siguientes:

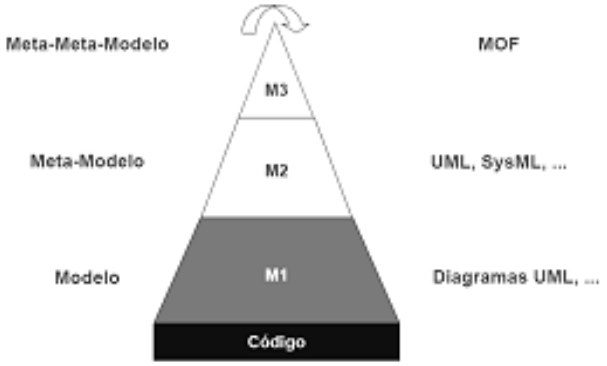

Figura 2.2: Esquema conceptual que sitúa los conceptos del DSDM.

DSL Tools (de Microsoft)[1], Meta-Edit+ (de la empresa Meta-Case)[2] y Eclipse[3]. Esta última es una plataforma abierta y de libre distribución que ofrece, entre otras muchas funcionalidades relacionadas con el desarrollo de software, un nutrido grupo de plug-ins relacionados con el DSDM. En los últimos años, Eclipse se ha convertido en un estándar de facto para la comunidad de DSDM, ya que implementa las principales tecnologías estandarizadas por el OMG para dar soporte a este enfoque.

2.1.2. Eclipse Modeling Project

Eclipse Modeling Project es un conjunto de proyectos relacionados con el modelado y con las tecnologías de DSDM utilizando la plataforma Eclipse. Está organizado en proyectos que proveen las siguientes capacidades: desarrollo de sintaxis abstracta, desarrollo de sintaxis concreta, transformación modelo-modelo, transformación modelo-texto.

En la figura 2.3 se puede ver el primer logo que fue propuesto. La imagen da una idea de la estructura y de sus áreas funcionales. Como se puede ver, EMF es el centro, aportando las capacidades básicas para desarrollar la sintaxis abstracta. EMFQuery, Validation y Transformation complementan la funcionalidad del núcleo de EMF para la gestión de instancias. En torno a estos componentes tenemos las tecnologías de transformación de modelos, ya sea modelo-a-texto (Java Emitter Templates y Xpand) o modelo-a-modelo (QVT y ATL). Despues, encontramos las herramientas de desarrollo de la sintaxis concreta: GMF (Graphic Modeling Framework), para la representación gráfica de los modelos y TMF (Textual Modeling Framework) para la representación textual de los modelos. Finalmente, una serie de proyectos y componentes satélites, que representan iniciativas que se están desarrollando para el Eclipse Modeling Project.

A continuación, se presentarán las principales tecnologías que se han utilizado en este proyecto.

[1]DSL Tools. `http://msdn.microsoft.com/en-us/library/bb126235.aspx`
[2]Meta Edit. `http://www.metacase.com/`
[3]Eclipse. `http://www.eclipse.org/`

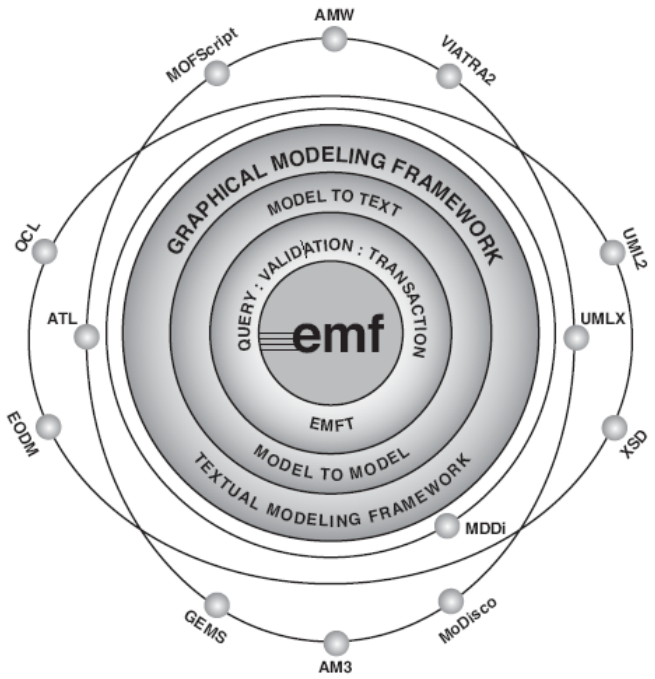

Figura 2.3: Eclipse Modeling Project.

2.1.2.1. Desarrollo de sintaxis abstracta mediante EMF

EMF [FB03] es el framework que proporciona la infraestructura básica para la creación de lenguajes de modelado en Eclipse. EMF unifica tres importantes tecnologías como son: Java, XML y UML [uml11] como se puede apreciar en la figura 2.4.

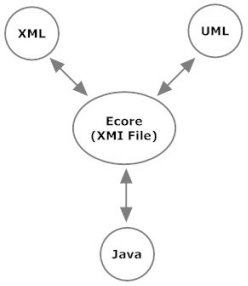

Figura 2.4: EMF unifica Java, XML y UML.

Imaginemos que queremos desarrollar una aplicación para manipular alguna estructura específica de mensajes XML. Deberíamos probablemente comenzar con un esquema de mensaje. Lo ideal sería tomar el esquema, apretando un par de botones conseguir un diagrama de clases UML, y apretando un par de botones más tener una serie de clases Java para manipular la implementación del XML. Y finalmente generar un editor para estos mensajes que funcione. Todo esto

es lo que consigue EMF. Además de el modelo EMF puede ser definido usando cualquiera de estas tecnologías sin necesidad de conocer bien cada una de ellas. A continuación detallaremos algunos aspectos principales de EMF.

- **Meta-modelo Ecore**

  El meta-meta-modelo usado para representar meta-modelos en EMF es Ecore. Ecore define la estructura de los objetos y proporciona una terminología común para describir un meta-modelo.

  A continuación, en la figura 2.5 se muestra un diagrama que define únicamente las partes principales necesarias para describir un meta-modelo Ecore.

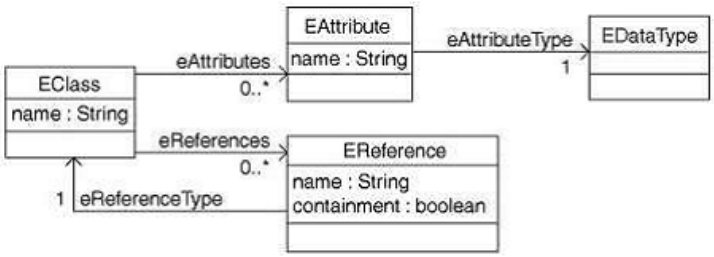

Figura 2.5: Conjunto simplificado del meta-modelo Ecore.

Las partes principales que podemos ver en la figura son las siguientes:

- EClass: Se utiliza para representar la clase modelada. Tiene un nombre, cero o más atributos y cero o más referencias.
- EAttribute: se utiliza para representar el atributo modelado. Los atributos tienen un nombre y tipo.
- EReference: Se utiliza para representar una relación entre clases. Tiene un nombre, un flag booleano para indicar si representar si es contenedor y una referencia a otra clase.
- EDataType: es usada para representar el tipo de un atributo. Un tipo de dato puede ser un tipo primitivo como un Integer o float, o un objeto como java.util.Date.

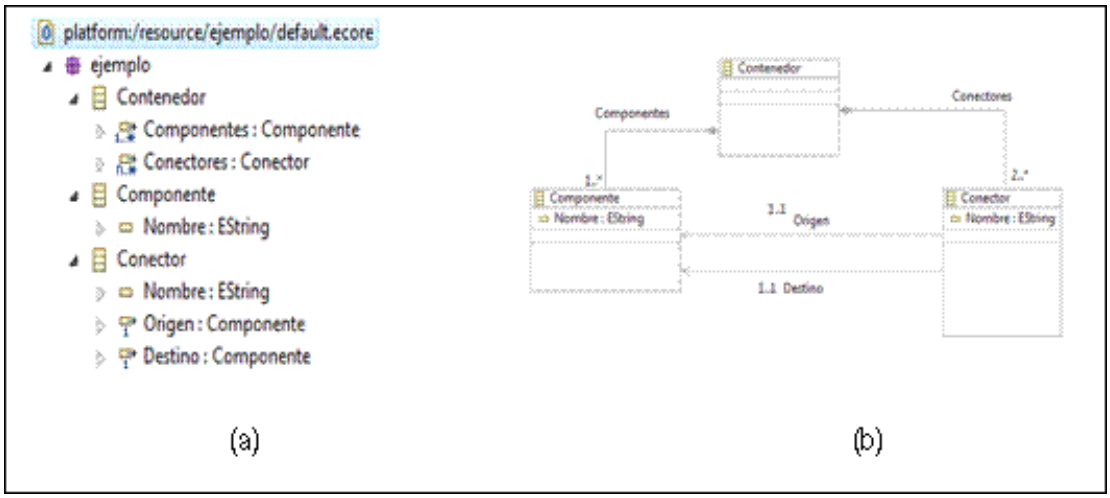

Figura 2.6: Ejemplo de meta-modelos. (a) Ejemplo con tree-editor. (b) Ejemplo con editor GMF.

- **Creación de un meta-modelo.**

  Se pueden crear meta-modelos usando EMF de diferentes formas. Utilizando el Tree-editor (un editor con una representación en árbol), implementando un conjunto de clases Java anotadas o importando un modelo UML2. Además, teniendo GMF (Graphical Modeling Framework) [4] instalado, se puede usar el Ecore diagram editor.

  Los meta-modelos creados se encontrarán en el workspace del proyecto, como un archivo .ecore. Abriendo este archivo con un editor de textos se verá que es una serialización XMI (XML Metadata Interchange) [Om02] del meta-modelo Ecore. Se pueden editar por lo tanto a mano, aunque es más recomendable utilizar el propio editor (tree-editor) que resulta más cómodo.

  A continuación supongamos un pequeño ejemplo para comprobar lo que sería la creación de un meta-modelo sencillo. Para ello utilizaremos el tree-editor que nos proporciona EMF. En la sintaxis abstracta utilizamos conceptos como las EClass (Componente y Conector), ambos tendran un EAttribute (nombre de tipo String) y la EClass Conector tendrá además dos EReference, puesto que un conector debe tener un origen y un destino y estos elementos harán referencia a ellos. En la figura 2.6a observamos el aspecto que tendría la sintaxis abstracta de nuestro ejemplo utilizando el tree-editor que nos proporciona EMF. En la figura 2.6b también se puede ver el aspecto que tendría con el editor Ecore diagram editor de GMF, con el que el meta-modelo se puede ver y definir gráficamente, se puede apreciar el parecido con los diagramas de clases UML.

  En la figura 2.7 se muestra el código XMI de la sintaxis abstracta que contiene el archivo .ecore que se creó con el tree-editor de EMF. En la figura se puede ver un elemento ”Contenedor” el cual resuelve una peculiaridad en EMF, que consiste en que las relaciones de

[4] The Eclipse Graphical Modeling Framework (GMF), `http://www.eclipse.org/modeling/gmf/`

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="ejemplo">
  <eClassifiers xsi:type="ecore:EClass" name="Contenedor">
    <eStructuralFeatures xsi:type="ecore:EReference"
        name="Componentes" lowerBound="1"
        upperBound="-1" eType="#//Componente" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference"
        name="Conectores" lowerBound="2"
        upperBound="-1" eType="#//Conector" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Componente">
    <eStructuralFeatures xsi:type="ecore:EAttribute"
        name="Nombre" e
        Type="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Conector">
    <eStructuralFeatures xsi:type="ecore:EAttribute"
        name="Nombre"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference"
        name="Origen" lowerBound="1"
        eType="#//Componente"/>
    <eStructuralFeatures xsi:type="ecore:EReference"
        name="Destino" lowerBound="1"
        eType="#//Componente"/>
  </eClassifiers>
</ecore:EPackage>
```

Figura 2.7: Ejemplo de meta-modelos. Vista del código XMI contenido en el archivo .ecore.

composición controlan la serialización de los modelos, por lo que siempre es necesaria la existencia de un elemento raíz que contenga a todo lo demás.

- **Generación de código del lenguaje y del editor de modelos**

  Se necesita un modelo generador que proporcione información adicional para la generación de código. En EMF este modelo generador tiene la extensión .genmodel y es básicamente un modelo decorador para el correspondiente .ecore. Este modelo generador utiliza Java Emitter Templates (JET) como motor de transformación modelo-a-texto para obtener el código Java desde el correspondiente meta-modelo definido con Ecore.

2.1.3. Graphic Modeling Framework

GMF proporciona una herramienta para generar editores gráficos a partir de meta-modelos EMF. Para ello, GMF utiliza diferentes plug-ins de Eclipse: EMF, para definir los meta-modelos, GEF para definir elementos gráficos de sintaxis concreta y QVT (Query/Validation/Transaction) y JET para las transformaciones modelo-a-modelo y modelo-a-texto respectivamente que soportan el proceso de generación de GMF y OCL (Object Constraint Language) [Spe06] para añadir restricciones sobre elementos y manejar, validar y controlar los modelos y meta-modelos.

Un proyecto GMF, cuyo flujo de trabajo está representado en la figura 2.8, permite crear una notación gráfica para un lenguaje de dominio específico. Se diseña una definición gráfica del modelo, creando las figuras (nodos, conexiones, contenedores y otros elementos) que serán representadas como elementos en el diagrama. Posteriormente se diseñará una paleta de herramientas que se utilizará para crear el diagrama. Finalmente se realiza un mapeo en el cuál, se fijan los elementos gráficos que se han diseñado a su correspondiente elemento de la paleta de

herramientas. Una vez conseguidos estos pasos se realizaría la tarea de generar el editor gráfico de modelos (diagrama), lo que daría como resultado un nuevo plug-in.

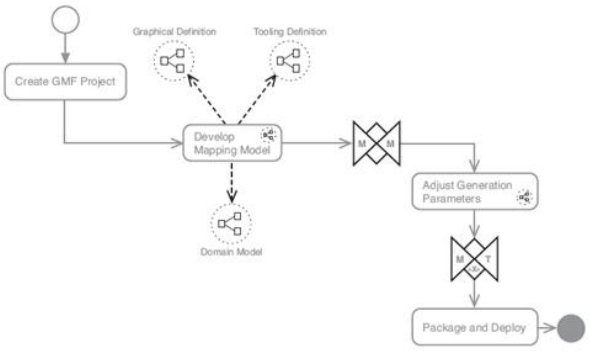

Figura 2.8: Flujo de trabajo de la herramienta GMF [Gro09].

2.1.3.1. Desarrollo de sintaxis concreta gráficas mediante GMF

- **Definición del meta-modelo**

  GMF parte de un meta-modelo, para ello es necesario utilizar EMF, definiendo la sintaxis abstracta y posteriormente creando el fichero .genmodel que nos permita generar el código del plug-in.

- **Definición de sintaxis gráfica**

  Hay que pensar en la definición gráfica del modelo como tres capas: La primera define la representación visual de los elementos del diagrama. La segunda define descriptores de figuras, referencias a accesorios de la primera capa, que se utilizará en las siguientes capas. Y en tercer lugar los elementos del diagrama son definidos para usarlos en el mapeo del modelo y poder contener información de elementos específicos del layout. Estas tres capas, proporcionan flexibilidad en la definición gráfica de modelos, porque permite el reuso de los elementos. Las figuras pueden ser rehusadas para construir otras figuras, los descriptores de figuras pueden ser utilizadas por múltiples elementos del diagrama, y el mismo elemento del diagrama puede utilizado en diferentes mapeos. Toda esta información queda recogida en un fichero de extensión .gmfgraph.

- **Definición de la paleta de herramientas**

  En este paso, se define la paleta de herramientas que se utilizará en el diagrama que se generará al final. Por lo tanto se tendrán que definir los iconos que describan cada uno de los elementos que estarán disponibles para ser utilizados. Entre otras cosas, a la paleta se le pueden añadir iconos, separadores y grupos de herramientas. Los grupos de herramientas o Tool Group, mantiene conjuntos de elementos agrupados, tiene sentido para agrupar

elementos similares, como pueden ser nodos, o conectores. Para utilizar estos Tool Group será necesario crear otro Tool Group que contenga al resto de Tool Groups. Toda esta información queda recogida en un fichero de extensión .gmftool.

- **Mapeo de la herramienta gráfica**

  El mapeo es el corazón de los modelos GMF y donde se representa en si mismo el diagrama. Es el elemento de unión de los dos modelos anteriores (gmfgraph y gmftool). El modelo de mapeo transformará varios modelos generadores con los que se crearán plantillas para generar código. En la figura 2.9 se puede ver como se organiza este modelo. La raíz del modelo es el elemento de mapeo, el cual tiene diferentes elementos "hijos" donde algunos de los principales son: Top Node Reference, Link Mapping, Canvas Mapping, Audit Container. Toda esta información queda recogida en un fichero de extensión .gmfmap.

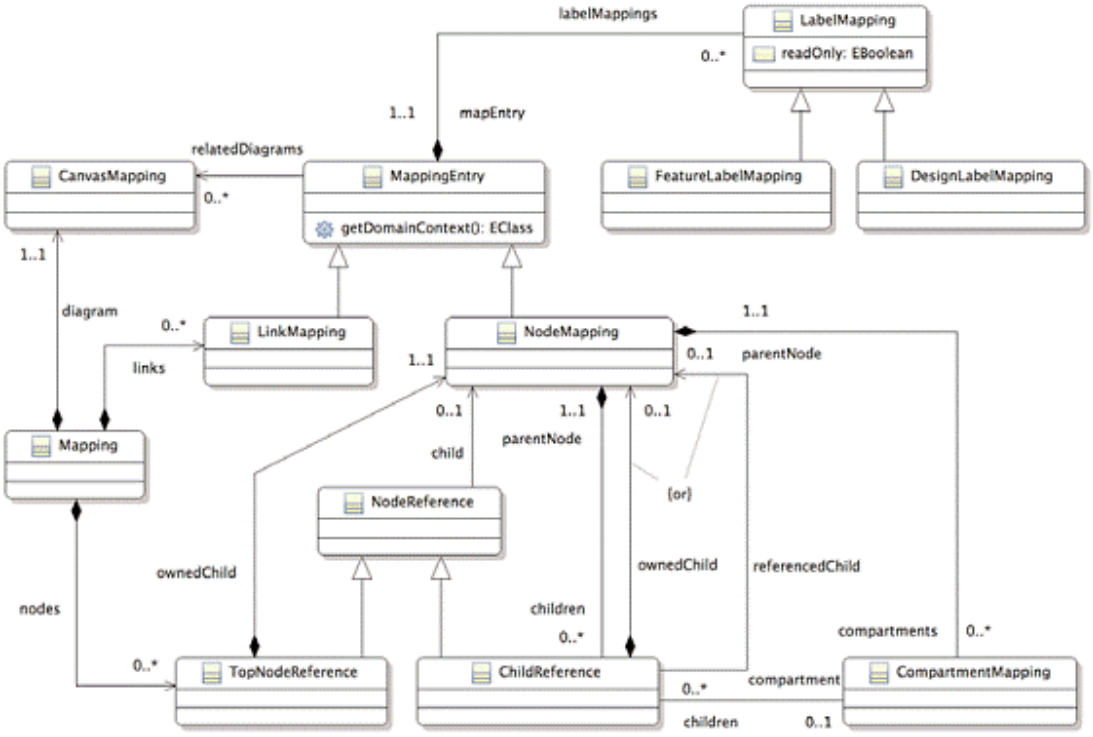

Figura 2.9: Modelo de mapeo [Gro09].

- **Generación de la herramienta gráfica**

  Una vez llegados a este punto, y si todos los pasos anteriores han sido validados correctamente, se generará a partir del mapping, un archivo de extensión .gmfgen con el que se generará todo el código necesario para arrancar un nuevo Eclipse desde el que se podrá ejecutar el editor gráfico. En la figura 2.10 se puede ver un ejemplo de lo que sería un editor gráfico.

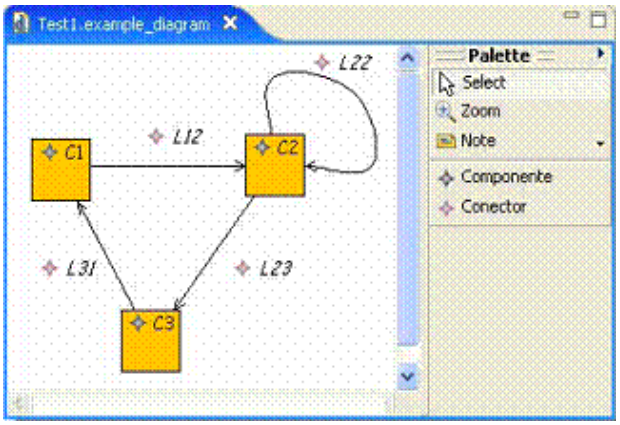

Figura 2.10: Ejemplo de editor gráfico.

2.1.4. Xtext

Xtext es un componente de TMF que soporta el desarrollo de la gramática de un DSL (Domain Specific Language) generando meta-modelos basados en un Ecore, editores de texto para Eclipse y su correspondiente analizador de sintaxis basado en ANTLR [P+09]. Xtext permite desarrollar un editor textual de modelos siguiendo dos posibles estrategias, por un lado, el diseñador puede escribir la gramática del lenguaje y, a partir de ella, generar el meta-modelo, y por otro, Xtext puede crear una gramática importando un meta-modelo existente, esta gramática podrá ser ajustada por el diseñador. Además, dado que Xtext está basado en el framework EMF, los modelos textuales escritos con un editor Xtext pueden ser utilizados en otras herramientas de DSDM de Eclipse, como QVT o JET.

2.1.4.1. Desarrollo de sintaxis concreta textual mediante Xtext

En la figura 2.11, se puede ver un ejemplo de la gramática de un modelo sencillo. Se puede destacar del código, que los elementos que identifican con un nombre seguido de ':' (DomainModel, Type, DataType, Entity, etc.) corresponden a elementos de tipo EClass en un meta-modelo. El primer elemento tiene la función de contenedor, tal y como se mencionó anteriormente, EMF requiere de un elemento raíz para la serialización de los modelos. En el código DomainModel contendrá elementos de tipo Type. Type a su vez, se define en el código de una forma genérica para identificar elementos de tipo DataType o Entity, lo que se puede interpretar como una relación de herencia. Por último, se puede ver como DataType, Entity y Feature, definen la forma en la que se utilizarán estos elementos con la sintaxis concreta del editor, es decir, los atributos a configurar, las palabras clave que serán necesarias añadir y los distintos elementos opcionales.

Algunos de los elementos principales que son interesantes comentar sobre Xtext para definir como será el lenguaje del editor son los siguientes:

- Las palabras o caracteres entre comillas simples, que se ven en azul, serán las palabras o caracteres obligatorios que se deberán escribir cuando se introducen estos elementos.

```
grammar org.example.domainmodel.Domainmodel with
                                    org.eclipse.xtext.common.Terminals

generate domainmodel "http://www.example.org/domainmodel/Domainmodel"

Domainmodel :
  (elements += Type)*
;

Type:
  DataType | Entity
;

DataType:
  'datatype' name = ID
;

Entity:
  'entity' name = ID ('extends' superType = [Entity])? '{'
     (features += Feature)*
  '}'
;

Feature:
  (many ?= 'many')? name = ID ':' type = [Type]
;
```

Figura 2.11: Ejemplo de código Xtext.

- Se pueden realizar múltiples tipos de asignaciones. Para expresar que la asignación es de tipo booleano, se utiliza *?*= o por ejemplo, para expresar que habrá múltiples valores +=.
- La cardinalidad de los elementos, se expresa añadiendo un paréntesis que englobe al elemento, añadiendo al final del paréntesis un elemento que pueden ser: *?* para expresar que un elemento es opcional, * para expresar que puede ser cualquier número de elementos (cero o más) o + para expresar que debe haber al menos un elemento o más.
- Los elementos que se introducen entre corchetes, expresan que estos elementos son referencias.

La figura 2.12 muestra un ejemplo de un modelo que ha sido desarrollado utilizando el editor textual generado con Xtext a partir del código que se ha descrito anteriormente en la figura 2.11.

2.2. Técnicas para la evaluación de lenguajes

En general, los estudios empíricos consisten en una recogida de datos que posteriormente son analizados para extraer resultados y conclusiones. Hay tres tipos principales de estrategias empíricas [Rob02]: experimentos, casos de estudio y encuestas.

En los siguientes apartados se describen los fundamentos para llevar a cabo un estudio empírico, esta información resultará de utilidad para alcanzar los objetivos del proyecto, en concreto, para definir la evaluación comparativa entre editores de modelado gráficos y textuales.

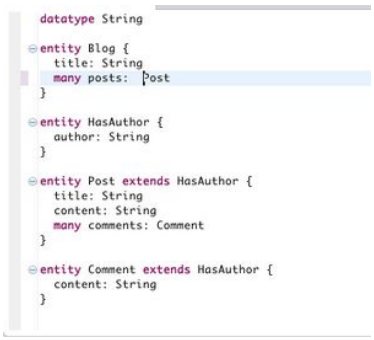

Figura 2.12: Ejemplo de editor textual con Xtext.

2.2.1. Experimentos

Un experimento es un procedimiento o examen formal mediante el cual se trata de comprobar una o varias hipótesis relacionadas con un determinado fenómeno, mediante la manipulación de la/s variable/s que presumiblemente son su causa [Rob02].

Los experimentos pueden variar mucho dependiendo de la disciplina pero todos persiguen el objetivo de obtener una explicación única sobre los resultados. Los experimentos serán más sólidos si al ser replicados por otros equipos se obtienen los mismos resultados. Un experimento bien planificado aportará resultados útiles que permitirán determinar cuándo determinadas afirmaciones son ciertas y en qué contexto se cumplen. Un característica muy interesante a tener en cuenta sobre los experimentos son las réplicas de los experimentos. A menudo será complicado obtener resultados feacientes de un único experimento.

2.2.1.1. Proceso para la realización de experimentos

Cualquier estudio empírico debería contar con los pasos que se describen a continuación, ya que partiendo de la siguiente base se pueden diseñar procesos de estudio para cualquier tipo de tarea. El proceso consta de seis pasos principales [WRH+12] como se puede ver en la figura 2.13.

- **Definición**: es la etapa en la que se determina "el porqué" del experimento. En esta fase se deberán definir los objetivos que se desean cumplir a partir del problema planteado.
- **Planificación**: En esta paso se define cómo se ve a llevar a cabo el experimento. Se puede dividir en seis puntos:
  - Selección de contexto:
    - Offline vs Online.
    - Estudiantes vs profesionales.
    - Simulación vs problemas reales.
    - Específico vs General.

- Formulación de hipótesis. Se establecen las dos hipótesis posibles del experimento.
- Selección de variables.
- Selección de sujetos.
- Diseño del experimento.
- Instrumentación.

- **Operación**: Se trata del paso en el que se obtendrán los datos para posteriormente ser analizados, es decir, la etapa en la que el experimento se lleva a cabo. Se puede dividir en tres etapas:

  - **Preparación**. Paso previo al inicio del experimento en el que por ejemplo se buscará al personal que participará en el experimento.
  - **Ejecución**. Realización del experimento.
  - **Validación de los datos**. Comprobación de que los datos son correctos y razonables.

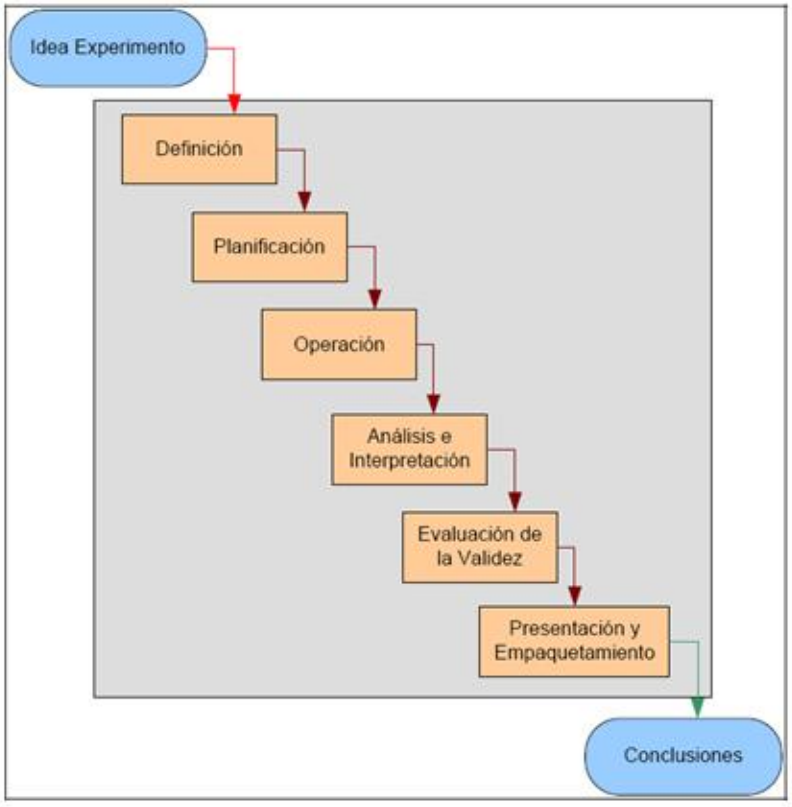

Figura 2.13: Visión general del proceso de realización de experimentos [MGR+08].

- **Análisis e interpretación**: Una vez se han obtenido los datos del experimento, se deben analizar. Para ello habrá que considerar qué tipo de técnicas se utilizarán para el análisis, teniendo en cuenta la naturaleza de los datos y el por qué de la realización del experimento. Se pueden realizar estudios estadísticos para determinar si la hipotesis nula se puede rechazar.

- **Evaluación de la validez**: En este paso se comprueba el grado de credibilidad del experimento. Esto depende entre otras cosas de las conclusiones que se obtengan del experimento. Es importante tener en cuenta en la fase de planificación la validez que tendrá posteriormente el experimento. Se pueden evaluar distintas amenazas para la validez de los experimentos como:

  - Validez de la conclusion: Corresponde a la fiabilidad o posibles errores de las medidas estadísticas que se han tomado en el experimento.
  - Validez de constructo: Corresponde a la forma en la que se ha realizado el proceso de experimentación (por ejemplo, métodos utilizados, operaciones, expectativas del experimento, etc.).
  - Validez interna: Se debe a causas internas del proceso de experimentación como puede ser los instrumentos utilizados.
  - Validez externa: Interacciones debidas al ambiente o entorno en el que se realiza el experimento.

- **Presentación y empaquetamiento**: Sería la forma en la que se presentarán los datos del experimento. Entre las diferentes opciones para realizar la presentación de los resultados del experimento se encuentran: realización de una conferencia, un informe, un artículo o algún tipo de material educativo.

2.2.2. Casos de estudio

Este tipo de procedimiento está relacionado con la investigación y es muy utilizado para monitorizar proyectos y actividades. Los datos se recogen para un objetivo específico de la investigación. Los casos de estudio son estudios basados en la observación de un proyecto o actividad que está en marcha mientras que los experimentos son estudios controlados.

Los experimentos y casos de estudio tienen similitudes en la mayoría de aspectos. En los casos de estudio es necesario establecer una buena base sobre la que evaluar los resultados del caso de estudio para evitar problemas de validez interna. Además hay que controlar los factores confusos, es decir, factores que pueden hacer difícil o imposible distinguir cuando un efecto es producido por un factor u otro.

2.2.2.1. Proceso para la realización de casos de estudio

Como en el caso anterior de los experimentos, se resumirán brevemente las etapas del desarrollo de casos de estudio, que en este caso propone Yin [Yin09]. En la figura 2.14 se pueden ver

las diferentes etapas que se describen a continuación.

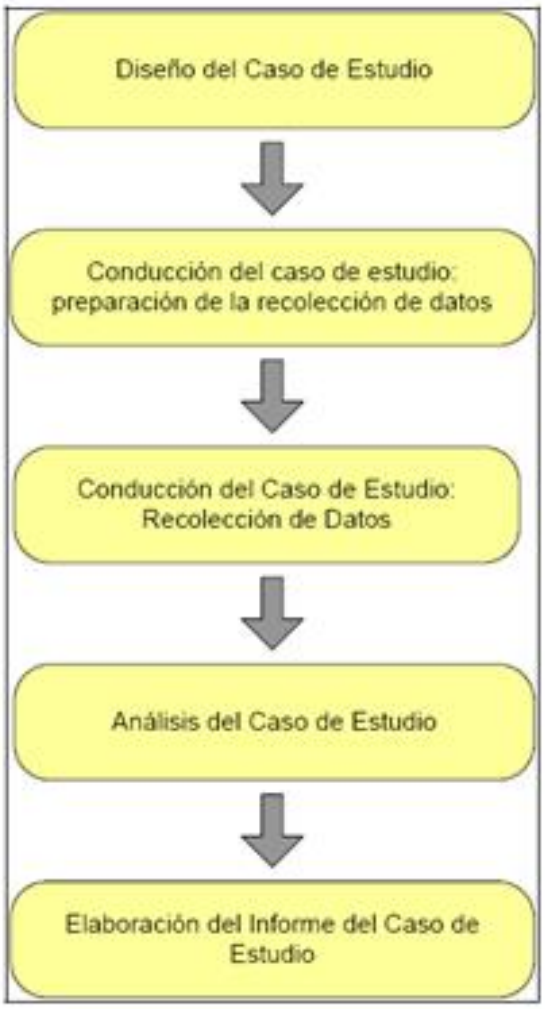

Figura 2.14: Etapas del desarrollo de casos de estudio [MGR+08].

Diseño del caso de estudio.

El diseño consiste en establecer los puntos de análisis en los que se centrará el caso de estudio.

- **Preguntas de estudio**: Estas dependen del objetivo que se busque, pueden ser:
  - ¿Qué? Cuando el objetivo es responder a preguntas de este tipo, por ejemplo, "¿qué se puede aprender?".
  - ¿Cómo? ¿Por qué? Cuando el objetivo es explicativo.
  - ¿Quién? ¿Dónde? ¿Cuánto? Para este tipo de preguntas, es más adecuado otro método de investigación como las encuestas.

- **Proposiciones**: Se trata de una proposición que se centre en lo que se va a estudiar.
- **Unidades de análisis**: Definen qué es el caso de estudio, a través de las preguntas formuladas en el caso de estudio.
- **Relación lógica entre preguntas y proposiciones**.
- **Criterios para interpretar los resultados**: Son los criterios que se seguirán para analizar los resultados que se obtengan.

Seguir los pasos anteriores facilita que el caso de estudio tenga un diseño que garantice ciertas características como pueden ser coherencia, validez interna y externa y fiabilidad.

Preparación de la colección de datos.

Este procedimiento requiere de varios pasos en los que el investigador o la persona que vaya a realizar el caso de estudio debe primero adquirir y comprender las habilidades que se necesitan para llevar a cabo del trabajo. Éstas serían habilidades como realizar buenas preguntas e interpretar respuestas, ser objetivo, adaptativo y flexible.

El siguiente paso sería el entrenamiento del caso de estudio. Es decir, un conocimiento profundo de la metodología y protocolos necesarios para llevar a cabo el caso de estudio, y de la materia sobre la que se va a realizar el estudio. El protocolo de un caso de estudio se puede resumir en un número de pasos que el investigador debe seguir. Estos pasos son los siguientes:

- La toma en contacto del proyecto, como puede ser la documentación sobre el tema, definir los objetivos y cuestiones a contestar.
- Cuestiones sobre el caso de uso, es decir, las preguntas que el investigador deberá tener en cuenta para la recogida de datos.
- Modo de presentación del caso de estudio. Formato con el que se tratarán los datos, el tipo de documentación que se utilizará para mostrarlos, etc.

El siguiente paso será identificar el caso de estudio más adecuado. A menudo se deberán realizar pequeños casos de estudio para obtener uno de ellos.

Finalmente, podría ser necesario realizar un caso de estudio previo para poder realizar iteraciones con las que ir mejorando las preguntas previstas, la recogida de datos y el diseño.

Recolección de datos.

Los datos pueden obtenerse de numerosas fuentes, algunas de ellas pueden ser:

- Archivos de registro.
- Entrevistas.

- Observación directa.
- La observación participante.
- Artefactos físicos.
- Documentación.

Análisis de los datos recogidos.

El análisis de datos consiste en realizar el tratamientdo correcto de los datos recogidos para poder utilizarlos en la proposición inicial del estudio. Es importante definir la estrategia que se seguirá antes de la preparación del a recogida de datos con el fin de que proceso se realice correctamente, es decir, que puedan ser interpretados razonablemente.

Elaboración del informe.

La elaboración del informa será la fomra en la que se presentarán los datos y las conclusiones obtenidas. El caso de uso ideal debería las siguientes condiciones:

- Debe ser significativo.
- Debe ser completo.
- Debe mostrar pruebas suficientes.
- Debe tener un atractivo que atraiga al lector.

2.2.3. Encuestas

A pesar de que la realización de encuestas es un medio ampliamente utilizado en investigaciones relacionadas con las Ciencias Sociales, se trata de un procedimiento útil para realizar evaluaciones empíricas en muchos otros dominios, por ejemplo, para examinar lenguajes de modelado [MGR+08]. Básicamente, los estudios basados en encuestas utilizan cuestionarios como medio principal para obtener información. Estos cuestionarios son completados por sujetos que plasman de forma escrita, oral o telemática (a través de un formulario Web) y de manera sencilla sus respuestas (subjetivas u objetivas) para cada una de las cuestiones planteadas.

Es muy importante que el investigador sólo proporcione la información indispensable, la mínima para que sean comprendidas las preguntas. Más información, o información innecesaria, puede derivar en respuestas no veraces. De igual manera, al diseñar la encuesta y elaborar el cuestionario hay que tomar en cuenta los recursos (tanto humanos como materiales) de los que se disponen, tanto para la recopilación como para la lectura de la información, para así lograr un diseño funcionalmente eficaz.

Según M. García Ferrando, "prácticamente todo fenómeno social puede ser estudiado a través de las encuestas"[Fer96], y se pueden ver las siguientes razones para constatar ésto:

- La encuesta es una técnica que permite realizar estudios sobre actitudes, valores, creencias y motivos.
- Las encuentas se pueden adaptan a cualquier tipo de información y a cualquier población.
- Las encuestas son un medio de obtener información de los sujetos respecto a experiencias que éstos han vivido.
- Las encuestas son un método rápido de obtener información fácilmente clasificable para su posterior estudio.

Las encuestas se pueden clasificar atendiendo al ámbito que abarcan, a la forma de obtener los datos y al contenido, de la siguiente manera:

- **Exhaustivas o parciales**: Se considera que una encuesta es exhaustiva cuando abarca a todas las unidades estadísticas que componen el conjunto estudiado, en caso contrario, se considera parcial.
- **Directas e indirectas**: Se considera directa cuando los datos a medir se observa directamente en el propio cuestionario. Se considera indirecta cuando los datos que se quieren medir se obtienen de una forma indirecta, deduciéndolos o averiguándolos a través del propio cuestionario.
- **Sobre hechos o de opinión**: Son aquellas que tienen por objetivo obtener información sobre lo que el público piensa acerca de una determinada cuestión. Se suele realizar un procedimiento de muestreo y se aplican a determinada parte de la población.

Hay que tener en cuenta que las encuestas de opinión no indican necesariamente lo que los sujetos piensan acerca de una cuestión, sino lo que pensaría si le planteásemos una pregunta a ese respecto, ya que hay personas que no tienen una opinión formada sobre lo que se les pregunta y contestan con lo que dicen los periódicos y las revistas. A veces las personas encuestadas tienen más de una respuesta a una misma pregunta dependiendo del marco en que se le haga la encuesta y por consecuencia las respuestas que se dan no tienen por qué ser sinceras. Las encuestas sobre hechos se realizan sobre acontecimientos ya ocurridos, hechos materiales.

Los cuestionarios pueden ser:

- **Individuales**: El sujeto contesta de forma autónoma al cuestionario, es decir, el investigadoro entrevistador no participa.
- **Cuestionario-lista**: El sujeto responde a preguntas formuladas por un entrevistador.

Con respecto a las preguntas de los cuestionarios, se pueden clasificar estas según la forma de respuesta de estas preguntas como se ve a continuación:

- **Preguntas cerradas**: Se le presentan al sujeto una serie de opciones entre las que tendrá que escoger la respuesta. La ventaja de este método es la facilidad de leer los resultados así como un proceso mas liviano para el sujeto que va a ser encuestado. Una desventaja de este método es realizar un mal diseño de las opciones posibles. Una forma de evitar esto último es realizar un estudio piloto con el que se pueda realizar una fase de mejora del cuestionario.

- **Preguntas abiertas**: El sujeto puede responder libremente a las preguntas que se le hacen. La ventaja de este método es que las respuestas tendrán una gran riqueza, pero a su vez esto puede complicar el proceso de estudio posterior, debido a la posible variedad de respuestas.

Para la realización de un cuestionario eficaz y útil, conviene seguir ciertas reglas para su diseño. Se pueden destacar las siguientes:

- La encuesta debería colocar a todos los sujetos en idéntica situación psicológica. Ésto permite estandarizar la obtención de datos, para así hacerlos comparables.

- No conviene realizar un número alto de preguntas (más de 30) y éstas deben ser mayoritariamente cerradas y númericas.

- El lenguaje usado debe ser conocido por los sujetos. Si un término puede no ser entendido, debe sustituirse, o explicarse brevemente en el enunciado. Tambíen debe evitarse utilizar lenguaje o términos ambiguos que den lugar a errores de interpretación de la pregunta.

- Las preguntas no deberían dar lugar a respuestas que requieran grandes esfuerzos de memoria, consultar información o realizar cálculos númericos.

- Las preguntas deben ir dirigidas en una sola dirección o idea.

- Las preguntas deben formularse de forma que se eviten respuestas condicionadas.

Anteriormente se comentó la posible realización de un cuestionario o estudio piloto para mejorar el cuestionario final. La realización de una prueba piloto con la que se pueda experimentar si el cuestarionario esta bien diseño nos permitirá detectar preguntas que no se contestan por alguna razón, encontrar que las preguntas abiertas que se dejan al final contestan algunas de las preguntas anteriores que quizás incluso se dejaron en blanco, por lo que nos permitiría incluir estas preguntas como nuevas opciones en las preguntas cerradas. La prueba piloto también nos podría ayudar a detectar una excesiva respuesta «intermedia», como respuesta comodín, por lo que nos llevaría a intentar evitar esto.

Capítulo 3

Desarrollo de un Lenguaje de Modelado para Especificar Arquitecturas de Componentes

En este capítulo de la memoria se mostrarán los pasos a realizar para el desarrollo de un lenguaje de componentes, comenzando con el desarrollo de la sintaxis abstracta (meta-modelo) y sus restricciones OCL de las que dispondrá el lenguaje y posteriormente, se verán los pasos a realizar para desarrollar el editor gráfico con GMF, y el editor textual con la herramienta Xtext.

3.1. Lenguaje de modelos de componentes

En la primera fase del desarrollo del proyecto, el objetivo era crear un meta-modelo mediante el cual quedaría definido nuestro lenguaje de componentes. En primer lugar se definieron dos versiones distintas del meta-modelo, para posteriormente estudiar el impacto que esto supondría en las herramientas de modelado gráfico y textual. En la figura 3.1 se puede ver el meta-modelo definido para la primera versión. Además se mostrarán los pasos necesarios para poder crear estos meta-modelos y la definición de las restricciones que se aplican.

3.1.1. Primera versión del meta-modelo

En esta primera versión el meta-modelo consiste en un conjunto de componentes que se interconectan entre ellos mediante conectores. Cada componente tiene una serie de interfaces, que pueden tener dos tipos de servicio, Required y Provided. Los conectores tendrán como principio y fin de la conexión a estas interfaces. Por otro lado, los componentes simples contendrán únicamente sus interfaces, pero habrá además componentes compuestos que albergarán en su interior a otros componentes, ya sean simples o compuestos. Estos componentes compuestos serán también los que contengan a los conectores, por lo que el componente "root" será un componente compuesto en si mismo.

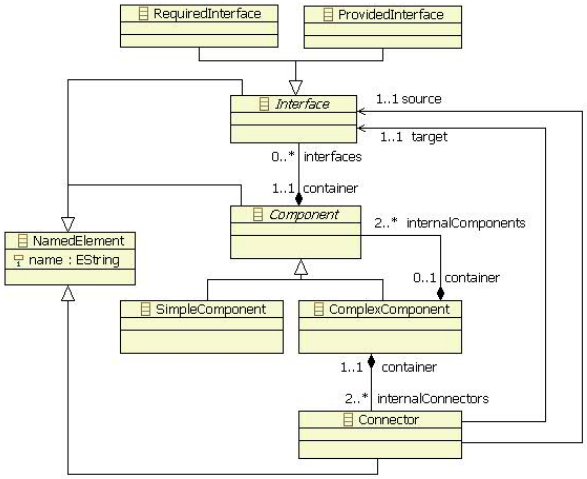

Figura 3.1: Meta-modelo de componentes, versión sencilla.

3.1.1.1. Descripción de los pasos para crear el meta-modelo

A continuación se irán definiendo los pasos necesarios para llegar al meta-modelo de la figura 3.1 mediante Eclipse.

1. El primer paso será crear un nuevo proyecto en Eclipse como vemos en la figura 3.5.

2. Definición del meta-modelo

   Despues haciendo click derecho encima del proyecto que hemos creado, elegiremos New->Other... y en la carpeta Ecore Tools elegiremos Ecore Diagram. Despues elegiremos el nombre para nuestro meta-modelo y finalizar. Una vez aquí, tendremos en el proyecto dos archivos, uno con extensión .ecore y otro con extensión .ecorediag , desde este último accederemos a un editor que nos permitirá construir nuestro meta-modelo mediante una paleta de herramientas. De esta forma deberemos ir escogiendo las EClass, EAttribute, EReference, etc que sean necesarias para construir nuestro meta-modelo. El otro archivo de extensión .ecore por defecto nos mostrará una vista tipo tree-editor. Ambos archivos están sincronizados, por lo que cualquier cambio en uno de ellos se actualizará en el otro.

3. Validación del meta-modelo

   Una vez completado el meta-modelo, el siguiente paso es realizar el proceso de validación para comprobar que no ha habido ningún error, como referencias que no apunten a nada. Para ello nos dirigiremos al menú Diagram y ahí encontraremos la opción Validate.

4. Generación del EMF Generator Model

   Para hacer esto, añadiremos al proyecto un nuevo archivo como se hizo para crear el Ecore Diagram, pero esta vez de la carpeta Eclipse Modeling Framework seleccionaremos EMF

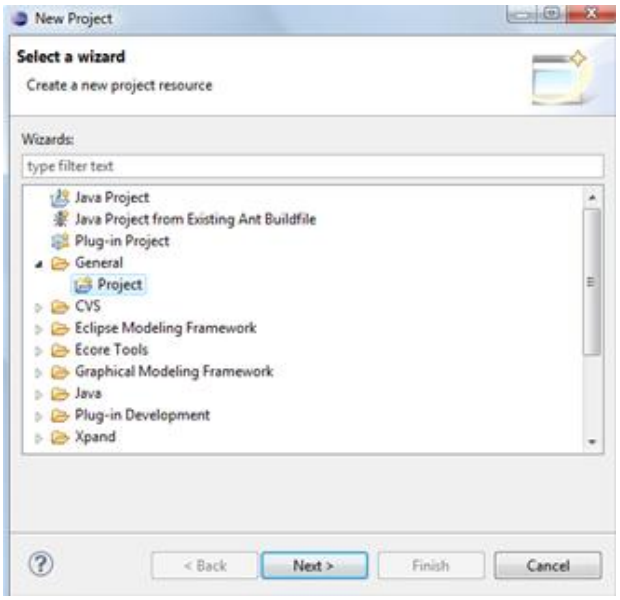

Figura 3.2: Creación de un nuevo proyecto.

Generator Model. Durante los pasos para su creación tendremos que seleccionar el .ecore de nuestro meta-modelo. Este fichero reunirá un conjunto de propiedades e información del cual se podrá obtener información posteriormente para generar archivos .java con la información del meta-modelo.

5. Generación de código JAVA

   Este sencillo paso consiste en desplegar el menú del .genmodel con un click derecho del ratón y seleccionar Generate All. Con esto se generarán un conjunto de archivos Java que nos dará la opción de arrancar una nueva instancia de Eclipse y utilizar nuestro meta-modelo como un plug-in.

3.1.1.2. Descripción de las reglas OCL aplicadas a esta versión del meta-modelo

OCL (Objetct Constrqaint Language) es usado generalmente para definir restricciones a modelos. También puede utilizarse para configurar plantillas de comportamiento durante la ejecución, inicializar características en modelos, definir métricas en modelos y servir como base de transformación en lenguajes. OCL incorpora una consola en la que se pueden realizar test de las sentencias que se apliquen a los elementos del modelo.

En este proyecto la implementación de las sentencias OCL se realizó a través de la herramienta OCLinEcore que nos permite introducir las restricciones directamente en el meta-modelo, de forma que al diseñar otros modelos posteriormente con herramientas como GMF o Xtext, las restricciones se aplicarán directamente.

En la figura 3.3 se puede ver un ejemplo de cómo se insertan las reglas OCL dentro de nuestro meta-modelo. Para ello debemos ir al fichero .ecore y abrir con OCLinEcore Editor. Una

vez hecho esto veremos el código de nuestro meta-modelo en el cual tendremos simplemente que añadir la palabra clave invariant dentro de la clase a la que queramos añadir la restricción y ponerle un nombre. Tras los dos puntos se insertará el código que expresa la regla.

```
abstract class Component extends NamedElement
{
    invariant DistintoNombre:
    Component.allInstances()->forAll(c1 : Component, c2 : Component | c1 <> c2 implies c1.name <> c2.name);
    property container#internalComponents : ComplexComponent[?] { ordered };
    property ports#container : Port[+] { ordered composes };
}
```

Figura 3.3: Regla OCL insertada mediante OCLinEcore.

Las reglas que se han incluido en este meta-modelos son las siguientes:

1. No puede haber componentes con nombres iguales.
2. Los puertos unidos por conectores de distinto nivel, deben tener el mismo tipo y nombre.
3. Los puertos unidos por conectores de un mismo nivel deben ser de distinto tipo, mismo nombre.

La definición de estas reglas son las que se muestran en la tabla siguiente:

Regla	Elemento	Definición
1º	Componente	Component.allInstances()->forAll(c1 : Component, c2 : Component | c1 <>c2 implies c1.name <>c2.name);
2º	Conector	self.source.container = self.target.container.container or self.source.container.container =self.target.container and self.source.interfaces->notEmpty() implies self.source.interfaces->forAll(i : Interface | self.target.interfaces->exists(j : Interface | i.name = j.name and (i.oclIsTypeOf(RequiredInterface) and j.oclIsTypeOf(RequiredInterface) or j.oclIsTypeOf(ProvidedInterface) and i.oclIsTypeOf(ProvidedInterface))));
3º	Conector	self.source.container.container = self.target.container.container and self.source.interfaces->notEmpty() implies self.source.interfaces->forAll(i : Interface | self.target.interfaces->exists (j : Interface | i.name = j.name and (i.oclIsTypeOf(RequiredInterface) and j.oclIsTypeOf(ProvidedInterface) or j.oclIsTypeOf(RequiredInterface) and i.oclIsTypeOf(ProvidedInterface))));

Cuadro 3.1: Reglas OCL para la segunda versión del meta-modelo.

3.1.2. Segunda versión del meta-modelo

En esta segunda versión del meta-modelo se buscaba añadir un punto mas de complejidad al meta-modelo. Para ello se incluyo una nueva clase, Port, de forma que esta clase sea la que albergue las interfaces. Por lo que el meta-modelo sigue siendo como la primera versión con la salvedad de que los componentes tanto simples como compuestos tienen en su interior Ports en lugar de Interfaces. En la figura 3.4 se puede ver el meta-modelo definido.

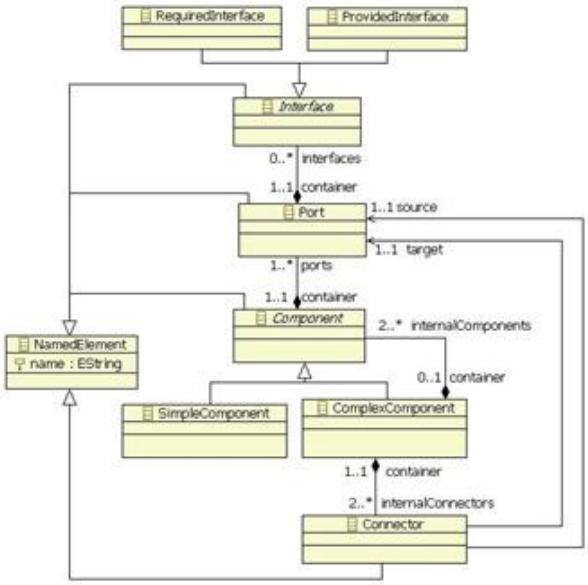

Figura 3.4: meta-modelo de componentes, versión con Puertos.

3.1.2.1. Descripción de los pasos para crear el meta-modelo

La creación del meta-modelo resulta idéntica a la de la primera versión por lo que podemos remitirnos al apartado 3.1.1.1. en el que se hablaba sobre los pasos necesarios para llegar al meta-modelo de la figura 3.4.

3.1.2.2. Descripción de las reglas OCL aplicadas a esta versión del meta-modelo

En esta segunda versión, el meta-modelo a necesita de algunas reglas más de las que tenía la primera versión, puesto que se tendrá que añadir alguna restricción al componente Port. Las reglas que se han incluido en este meta-modelos son las siguientes:

1. No puede haber componentes con nombres iguales.

2. Los puertos unidos por conectores de distinto nivel, deben tener el mismo tipo y nombre.

3. Los puertos unidos por conectores de un mismo nivel deben ser de distinto tipo, mismo nombre.

4. Las interfaces contenidas en un mismo puerto deben tener nombres distintos.

La definición de estas reglas son las que se muestran en la tabla siguiente:

Regla	Elemento	Definición
1º	Componente	Component.allInstances()->forAll(c1 : Component, c2 : Component | c1 <>c2 implies c1.name <>c2.name);
2º	Conector	self.source.container = self.target.container.container or self.source.container.container = self.target.container and self.source.interfaces->notEmpty() implies self.source.interfaces->forAll(i : Interface | self.target.interfaces->exists(j : Interface | i.name = j.name and (i.oclIsTypeOf(RequiredInterface) and j.oclIsTypeOf(RequiredInterface) or j.oclIsTypeOf(ProvidedInterface) and i.oclIsTypeOf(ProvidedInterface))));
3º	Conector	self.source.container.container = self.target.container.container and self.source.interfaces->notEmpty() implies self.source.interfaces->forAll(i : Interface | self.target.interfaces->exists (j : Interface | i.name = j.name and (i.oclIsTypeOf(RequiredInterface) and j.oclIsTypeOf(ProvidedInterface) or j.oclIsTypeOf(RequiredInterface) and i.oclIsTypeOf(ProvidedInterface))));
4º	Port	self.interfaces->forAll(i : Interface | self.interfaces->exists(j : Interface | i.name <>j.name)) or self.interfaces->size() = 1;

Cuadro 3.2: Reglas OCL para la segunda versión del meta-modelo.

3.2. Editor gráfico de modelado

Una vez definido el meta-modelo, el siguiente paso consiste en desarrollar la herramienta gráfica utilizando para ello el plug-in GMF de Eclipse, del que ya se ha hablado anteriormente. El objetivo de una herramienta gráfica es tener un editor que sea amigable, fácil de usar. El usuario no necesitará tener un conocimiento profundo de cada componente, tendrá que saber encajar las piezas (cajas negras).

En la figura **??** se puede ver el aspecto que tendría el editor gráfico diseñado con GMF. Se pueden apreciar varias partes principales: La zona de trabajo, donde se irán colocando los

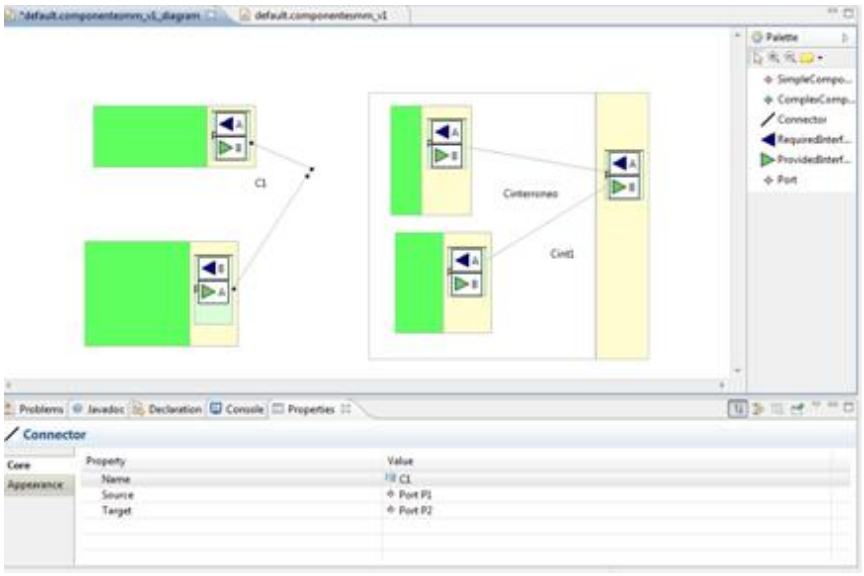

Figura 3.5: Herramienta gráfica diseñada con GMF.

elementos del modelo, la paleta situada a la derecha, de la que se escoge cada elemento a incluir en el diseño, y finalmente en la parte inferior podemos ver la vista de propiedades en las que se pueden editar las propiedades de cada elemento del modelo.

3.2.1. Descripción de los pasos para crear la herramienta gráfica con GMF

A continuación se irán definiendo los pasos necesarios para llegar a la herramienta de la figura **??** mediante el plug-in GMF de Eclipse. El desarrollo de una herramienta gráfica, consta de 3 pasos como ya vimos en el capítulo anterior.

1. Definición del apartado gráfico (.gmfgraph).

   El primer paso consiste en definir el aspecto gráfico de cada elemento de nuestra sintaxis. Para ello añadiremos al proyecto un nuevo archivo .gmfgraph que será un Simple Graphical Definition Model como se puede ver en la figura 3.6 Durante la creación de este archivo, se deben ir seleccionando varias opciones, la primera de ella será seleccionar el meta-modelo. Seguidamente habrá que seleccionar qué elemento será la raiz o root, que en este caso sería el ComplexComponent. Otra elección que se debe escoger, será la de definir qué elementos son de tipo “forma” y cuales de tipo “enlace”. Los elementos de tipo “forma” son aquellos que estarán representados por figuras geométricas o iconos y los de tipo enlace que estarán representados por líneas o flechas, que se encargarán de unir los de tipo “forma”.

   Una vez el fichero esta creado, se debe comenzar a definir los elementos gráficos de cada componente del meta-modelo. En la figura 3.7 se puede ver una vista de lo que sería el fichero gmfgraph para la segunda versión del meta-modelo. Podemos diferenciar unos 4 tipos principales de elementos:

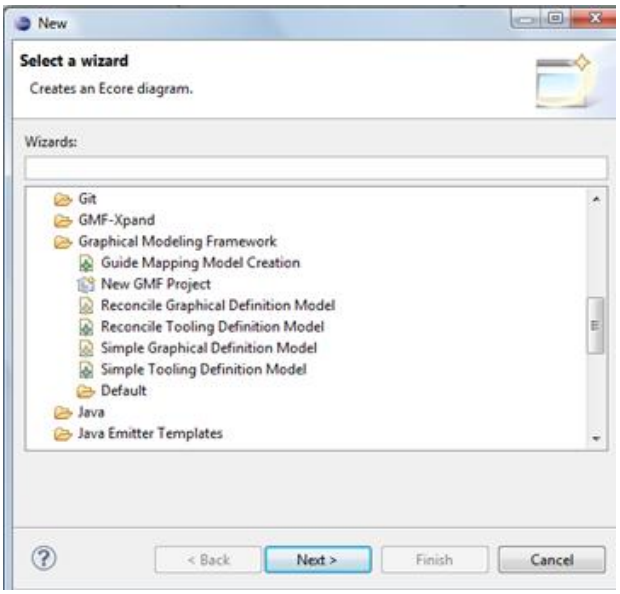

Figura 3.6: Menú con los diferentes archivos GMF.

- Los Figure Descriptor que están contenidos en el Figure Gallery. Contienen la representación gráfica de cada elemento a incluir en nuestro modelo.
- Los Node que mapean cada Figure Descriptor con el elemento del meta-modelo.
- Los Diagram Label permite configurar algunos aspectos de las etiquetas, como el uso de un icono, por lo que relaciona con el Figure Descriptor.
- Los Comparment sirve para hacer que un objeto pueda contener a otros en su interior.

En la figura 3.8 se puede ver desplegado los diferentes elementos que contiene la figura gráfica del ComplexComponent y el Connector. Se puede ver como se definen diferentes características del ComplexComponent, como su forma, que sera un rectángulo y dentro de el, características, como el fondo, el borde, y además los rectángulos interiores que contendrán a otros componentes o puertos. La figura gráfica que representa al conector se basa en una línea. En ambas figuras se puede ver un elemento común, los Child Access que sirven como elemento de acceso a dicha figura.

2. Definición de la paleta de herramientas (.gmftool).

   El segundo paso consiste en definir la paleta de herramientas que nos permita elegir cada elemento para posteriormente situarlo en la zona de trabajo para crear el modelo. Para ello añadiremos al proyecto un nuevo archivo .gmftool que será un Simple Tooling Definition Model como se puede ver en la figura 3.6 Durante la creación de este archivo, se deben seleccionar de nuevo el meta-modelo con el que se trabaja y que elementos serán de tipo forma y cuales de tipo enlace.

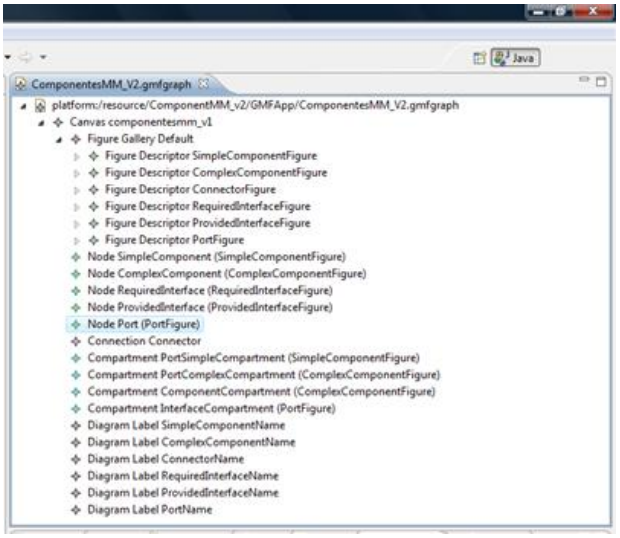

Figura 3.7: Archivo .gmfgraph

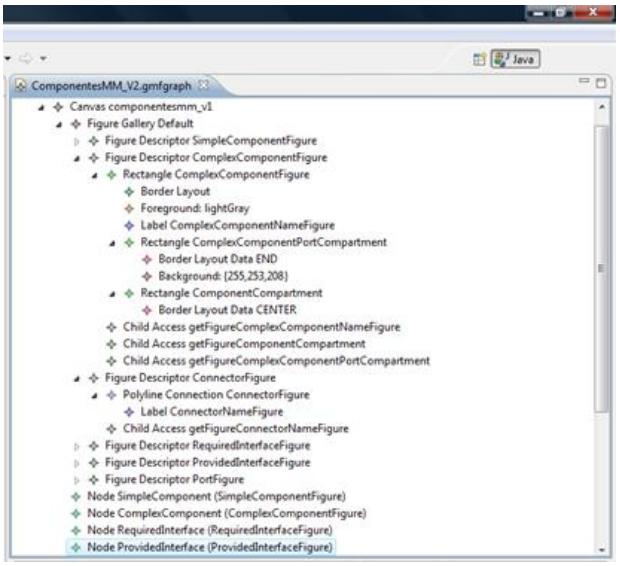

Figura 3.8: Definición de objetos gráficos en el archivo .gmfgraph

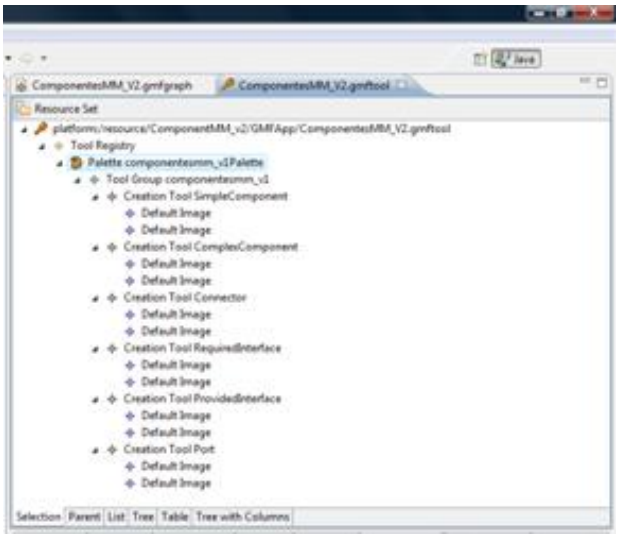

Figura 3.9: Fichero .gmftool.

En la figura 3.9 se puede ver la paleta de herramientas que se ha obtenido por defecto a partir de nuestro meta-modelo.

3. Definición del mapeado entre los elementos gráficos y la paleta de herramientas (.gmfmap).

   El tercer paso consiste en realizar el mapeado de los elementos gráficos definidos en el gmfgraph y la paleta de herramientas gmftool. Para ello añadiremos al proyecto un nuevo archivo .gmfmap que será un Guide Mapping Model Creation como se puede ver en la figura 3.6. Durante la creación de este archivo, se debe seleccionar de nuevo el meta-modelo con el que se trabaja y los archivos gmfgraph y gmftool, además en un último paso por defecto se ofrece una propuesta de mapeo que normalmente es necesario revisarla y modificarla porque no suele ser completamente acertada. Finalmente habrá que realizar la validación para comprobar que todo es correcto.

   En la figura 3.10 se puede ver el resultado final que tendría el archivo .gmfmap de la segunda versión del meta-modelo.

   Se puede ver ciertas partes características de la estructura del archivo:

   - Los Top Node Reference de los cuales cuelgan los diferentes elementos que contienen representados por los Child Reference.
   - Los cuales Child Reference hacen referencia a los Node Mapping en los que se define el mapeo de ese elemento en concreto.
   - Los Compartment Mapping, hacen referencia a la propiedad de contención que puedan tener los componentes, como en nuestro caso, que un ComplexComponent contendrá otros componentes y puertos o los Ports que contendrán interfaces.

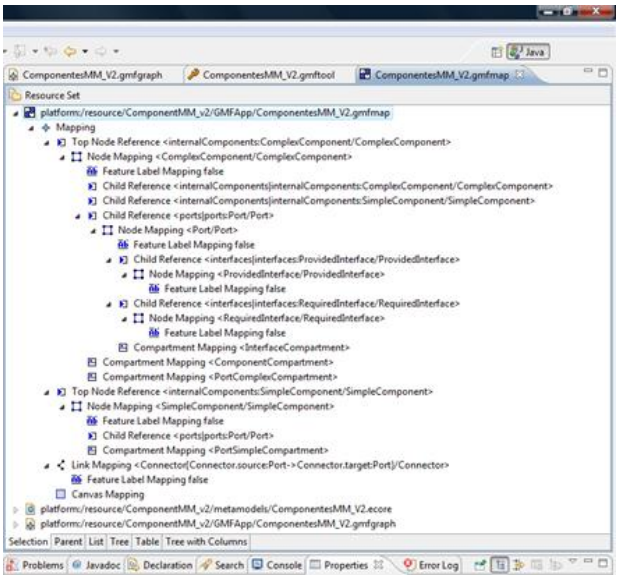

Figura 3.10: Fichero .gmfmap.

4. Generación de la herramienta gráfica (.gmfgen).

   El cuarto paso consiste en generar un archivo de extensión .gmfgen que tendrá una función similar a la tenia el genmodel del capítulo anterior. Este archivo se genera a partir del gmfmap, haciendo click derecho y eligiendo la opción Create generador model. Una vez disponemos del gmfgen, haciendo click derecho sobre el se elige la opción Generate diagram code, que generará todo el código necesario para poder iniciar nuestra herramienta gráfica como una nueva instancia de Eclipse.

3.3. Editor textual de modelado

En este apartado, como ocurrió en el anterior, se describirá el desarrollo de la herramienta textual utilizando para ello el plug-in Xtext de Eclipse, del que ya se ha hablado anteriormente. El objetivo de una herramienta textual es el poder desarrollar un modelo utilizando un lenguaje cómodo e intuitivo, similar a como serían los lenguajes de programación conocidos.

En las figura 3.11 y 3.12 se puede ver el fichero completo de Xtext en el que se describe completamente el lenguaje textual que se utilizará en crear el lenguaje que se usará en el editor textual para implementar los modelos.

3.3.1. Descripción de los pasos para crear la herramienta textual con Xtext

A continuación se irán definiendo los pasos necesarios para conseguir lo que se ve en las figuras 3.11 y 3.12 mediante el plug-in Xtext de Eclipse. Hay dos métodos para crear un proyecto Xtext. El primero de ellos consiste en crear un proyecto Xtext normal, y desarrollar desde cero

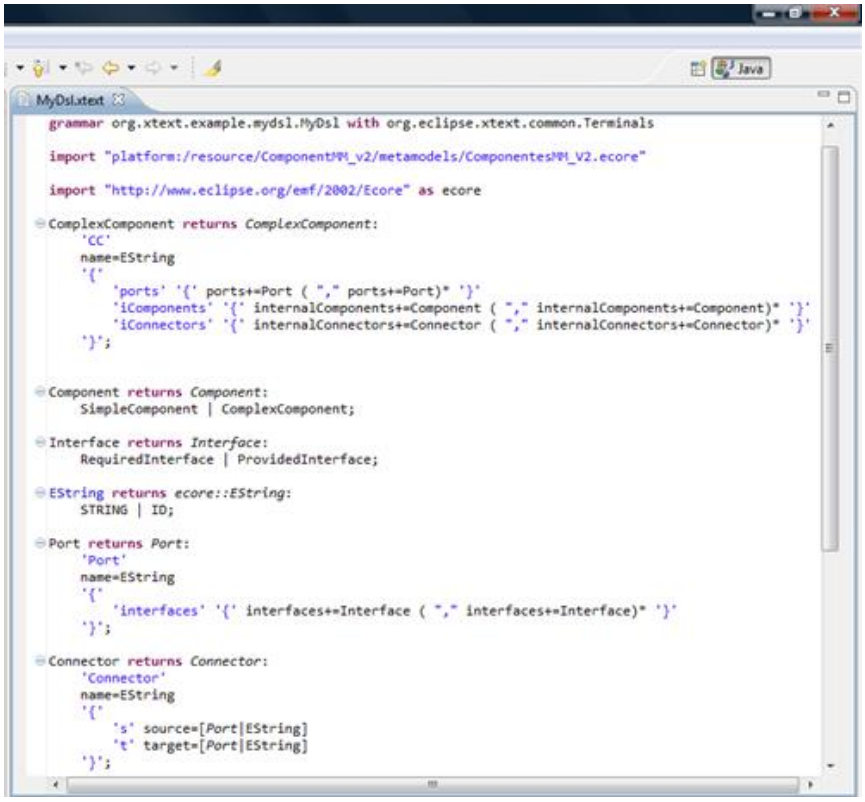

Figura 3.11: Fichero Xtext.

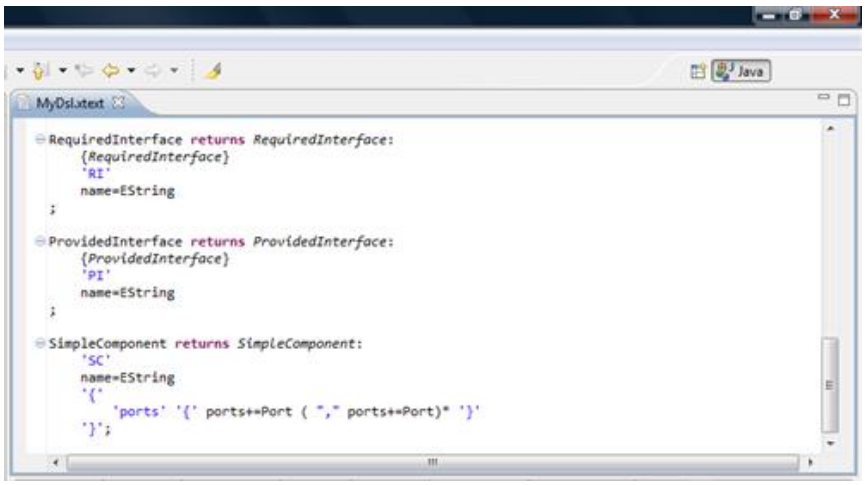

Figura 3.12: Fichero Xtext.

toda la sintaxis textual que tendrá nuestra herramienta. La segunda opción, es más cómoda, consiste en crear un nuevo proyecto Xtext a partir de un meta-modelo Ecore existente. La opción que ha sido utilizada en este proyecto ha sido la segunda.

1. **Creación del proyecto Xtext a partir de un meta-modelo Ecore**.

   Como se ve en la figura 3.13 el proceso comienza eligiendo la opción Xtext Project From Existing Ecore Models, y en el paso siguiente se nos pedirá que elijamos el meta-modelo, pero necesitaremos el archivo .genmodel del meta-modelo, no el .ecore.

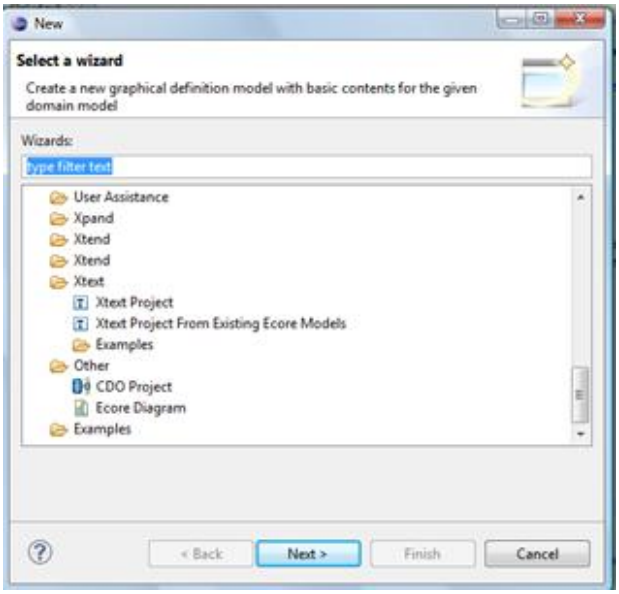

Figura 3.13: Creación proyecto Xtext.

   Tras seleccionar el .genmodel sólo queda elegir la extensión del nuevo lenguaje. De esta forma dispondremos del proyecto en el que desarrollar nuestro lenguaje textual.

2. **Generación de la herramienta textual**.

   En el proyecto encontraremos el archivo de nombre “extensión.xtext” (donde extensión será el nombre que se haya definido para nuestro lenguaje) en el cual se puede ver la definición por defecto que se ha generado del meta-modelo. Para poder generar el código necesario para arrancar una instancia de Eclipse y utilizar nuestro nuevo lenguaje, sólo tendremos que utilizar el archivo de extensión .mwe2 que se encuentra en el mismo lugar. Para ello tendremos que utilizar la función “Run as... MWE2 Workflow”, de esta forma se compilará el código de nuestro lenguaje y se generarán los archivos necesarios para arrancar el editor. En la figura 3.14 se puede ver un ejemplo en el que aparece la carpeta principal sobre la que se trabaja en el proyecto Xtext, donde aparece el archivo “compo.xtext” en el que se encuentra el código de desarrollo del lenguaje y el archivo “GenerateCompo.mwe2” con el que se puede compilar y generar los archivos como se ha explicado anteriormente.

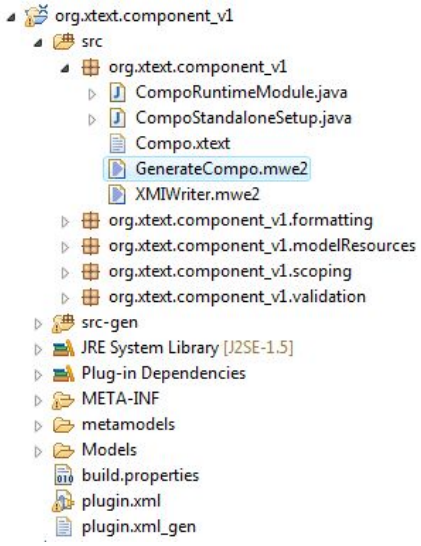

Figura 3.14: Imagen del proyecto Xtext.

La gramática que se utilizará en el lenguaje que se quiera crear, se podrá modificar en el "extensión.xtext". En este punto comienza la edición del lenguaje en el que el diseñador del editor tratará de definir como se deberán escribir los elementos del modelo. Entre otras cosas, añadir las funcionalidades que se consideren oportunas para facilitar la tarea del usuario que vaya a utilizar el editor. El aspecto que tiene finalmente el código Xtext del editor, es el de las figuras 3.11 y 3.12.

Capítulo 4

Planificación del Estudio para Evaluar Editores Gráficos vs Editores Textuales

En este capítulo se aborda la planificación del estudio empírico que permitiría evaluar los editores de modelos desarrollados en el capítulo anterior. Se introducirán las diferentes pruebas, sus objetivos, así como la forma en la que éstas se llevarán a cabo.

El objetivo principal del proyecto consiste en poder realizar una comparativa de las herramientas gráfica y textual vistas anteriormente, desde diferentes puntos de vista, para de esta forma, poder evaluar qué herramienta puede ser más apta para el trabajo a desarrollar dependiendo de la situación. Así, dado que la discusión sobre el uso de editores gráficos o textuales para un lenguaje de dominio específico es controvertida y suscita discusión, resulta interesante plantear un estudio que permita comparar el desempeño de ambos editores en aspectos como: la escalabilidad, flexibilidad ante cambios, etc.

Además, en este capítulo se introducen las características de los editores de modelos gráficos y textuales, de manera que los aspectos más relevantes queden destacados y puedan ser considerados en el diseño del estudio empírico.

Para realizar un mejor análisis de los puntos que se deben evaluar para realizar la comparativa entre la herramienta gráfica y la textual, se va a diferenciar entre dos roles distintos. El primer rol será el usuario final, es decir, la persona que finalmente trabajará con la herramienta y por lo tanto deberán tenerse en cuenta características como el nivel de dificultad que representa el aprender a usar la herramienta.

El otro rol que se evaluará, será el desarrollador. Este rol hace referencia a la persona que tiene que crear una herramienta. Las características que se deben tener en cuenta en la evaluación bajo el punto de vista de este rol estarán centradas en la dificultad de la propia creación de la herramienta y el mantenimiento de ésta.

En el proyecto no se utiliza una única técnica de estudio. Ya que para tratar el rol del usuario final se realizará un estudio mediante encuestas de las que se podrá obtener información de un grupo de usuarios que utilizarán los editores. En el rol del desarrollador se realiza un estudio semejante a los casos de estudio.

En las siguientes secciones de este capítulo se irán analizando las diferentes características de cada rol en detalle. El proceso de análisis de las características consistirá en definir en que

consiste dicha característica, definir de que modo se puede evaluar y posteriormente detallar los pasos que se llevarán a cabo en el experimento.

4.1. Usuario Final

En esta sección se describe el proceso de evaluación de las características enfocadas en el usuario final. Estos usuarios finales estarán representados por un grupo de sujetos que realizarán las pruebas con los editores. Estas pruebas se evaluarán mediante el uso de un cuestionario en el que los usuarios deberán realizar varios ejercicios y contestar a las preguntas que se realicen sobre el ejercicio. Lo ideal sería realizar dos grupos de diez a quince personas por grupo. Un grupo realizará las pruebas sobre el editor gráfico y el otro grupo sobre el editor textual. Cada grupo contará con el siguiente material y condiciones.

- Un ordenador con el sistema operativo Windows y la plataforma Eclipse y los editores.
- Una guía de usuario y ejercicios propuestos (se pueden encontrar en los anexos).
- Duración de la prueba de un tiempo aproximado de 2 horas.

En el caso de poder disponer de un grupo de usuarios que tengan experiencia previa en en CBSE (Component Based Software Engineering), es interesante realizar los ejercicios propuestos en los anexos, específicamente respecto a la escalabilidad y legibilidad del lenguaje ya que se podría realizar un estudio con ejercicios más técnicos y para obtener una información diferente con respecto a usuarios que están teniendo una primera toma de contacto con CBSE. Esto no está contemplado en la planificación del proyecto puesto que el interés del estudio está focalizado en partir desde un inicio en el que no se tienen conocimientos de las herramientas y obtener resultados respecto al proceso de aprendizaje y trabajo con los editores.

4.1.1. Curva de Aprendizaje

Esta prueba se realiza sobre un grupo de usuarios que no tengan ningún conocimiento previo de ninguna de las herramientas. Creando dos grupos, a cada uno se le asignaría una de las dos herramientas y se les proporcionaría una guía de usuario (se puede consultar en los anexos). Tendrán que realizar un ejercicio en el que se va guiando al usuario sobre como se puede construir un modelo. El usuario debe indicar la hora de inicio y fin del ejercicio (ya que cada usuario puede necesitar un tiempo distinto en realizar la prueba), de esta forma se puede evaluar el tiempo que el usuario emplea en realizar esa parte del problema. Además, se le pide al usuario valorar las dificultades que ha tenido en su realización. Posteriormente se plantea un ejercicio mas complejo para evaluar como responden los usuarios ante un incremento de la dificultad, valorando de nuevo el tiempo utilizado y las dificultades que hayan tenido lugar, el ejercicio sigue siendo guiado. Finalmente se añade un grado mas de complejidad, el usuario debe realizar una modificacion al ejercicio anterior, esta vez al usuario no se le guía sobre como resolverlo.

4.1.1.1. Preguntas/pruebas

Con respecto a las preguntas que queremos contestar con esta prueba, nos centramos en la dificultad que supone al usuario superar cada uno de los niveles que va superando a lo largo del ejercicio. Para ello se le realizarán varías preguntas con respecto al grado de dificultad en cada uno de los tres niveles del ejercicio, comentar qué dificultades específicamente ha tenido y entendibilidad del modelo una vez ha sido creado con el editor. Estas preguntas pueden encontrarse en los anexos A.2.1 y B.2.1 para los editores textual y gráfico respectivamente.

4.1.1.2. Proposiciones

Con esta prueba, se espera obtener una idea de cuán difícil es para el usuario afrontar retos de una dificultad que se va incrementando con los diferentes ejercicios. Además, al ser un proceso a varios niveles, se puede tener una idea de con qué editor resulta más fácil aprender a realizar tareas sencillas, con cuál es más fácil realizar ejercicios más completos y con cuál editor se resuelve más rápido el problema.

4.1.1.3. Relación lógica entre pruebas y proposiciones

Relacionando las preguntas que se proponen en el cuestionario con lo que se espera obtener de ellas para evaluar este apartado, se van a presentar las preguntas que aparecen en el anexo y cómo con ellas se pueden obtener resultados para posteriormente evaluar la información.

Comenzando por la dificultad que en sí puede suponer cada nivel del ejercicio, la pregunta «indique el grado de dificultad que ha supuesto su realización» formulada en cada uno de los niveles del ejercicio, se puede obtener la dificultad en el proceso de realización del ejercicio. La siguiente cuestion es una pregunta abierta que da libertad al usuario para comentar las dificultades específicas que el usuario ha encontrado durante la realización del ejercicio. Con esta cuestión se espera obtener una lista de características que para los usuarios resulta en dificultades o molestias al trabajar con el editor. Finalmente se pide al usuario que observando el resultado del ejercicio, valore qué dificultad supondría el explicar el modelo resultante a otra persona. Esta cuestión proporciona información sobre la percepción de cómo el usuario entiende el modelo que ha resultado y si para otra persona sería fácil de entender.

4.1.1.4. Criterios para interpretar los resultados

En la prímera y tercera cuestión, el usuario puede elegir de forma cerrada una opción sobre el grado de dificultad. Las respuestas van desde «Muy fácil», «Fácil», «Normal», «Difícil» hasta «Muy difícil». A la hora de analizar los resultados obtenidos del cuestionario, las respuestas se codifican utilizando una escala numérica de 1-5, siendo 1 el grado de más facilidad y 5 la máxima dificultad. Esta calificación se calcula como la media aritmética de los resultados obtenidos en las preguntas 1 y 3. En la segunda cuestión, al ser de carácter abierto, se reunirá la lista de ventajas y desventajas que los usuarios indiquen, dando especial importancia a las respuestas que se repitan en un mayor número.

4.1.2. Escalabilidad

Esta característica especifica la dificultad que supone el desarrollar y manejar modelos de un mayor tamaño con los editores.

4.1.2.1. Preguntas/pruebas

A diferencia de la prueba anterior, ésta consiste en disponer de diferentes ejemplos de modelos de componentes. Un ejemplo con una estructura básica de pocos componentes (4 componentes), otro ejemplo con una estructura de tamaño medio (13 componentes) y un último ejemplo con una estructura muy grande (27 componentes). Lo ideal sería desarrollar estos ejemplos con cada herramienta y obtener resultados en torno al tiempo necesario, dificultad del proceso conforme crece el tamaño de la estructura y la complejidad que pueda suponer entender la estructura una vez terminada con cada herramienta. Sin embargo, la evaluación de esta característica se realiza planteando a los usuarios dichos ejemplos, y dejando libertad a los usuarios para comentar como realizarían los ejemplos, describiendo las dificultades que los usuarios creen que se podrían encontrar al enfrentarse a modelos así.

Se decide realizar esta prueba de esta forma debido a que supondría un tiempo muy alto el realizar cada uno de los ejercicios por parte de los usuarios, especialmente los de mayor tamaño. Este sería un punto a poder desarrollarse entre las lineas futuras.

4.1.2.2. Proposiciones

De esta prueba se espera saber con qué editor resulta más fácil desarrollar los modelos sencillos y los más complejos. Además, obtener una lista de ventajas y desventajas que cada editor presenta en cada situación

4.1.2.3. Relación lógica entre pruebas y proposiciones

En el anexo A.2.2 y B.2.2 se pueden ver las cuestiones que se les presentan a los usuarios de las pruebas. En la primera, como ocurriera con la prueba de la *curva de aprendizaje*, el usuario deberá indicar en una pregunta cerrada el grado de dificultad que supone realizar un modelo del tamaño mostrado en cada uno de los niveles. En la segunda cuestión, al usuario se le pide que indique las ventajas y desventajas que se encontraría en el desarrollo de un modelo de ese tamaño con el editor. Por lo que se podrá obtener una lista de ventajas y desventajas para cada uno de los tres modelos que se presentan al usuario.

4.1.2.4. Criterios para interpretar los resultados

En la prímera cuestión, el usuario puede elegir de forma cerrada una opción sobre el grado de dificultad. Las respuestas van desde «Muy fácil», «Fácil», «Normal», «Difícil» hasta «Muy difícil». A la hora de analizar los resultados obtenidos del cuestionario, las respuestas se codifican utilizando una escala numérica de 1-5, siendo 1 el grado de más facilidad y 5 la máxima dificultad. Esta calificación se calcula como la media aritmética de los resultados obtenidos en

la pregunta 1. En la segunda cuestión, al ser de carácter abierto, se reunirá la lista de ventajas y desventajas que los usuarios indiquen, dando especial importancia a las respuestas que se repitan en un mayor número.

4.1.3. Legibilidad o Comprensibilidad de los Editores

Se trata de que el usuario describa en un lenguaje natural lo que ve en un modelo creado con el editor gráfico y el textual. Dicho con otras palabras, qué fácil es comprender un modelo que ha sido desarrollado con una herramienta u otra.

4.1.3.1. Preguntas/pruebas

Esta prueba consiste en presentar a los sujetos un modelo construido con cada uno de los editores. El sujeto deberá describir en lenguaje natural lo que entiende de ese modelo con la mayor precisión posible. El modelo propuesto para cada uno de los editores se pueden ver en los anexos A.2.3 y B.2.3 para el editor textual y gráfico respectivamente.

4.1.3.2. Proposiciones

La idea es saber con esta prueba que editor resulta más sencillo de comprender.

4.1.3.3. Relación lógica entre pruebas y proposiciones

Al usuario se le plantea una pregunta abierta en la que deberá definir con todo detalle el modelo que se le presenta. El mayor o menor acierto del usuario al describir el modelo proporcionará la información sobre qué editor resulta más claro de entender.

4.1.3.4. Criterios para interpretar los resultados

Al mostrarse un ejemplo que describe el mismo modelo realizado con dos editores distintos (cada grupo evaluará el de su editor), con los resultados obtenidos se podrá comparar que editor consigue que un usuario proporcione una descripción más acertada del modelo y por lo tanto pueda interpretarse que dicho editor resulta más legible

4.1.4. Impresión del Usuario

Con esta prueba se trata de recoger la impresión que se lleva el usuario tras la toma de contacto con la herramienta.

4.1.4.1. Preguntas/pruebas

Al usuario se le plantea que exprese su opinión sobre la facilidad del uso de los editores, no sólo general, sino de las características que ofrecen los editores, como podrían ser característi-

cas de autocompletar en Xtext, díalogos desplegables, etc. y qué comenten cualquier ventaja o desventaja que no hayan tenido cabida en las otras preguntas del cuestionario.

4.1.4.2. Proposiciones

Con esta prueba, se trata de cubrir algun aspecto que no haya sido tenido en cuenta en las pruebas anteriores. Así como mejora de los editores para futuras evaluaciones.

4.1.4.3. Relación lógica entre pruebas y proposiciones

Se plantean dos preguntas abiertas al usuario. En la primera el usuario tendrá que comentar las ventajas y desventajas que posee el editor, así como dificultades que presente. En la segunda, aquellas funcionalidades o características que resultarían útiles para la mejora del editor.

4.1.4.4. Criterios para interpretar los resultados

Los resultados obtenidos en esta prueba están orientados a la mejora de los editores para futuras pruebas así como mejora de los cuestionarios para futuras evaluaciones que se realizaran con otros grupos de usuarios. De esta forma se podrían incluir más cuestiones u opciones en las cuestiones actuales que no hayan sido tenidas en cuenta.

4.2. Desarrollador

En esta sección se muestra el proceso de evaluación de las características relacionadas con el rol del desarrollador de herramientas de DSDM. Durante el proceso de realización de los editores se realizarón ciertas medidas para evaluar entre otras cosas el coste de tiempo que requiere el aprendizaje de las herramientas GMF y Xtext, la consulta de la bibliografía disponible, etc. que permiten introducir un estudio desde la perspectiva del desarrollador de editores de modelos.

A continuación, se irán presentando los distintos puntos que se han evaluado en este apartado más teórico.

4.2.1. Tiempo de Desarrollo

El tiempo de desarrolllo es el tiempo que se ha destinado a poder desarrollar las herramientas. Esta característica a comparar, consiste en la medida del tiempo que se dedica a todo el proceso completo para obtener un editor. Este proceso parte desde la consulta de la documentación o bibliografía disponible sobre la herramienta textual y gráfica para comenzar el aprendizaje y manejar la herramienta hasta la finalización de las herramientas descritas en el capítulo 3.

4.2.1.1. Descripción de las prueba

La prueba consiste en llevar a cabo un control del tiempo dedicado en todo el desarrollo de las herramientas. Por lo tanto, inicialmente se comenzó con el aprendizaje de GMF, realizando

tutoriales disponibles en la página de referencia del proyecto Eclipse GMF y anotando el tiempo dedicado en cada sesión. Posteriormente se comenzó a desarrollar el editor con GMF de la primera versión del meta-modelo, anotando de la misma forma el tiempo dedicado. Una vez terminado el editor con la primera versión, se procedio a realizar las modificaciones para adaptar el editor a la segunda versión del meta-modelo. Tras tener el editor gráfico desarrollado, el siguiente punto fue comenzar a leer la documentación sobre Xtext y comenzar a aprender su manejo mediante los tutoriales disponibles. Al igual que en el caso del editor gráfico, una vez obtenido un conocimiento básico sobre Xtext, se procedió a implementar la primera versión del editor y posteriormente, la adaptación del editor a la segunda versión del meta-modelo. Además, se contabilizó el tiempo dedicado a mejorar el aspecto gráfico del editor o la facilidad de escritura en el caso del editor textual.

En la siguiente tabla, se puede ver la información del tiempo empleado (en número de horas) en las distintas fases que se han medido.

	GMF	Xtext
Aprendizaje	15	12.5
Desarrollo del editor	19	3*
Actualización a versión 2	3	1
Mejoras en editores	8	4

Cuadro 4.1: Tiempo empleado en las distintas fases del proyecto.

Comentar que el asterisco en la columna referente a Xtext con el tiempo de desarrollo del editor que hace referencia a un tiempo bastante corto, es debido a la generación de una sintaxis directamente a través del meta-modelo Ecore. Al tener una base hecha, el tiempo requerido para tener un editor es muy pequeño.

Es interesante añadir en este punto, que durante la realización del proyecto, los proyectos GMF y Xtext han estado en continua evolución. Esto es importante debido especialmente a un problema durante la implementacion del editor textual, en el que los conectores (que estan contenidos en un componente) no eran capaces de reconocer las interfaces de otros componentes distintos. Este problema se solucionó con posteriores actualizaciones de la herramienta Xtext, pero el tiempo dedicado a tratar de solucionar este error, sin resultados positivos, no ha sido tenida encuenta en esta comparativa, aunque es importante remarcar este dato.

Con la evolución de estos proyectos Eclipse, surgió la posibilidad de implementar las restricciones OCL directamente en el meta-modelo con el editor OCLinEcore. En un primer momento, durante el desarrollo del editor gráfico, las restricciones OCL fueron implementadas directamente en el editor, aunque posteriormente se decidió el uso de OCLinEcore, pues, al partir de la información del meta-modelo, no hay que preocuparse en implementar nada adicional en cada editor.

4.2.1.2. Expectativas

Lo interesante de esta prueba es obtener una medida del tiempo empleado en el desarrollo de editores, de esta forma se puede tener una idea sobre si en principio es más rápido realizar un editor con Xtext que con GMF. Además, saber en que fases del desarrollo del editor se puede requerir más tiempo.

4.2.2. Mantenibilidad

Esta característica evalúa la dificultad que entraña la modificación de los editores gráfico y textual, ante la incorporación de un cambio del meta-modelo, teniendo en cuenta el tiempo, facilidades o dificultades que se presenten ante posibles cambios.

4.2.2.1. Descripción de las prueba

Esta prueba se plantea en la introducción de un pequeño cambio en el meta-modelo de partida, que sería el cambio que se realizó de la primera a la segunda versión del meta-modelo del que se habló en el capítulo 3. Evaluar el tiempo que requirió realizar esta modificación en ambas herramientas, así como algún aspecto de dificultad que hubiera podido suponer dicho cambio en cada herramienta.

A continuación se procederá a describir la repercusión que tuvo la modificación de la primera versión (lo que dio luegar a la segunda versión) del meta-modelo en los editores que estaban adaptados para esta versión.

- **GMF**. En GMF, cuando se van a generar los diferentes archivos, se parte del fichero en el que se encuentra el meta-modelo (fichero .ecore), de esta forma, GMF proporciona una estructura básica sobre la que comenzar a modificar el editor. Al realizar una modificación en el meta-modelo, los posibles componentes que hay que añadir al editor se deben realizar manualmente. En el caso de la modificación que se realizó para obtener la segunda versión, en el fichero .gmfgraph, hay que añadir la figura que representará a los puertos en los que se añadirán las interfaces. En el fichero .gmftool habrá que añadir a la paleta de herramientas el elemento con el que añadir los puertos a los distintos componentes y .gmfmap añadir los elementos necesarios así como reestructurar el mapeo de elementos para la nueva configuración del meta-modelo.

- **Xtext**. En Xtext termina siendo un caso análogo al de GMF, en el que tras realizar modificaciones en el meta-modelo, se debe realizar los difernetes ajustes en Xtext de la forma habitual. Si el lenguaje en Xtext se creó generándolo directamente desde el meta-modelo, como se explicó en el capítulo 3, Xtext enseguida nos mostrará que hay ciertos errores en nuestro código (porque los nuevos elementos del meta-modelo no aparecen en el código). En cualquier caso, el proceso en Xtext es simple, ya que tras añadir los cambios al lenguaje, simplemente habrá que regenerar el archivo MWE2 Workflow y arrancar la instancia Eclipse con el editor para probar los cambios en el lenguaje.

4.2.2.2. Expectativas

Con esta prueba, el objetivo es descubrir como afectan los cambios de los meta-modelos en los editores. Por lo tanto, obtener información sobre qué dificultades supone realizar estos cambios en los editores, cuán laboriosa es esta tarea.

4.2.3. Extensibilidad

Esta característica mide las posibilidades que tiene cada herramienta de ser extendida con nuevas características o funcionalidades (por ejemplo, en Xtext: coloración de texto, configurar asistente de compleción, auto-compleción,)

No se dispone de una prueba en sí para evaluar esta característica, es más bien una comparativa informativa de las opciones que proporciona cada herramienta a la hora de añadir funcionalidades o ayudas al usuario.

GMF proporciona un gran editor gráfico pero el proceso de personalización del editor se puede adaptar sólo hasta cierto punto. Los editores tienen al final una apariencia similar y a menudo algunas características deseables no estan disponibles. Entre las diferentes opciones que permite GMF para extender la funcionalidad del editor gráfico, podemos encontrar las siguientes:

- Versionado. Modificando una serie de plantillas Xpand y utilizando EAnnotation en el diagrama, se puede añadir un mecanismo con el que indicar la versión del diagrama creado y posteriormente poder usarlo por ejemplo para migrar el modelo.

- Cambios en la apariencia gráfica. GMF sólo permite cambiar la apariencia del editor con opciones básicas como el color de las lineas de las figuras, el color de fondo o el tipo de fuente en las letras. Existe la posibilidad de extender el modelo gmfgen para añadir nuevas caracteristicas a los nodos del modelo, modificando las plantillas Xpand.

- Copiar-pegar. En GMF existe un problema cuando en el editor se intenta copiar algun elemento realizando el mecanismo de copiar y pegar. GMF realiza una copia por referencia, es decir, las figuras del diagrama se duplican pero ambas estarán asociadas al mismo objeto emf. Por este motivo, intentar realizar esta acción provoca comportamientos extraños. Es posible modifiar el comportamiento que tiene el comando copiar y pegar en GMF, de forma que no se produzca la asociacion al mismo elemento. Aunque el proceso es tedioso, es una muestra de la posibilidad de modificar o añadir nuevos comandos para mejorar o añadir opciones al editor.

- Edición mediante comandos. En GMF existe la posibilidad de crear comandos que permitan generar ciertas estructuras directamente. Tomando como ejemplo un modelo en el que los elementos se utilicen para crear diagramas de flujo. Se podrían crear comandos para comenzar el flujograma, para añadir estados, para añadir decisiones, etc. En la figura 4.1 se puede ver un ejemplo de como se iría creando el diagrama de flujo mediante el uso de estos comandos.

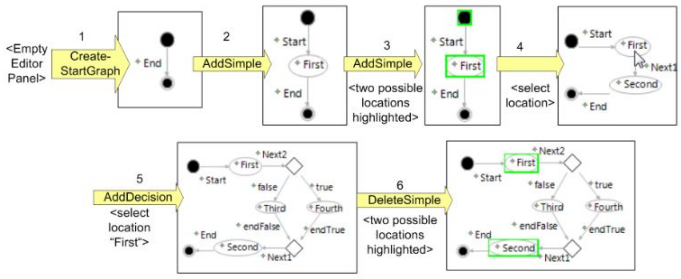

Figura 4.1: Diagrama de flujo mediante comandos en GMF.

XTEXT: Entre las diferentes características que Xtext ofrece para extender las opciones de la herramienta, se pueden encontrar las siguientes:

- **Coloración de sintáxis**. En Xtext es posible utilizar diferentes colores y fuentes de acuerdo al significado de las distintas partes del texto que se introduce. El uso de colores no tiene influencia en la semántica, pero ayuda a diferenciar las distintas partes, por ejemplo en la búsqueda de errores en el código. Algunos de los bloques a colorear, podrían ser identificadores, palabras clave y cadenas de caracteres. Hay dos modos de implementar la coloración de sintáxis en Xtext:
  - Resaltado léxico. La configuración mediante este modo se realiza mediante la implementación de una página de estilo que fijara la configuración de cada componente.
  - Resaltado semántico. Mediante este modo, el funcionamiento se ejecuta en segundo plano, calculando las partes a resaltar durante la escritura de los diferentes elementos basado en el significado de éstos en el modelo.
- **Autoinyección de texto**. Xtext ofrece la posibilidad de introducir automaticamente valores a las propiedades de un componente. Esto es muy útil con flujos de trabajo que son altamente configurables, con decenas de parámetros a rellenar por diferentes componentes.
- **Plantillas de opción múltiple**. Xtext ofrece una serie de plantillas predefinidads. Estas plantillas dan soporta a una resolución automatica de referencias. Suponiendo un ejemplo en el que se está trabajando con una serie de eventos y transiciones, estas plantillas permiten seleccionar un evento y mediante una lista desplegable, intercambiar entre los distintos tipos de evento.

4.2.3.1. Expectativas

Con la extensibilidad se pretende presentar información sobre las principales formas de mejorar los editores añadiendo características y funcionalidades a estos. De esta forma, se puede ver que herramienta permite una mayor posibilidad de personalización de los editores.

Capítulo 5

Conclusiones y Lineas de Trabajo Futuras

Este proyecto aborda la comparativa de herramientas textuales y gráficas para el modelado en el Desarrollo de Software Dirigido por Modelos (DSDM). Para resumir, el documento muestra el diseño de un estudio comparativo de dos tecnologías: GMF y Xtext. Comenzó con la introducción del DSDM en el capítulo *Lenguajes de Modelado y Técnicas para su Evaluación*, mostrando Eclipse como plataforma que soporta este paradigma y se introdujo además las dos tecnologías a comparar, GMF y Xtext, así como una introducción teórica sobre las técnicas de evaluación de lenguajes. Después, en el capítulo *Desarrollo de un Lenguaje de Modelado para Especificar Arquitecturas de Componentes* se mostró el meta-modelo sobre el que se construirían posteriormente los dos editores de modelado, describiendo además, el proceso de construcción de los editores. En el siguiente capítulo, *Planificación del Estudio para Evaluar Editores Gráficos vs Editores textua* se describieron los puntos sobre los que trataría la comparativa. Finalmente, en este capítulo se resumen las conclusiones y las líneas de trabajo futuras surgidas durante el desarrollo del presente proyecto.

5.1. Conclusiones

Las herramientas gráficas se pueden encontrar en múltiples aplicaciones, desde toolbox para diseño de protocolos (SDL), para diseñar sistemas de filtros (Simulink de Matlab), herramientas CAD para diseño de circuitos (PSpice), etc. Todas estas herramientas tienen en común que el entorno de desarrollo sea amigable y organizado. El uso de estas herramientas consiste normalmente en escoger elementos de una paleta e ir arrastrando estos elementos a un entorno de trabajo, y luego unir estos elementos. En este proceso, el desarrollo se realiza desde una perspectiva de alto nivel, en el que se suelen desarrollar unos bloques principales y posteriormente modificar algún parámetro de cada elemento.

El usuario que trabaja con estas herramientas no tiene porque conocer todos los detalles de cada bloque, puede tratarlos como cajas negras. Esto es realmente cómodo para realizar diseños sencillos, y poder identificar rápidamente el funcionamiento que puede tener el sistema completo con un simple vistazo, pero todo esto puede cambiar cuando el diseño es un sistema muy complejo, en el que se pueden tener decenas o centenares de elementos en pantalla. Para poder manejar

estas situaciones la herramienta debería ser capaz de tener ciertos mecanismos de encapsulado que permitiera al usuario visualizar solamente el área de interés.

A la hora de fijarnos en herramientas textuales, podemos recurrir a los muchos lenguajes de programación que se conocen, como Java, C, C++, PHP, etc. El uso de lenguajes de programación textuales, requiere que el usuario conozca en buena manera de todas las reglas sintácticas que tenga ese lenguaje, como pueden ser, la forma de declarar elementos, palabras clave, símbolos para declarar relaciones entre elementos, etc. Esto hace que para implementar soluciones muy sencillas, para el usuario pueda ser algo más tedioso el implementar el código que arrastrar un par de elementos y unirlos con una flecha, como se podría realizar con una herramienta gráfica. Cuando se dispone de un sistema complejo como se comentó anteriormente de decentas o centenares de elementos, puede resultar mucho más cómodo desarrollarlo mediante líneas de código, pudiendo realizar módulos que resuelvan pequeñas partes del problema.

Como se comentó anteriormente, en este proyecto no se realizó la evaluación de los editores con ningún grupo de usuarios, por lo que no se pueden obtener resultados concretos sobre los cuestionarios y las pruebas, pero sí se pueden analizar algunos puntos importantes obtenidos durante el desarrollo del proyecto.

- **Tiempo de Desarrollo**.

  En referencia a la tabla 4.1 del tiempo de desarrollo mostrada en el capítulo anterior, se puede ver que el tiempo de aprendizaje con una u otra herramienta no es muy diferente. La mejor forma de aprender y conocer cómo funcionan las herramientas es realizando los difernetes tutoriales disponibles. Es interesante comentar que Xtext dispone de una documentación muy completa de la herramienta. Como suele suceder con proyectos que están en continua evolución, a menudo muchos de los tutoriales disponibles para aprender son de versiones antiguas por lo que puede ser frustrante. Respecto al tiempo de desarrollo de los editores, hay una diferencia bastante grande entre GMF y Xtext. La funcionalidad de Xtext para comenzar un proyecto a partir de un meta-modelo EMF previamente creado es de mucha ayuda, ya que te proporciona una buena base, que realmente sólo necesita realizar las mejoras oportunas para que el editor sea cómodo para el usuario. Por otro lado, GMF requiere de un proceso algo más laborioso, en el que es bueno ir despacio para no cometer errores, además de que a menudo resulta complicado configurar correctamente el apartado gráfico de los componentes. Sobre la actualización de los editores a la segunda versión del meta-modelo, en GMF resulta algo más engorroso, por lo comentado previamente. Finalmente, las mejoras de los editores son también algo mas complejas de realizar en GMF. En Xtext se puede encontrar en la documentación como realizar varias mejoras para el editor. Con GMF, incluso retoques en los componentes, para añadir mejoras gráficas o más detalles, resulta complicado.

- **Mantenibilidad**.

  Como se pudo ver en el capítulo anterior, los cambios realizados en la primera versión del meta-modelo obligaban a incluir los nuevos elementos en los editores. En GMF el proceso es más lento que en Xtext, ya que hay que incluir los elementos gráficos que representan los elementos nuevos del meta-modelo, posteriormente añadir la opción en

la paleta de herramientas y más tarde realizar el mapeado del elemento gráfico con su equivalente del meta-modelo. En el caso particular de este proyecto, había que modificar además las referencias de los conectores, ya que en la segunda versión del meta-modelo, se unen puertos, no interfaces. En Xtext, cómo se trabaja en el código directamente con la implementación del meta-modelo, los cambios son mucho más directos y sencillos, basta con añadir el código de los nuevos elementos y generar el código del editor.

Se puede por lo tanto concluir que Xtext resulta más sencillo a la hora de realizar modificaciones en el meta-modelo que GMF.

- **Extensibilidad**.

  Del capítulo anterior se puede extraer que GMF proporciona un editor gráfico básico que sirve como una buena base para a partir de el poder trabajar añadiendo nuevas opciones. Estas opciones a menudo son bastante complejas y requieren el modificar clases específicas de la API. Es por lo tanto un trabajo complejo que requiere de bastante dedicación aunque hay diferentes proyectos que proporcionan información sobre como añadir estas extensiones a los editores GMF. En Xtext la mayoria de opciones comentadas en el capítulo anterior se encuentran explicadas en la documentación del proyecto. Como se pudo ver, se puede facilitar el uso de la herramienta textual enriqueciendo el código con diferentes colores, fuentes y añadiendo menús desplegables cuando hay que elegir entre diferentes opciones, como puede ser usando de ejemplo el meta-modelo de este proyecto, elegir entre las difernetes interfaces disponibles para unir por un conector.

  Por facilidad, en Xtext resulta mucho más sencillo extender las características del editor que en GMF. A pesar de esto, si se compara Xtext con GMF, quizás el funcionamiento básico del editor gráfico puede ser más sencillo que el textual, ya que el editor textual puede resultar tedioso si no se realizan mejoras incluso para ejemplos sencillos.

El trabajo realizado en este proyecto muestra el proceso de aprender a utilizar dos herramientas distintas, desarrollar los editores que describirán el mismo tipo de modelos y del trabajo realizado, extraer las características y peculiaridades de cada editor para establecer un mecanismo de análisis para la comparativa de los editores. Los puntos más interesantes a destacar sobre el proyecto realizado son:

- Queda pendiente realizar las pruebas con los grupos de usuarios para poder obtener resultados de las características relacionadas con el usuario final.
- El proyecto sirve como punto de partida para todo aquel que vaya a comenzar un proyecto en el que se plantee si utilizar un editor textual o gráfico para ver cuál se ajustaría a sus condiciones.
- La comparativa acerca al usuario las formas distintas de trabajo del DSDM, manejando modelos de forma gráfica y textual.
- La dificultad del aprendizaje en estas herramientas se debe a menudo a la falta de documentación adecuada y tutoriales desfasados. No obstante, Xtext dispone de una documentación bastante completa, que facilita la toma de contacto con la herramienta.

- Por lo que se ha visto, Xtext resulta una herramienta más sencilla de manejar, que permite en poco tiempo disponer de un editor en funcionamiento. Por otro lado, GMF proporciona un editor textual, que en primera instancia resulta más sencillo o atractivo para el usuario.

5.2. Lineas de Trabajo Futuras

Este proyecto sirve como punto de entrada para profundizar en las comparativas de herramientas del DSDM. El primer paso sería por supuesto, realizar las encuestas diseñadas en este proyecto para obtener unos resultados más profundos en la comparativa. Además, un interesante punto de estudio, sería analizar los modelos creados con ambos editores y comprobar la compatibilidad de los modelos resultantes. Esto se podría realizar con la herramienta EMFCompare por ejemplo. Otra linea de trabajo sería el añadir a la comparativa más herramientas gráficas y textuales, de forma que se puedan obtener además, resultados comparativos entre herramientas gráficas o entre herramientas textuales, además de servir de escaparate para todo aquel que se quiera iniciar en el DSDM, dando a conocer más posibilidades.

Apéndice A

Editor Textual. Guía de Usuario y Ejercicios

A.1. Guía de Usuario

En esta guía se va a explicar como manejar el editor textual de modelos. Antes de explicar como utilizar el editor es necesario conocer el tipo de modelos que se pueden construir utilizando el editor.

El editor permite construir modelos basados en componentes. Cada componente tendrá un número determinado de puertos. Se pueden interpretar los puertos como servicios que el componente va a proveer o requerir. Esto se indica mediante la inclusión de interfaces en los puertos. Por lo tanto, un puerto contendrá interfaces de tipo 'Required' o 'Provided'. Los componentes se unirán entre ellos por medio de conectores. La relación entre componentes será posible dependiendo de si se cumples ciertas condiciones con los puertos de los componentes a unir. Las condiciones son las siguientes:

- Los puertos cuyos componentes se encuentran en distinto nivel, para unirse deben tener interfaces con el mismo nombre y éstas deben ser del mismo tipo (por ejemplo, 'provided' con 'provided').
- Los puertos cuyos componentes se encuentran en un mismo nivel, para unirse deben tener interfaces con el mismo nombre y éstas deben ser de distinto tipo.

Existen además, dos tipos de componentes, el componente simple, que tendrá los puertos de los que se ha hablado anteriormente y el componente compuesto. El componente compuesto podrá albergar en su interior a otros componentes (simples y/o compuestos) y a los conectores.

Además de las condiciones anteriores sobre nombres y tipo de interfaces, hay otras reglas a tener en cuenta a la hora de construir los modelos.

- Todos los componentes deben tener nombres distintos.
- Las interfaces contenidas dentro de un mismo puerto, deben tener nombres distintos.

- Cada componente debe tener al menos un puerto definido.
- Cada componente compuesto debe tener en su interior al menos dos componentes y dos conectores.

Por último, indicar que es obligatorio añadir un componente compuesto, que será el contenedor del resto de componentes del modelo.

Conociendo estas reglas básicas, es hora de comenzar a saber como construir nuestros modelos.

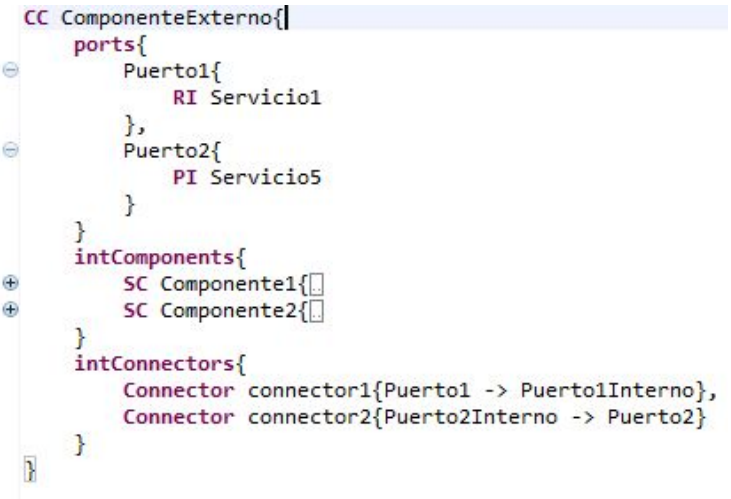

Figura A.1: Diagrama de flujo mediante comandos en GMF.

En la figura A.1 se muestra un modelo realizado con el editor. En él se puede ver las palabras clave para definir cada una de las partes del modelo. Para introducir un componente compuesto en el modelo hay que introducir:

```
1
2 CC nombre_componente{
3
4 }
```

A continuación, se debe introducir el puerto o puertos del componente compuesto que acabamos de definir, por lo tanto, en el interior de los corchetes bastará con introducir la palabra clave ports como se muestra a continuación:

```
1 ports{
2     nombre_puerto{
3       RI nombre_interfaz
4     }
5 }
```

Tras introducir ports y abrir corchetes, se debe introducir el nombre del puerto y en el interior definir las distintas interfaces. Como se muestra a continuación las interfaces se define introduciendo 'RI' o 'PI' para indicar que la interfaz es 'required' o 'provided' respectivamente. Cuando se introducen más de una interfaz en un puerto, éstas se separan con comas, como se puede ver a continuación

```
ports{
    nombre_puerto{
      RI nombre_interfaz
    },
    nombre_puerto{
      PI nombre_interfaz,
      RI nombre_interfaz
    }
}
```

El siguiente paso es introducir los componentes simples o compuestos que tendrá el componente compuesto. En este caso, habrá que introducir la palabra clave 'intComponents' y a continuación definir los componentes seguido de los puertos que tenga cada componente como se ha explicado previamente.

```

  intComponents{
    SC nombre_componente{
      ports{
        nombre_puerto{
          RI nombre_interfaz
        }
      }
    },
    SC nombre_componente{
      ports{
        nombre_puerto{
          PI nombre_interfaz
        }
      }
    }
  }
```

El último paso para completar al componente compuesto sería introducir los conectores (recordemos que un componente compuesto requiere de al menos dos conectores). Para introducir los conectores es necesario introducir la palabra clave 'intConnectors'. En el interior se deben definir los distintos conectores mediante la palabra clave 'Connector'. A continuación se puede ver un ejemplo.

```

  intConnectors{
    Connector nombre_conector{nombre_puerto -> nombre_puerto},
    Connector nombre_conector{nombre_puerto -> nombre_puerto}
  }
```

La conexión de los puertos mediante el conector se realiza introduciendo el nombre del puerto seguido del '->' y el nombre del otro puerto. Como en los casos anteriores, hay que separar con comas los distintos conectores que se introduzcan.

A.2. Ejercicios

A continuación se presentan los ejercicios que los usuarios tendrán que realizar para realizar la evaluación del editor textual. Entre los ejercicios a realizar se evaluarán:

- Curva de Aprendizaje.
- Escalabilidad.
- Legibilidad o Comprensibilidad del lenguaje.
- Impresión del Usuario.

A.2.1. Curva de Aprendizaje

1. **Ejercicio 1**.

   El primer ejercicio consiste en un componente compuesto de nombre «compuestoA» que contendrá en su interior dos componentes simples: «simpleA» y «simpleB». El componente 'simpleA' tendrá dos puertos en su interior «SA1» y «SA2», el puerto «A1» con una interfaz de tipo «provided» y nombre «I1» y el puerto «A2» con una interfaz de tipo «required» y nombre «I2». El componente «simpleB» tendrá un único puerto «SB1» con una interfaz de tipo «provided» y nombre «I2». El componente «compuestoA» tendrá también un puerto «CA1» con una interfaz de tipo «provided» y nombre «I1». El componente «compuestoA» tendrá dos conectores. El conector «conectorCA1» que unirá el puerto «SB1» con el «SA2» y el conector «conectorCA2» unirá el puerto «SA1» con el «CA1».

   No olvide rellenar el cuestionario, indicando el tiempo de comienzo y finalización del ejercicio 1.

2. **Ejercicio 2.**

   En el segundo ejercicio, vamos a comenzar añadiendo algunas modificaciones a lo hecho anteriormente y añadir más componentes. Comenzaremos añadiendo un componente simple más al componente compuesto «compuestoA» cuyo nombre será «simpleC» y tendrá un puerto «SC1» con una interfaz de tipo «provided» llamada «I3». A continuación añadiremos un puerto al componente «simpleA», que llamaremos «SA3» y que tendrá una interfaz de tipo «required». Añadiremos un conector al componente «compuestoA» con el que uniremos los puertos «SC1» con el «SA3». El componente «compuestoA» tendrá un nuevo puerto «CA2» con una interfaz de tipo «required» llamada «I4». Añadiremos al modelo un nuevo componente simple «simpleD» con un puerto «SD1» que tendrá una interfaz de tipo «provided» llamada «I4». Se creará un conector «conectorE1» que unirá los puertos «SD1» con «CA2». Además, añadiremos otro nuevo componente compuesto «compuestoB» que tendrá su interior dos componentes simples, «simpleE» y «simpleF» con un puerto cada uno, «SE1» y «SF1» respectivamente, y cada puerto una interfaz de tipo «required», «I5» e «I6» respectivamente. «compuestoB» tres puertos: «CB1», «CB2» y «CB3». Cada uno de los puertos tendrá una única interfaz de tipo «required» y nombres «I5», «I6» e «I1», además, tendrá dos conectores, «conectorCB1» que unirá los puertos «CB1» con «SE1» y «CB2» con «SF1». Añadiremos otro conector «conectorE2» que unirá el puerto «CA1» con el puerto «CB3».

   No olvide rellenar el cuestionario, indicando el tiempo de comienzo y finalización del ejercicio 2.

3. **Ejercicio 3.**

   En el tercer y último ejercicio, se quiere encapsular el componente «simpleD» y «compuestoB» en un componente compuesto que llamaremos «compuestoC». Para ello, con la metodología que se ha seguido en los dos ejercicios anteriores, se deberán añadir los puertos pertinentes al componente compuesto nuevo de forma que las interfaces de los componentes internos queden conectados con el componente «compuestoA» de la misma forma que estaban en el ejercicio 2.

   No olvide rellenar el cuestionario, indicando el tiempo de comienzo y finalización del ejercicio 3.

1. **Respecto al primer ejercicio, indique el grado de dificultad que ha supuesto su realización.**

   Anote la hora de inicio del ejercicio:.....

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

   Anote la hora de finalización del ejercicio:.....

2. **Respecto al primer ejercicio, comente las dificultades que ha encontrado durante su realización.**

3. **Respecto al primer ejercicio, una vez finalizado, observando el resultado obtenido, si tuviera que explicar el modelo mostrándoselo a otra persona, valore qué grado de dificultad tendría dicha acción.**

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

4. **Respecto al segundo ejercicio, indique el grado de dificultad que ha supuesto su realización.**

   Anote la hora de inicio del ejercicio:.....

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

   Anote la hora de finalización del ejercicio:.....

5. **Respecto al segundo ejercicio, comente las dificultades que ha encontrado durante su realización.**

6. **Respecto al segundo ejercicio, una vez finalizado, observando el resultado obtenido, si tuviera que explicar el modelo mostrándoselo a otra persona, valore qué grado de dificultad tendría dicha acción.**

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

7. **Respecto al tercer ejercicio, indique el grado de dificultad que ha supuesto su realización.**

   Anote la hora de inicio del ejercicio:.....

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

   Anote la hora de finalización del ejercicio:.....

8. **Respecto al tercer ejercicio, comente las dificultades que ha encontrado durante su realización.**

9. **Respecto al tercer ejercicio, una vez finalizado, observando el resultado obtenido, si tuviera que explicar el modelo mostrándoselo a otra persona, valore qué grado de dificultad tendría dicha acción.**

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

A.2.2. Escalabilidad

Partiendo de la experiencia previa obtenida con los ejercicios anteriores con el editor y observando los modelos que se presentan a continuación:

```
CC CA{
  ports{
    CA1{
      RI I1,
      PI I2
    }
  }
  intComponents{
    SC SA{
      ports{
        SA1{
          RI I1,
          PI I2
        },
        SA2{
          RI I3
        }

      }
    },
    SC SB{
      ports{
        SB1{
          PI I3
        }
      }
    }
  }
  intConnectors{
    Connector conector1{SB1 -> SA2},
    Connector conector2{SA1 -> CA1}
  }
}
```

```
CC CA{
  ports{
    CA1{
      RI final
    }
  }
  intComponents{
    CC CB{
      ports{
        CB1{
          RI I1,
          PI I2
        },
        CB2{
          PI I10
        },
        CB3{
          PI I11
        },
        CB4{
          RI I8
        }

      }
      intComponents{
        SC SA{
          ports{
            SA1{
              RI I1,
              PI I2
            },
            SA2{
              RI I3
            }

          }
        },
        SC SB{
          ports{
            SB1{
              PI I3
            }
          }
        },
        SC SC1{
          ports{
            SC1{
              RI I4
            }
          }
        }
      }
      intConnectors{
        Connector conector1{SB1 -> SA2},
        Connector conector2{SA1 -> CA1},
        Connector conector3{CB4 -> SC1}

      }
    },
    SC SD1{
      ports{
        SD1{
          PI I5,
          RI I6
```

```
65          }
66        }
67      },
68      SC SE1{
69        ports{
70          SE1{
71            PI I5,
72            RI I6
73          },
74          SE2{
75            PI I7
76          }
77        }
78      },
79      SC SF1{
80        ports{
81          SF1{
82            RI I7
83          },
84          SF2{
85            PI I8
86          }
87        }
88      },
89      CC CC1{
90        ports{
91          CC1{
92            PI I9
93          },
94          CC2{
95            RI I10
96          },
97          CC3{
98            RI I11
99          }
100
101       }
102       intComponents{
103         SC SG{
104           ports{
105             SG1{
106               RI I9
107             },
108             SG2{
109               PI I12
110             }
111
112           }
113         },
114         CC CD1{
115           ports{
116             CD1{
117               RI I12
118             },
119             CD2{
120               RI I13
121             },
122             CD3{
123               RI I14
124             }
125
126           }
127           intComponents{
128             SC SH{
129               ports{
```

```
130                 SH1{
131                   RI I13
132                 }
133
134               }
135             },
136             SC SI{
137               ports{
138                 SI1{
139                   RI I14
140                 }
141
142               }
143             }
144
145           }
146           intConnectors{
147             Connector conector11{CD2 -> SH1},
148             Connector conector12{CD3 -> SI1}
149
150           }
151         }
152       }
153       intConnectors{
154         Connector conector9{CC1 -> SG1},
155         Connector conector10{SG2 -> CD1}
156
157       }
158     }
159
160   }
161   intConnectors{
162     Connector conector4{CB2 -> CC2},
163     Connector conector5{CB3 -> CC3},
164     Connector conector6{SD1 -> SE1},
165     Connector conector7{SE2 -> SF1},
166     Connector conector8{SF2 -> CB4}
167
168   }
169 }
```

```
CC CA{
  ports{
    CA1{
      RI final
    }
  }
  intComponents{
    CC CB{
      ports{
        CB1{
          RI I1,
          PI I2
        },
        CB2{
          PI I10
        },
        CB3{
          PI I11
        },
        CB4{
          RI I8
        }

      }
      intComponents{
        SC SA{
          ports{
            SA1{
              RI I1,
              PI I2
            },
            SA2{
              RI I3
            }

          }
        },
        SC SB{
          ports{
            SB1{
              PI I3
            }
          }
        },
        SC SC1{
          ports{
            SC1{
              RI I4
            }
          }
        }
      }
      intConnectors{
        Connector conector1{SB1 -> SA2},
        Connector conector2{SA1 -> CA1},
        Connector conector3{CB4 -> SC1}

      }
    },
    SC SD1{
      ports{
        SD1{
          PI I5,
          RI I6
```

```
65          }
66        }
67      },
68      SC SE1{
69        ports{
70          SE1{
71            PI I5,
72            RI I6
73          },
74          SE2{
75            PI I7
76          }
77        }
78      },
79      SC SF1{
80        ports{
81          SF1{
82            RI I7
83          },
84          SF2{
85            PI I8
86          }
87        }
88      },
89      CC CC1{
90        ports{
91          CC1{
92            PI I9
93          },
94          CC2{
95            RI I10
96          },
97          CC3{
98            RI I11
99          },
100         CC4{
101           PI I16
102         }
103
104       }
105       intComponents{
106         SC SG{
107           ports{
108             SG1{
109               RI I9
110             },
111             SG2{
112               PI I12
113             }
114
115           }
116         },
117         CC CD1{
118           ports{
119             CD1{
120               RI I12
121             },
122             CD2{
123               RI I13
124             },
125             CD3{
126               RI I14
127             }
128
129           }
```

```
130           intComponents{
131             SC SH{
132               ports{
133                 SH1{
134                   RI I13
135                 }
136
137               }
138             },
139             SC SI{
140               ports{
141                 SI1{
142                   RI I14
143                 }
144
145               }
146             }
147
148           }
149           intConnectors{
150             Connector conector11{CD2 -> SH1},
151             Connector conector12{CD3 -> SI1}
152
153           }
154         }
155       }, //BLOQUE 2
156       CC CE1{
157       ports{
158         CE1{
159           PI I9
160         },
161         CE2{
162           RI I10
163         },
164         CE3{
165           RI I11
166         },
167         CE4{
168           PI I17
169         }
170
171       }
172       intComponents{
173         SC SG{
174           ports{
175             SG1{
176               RI I9
177             },
178             SG2{
179               PI I12
180             }
181
182           }
183         },
184         CC CD1{
185           ports{
186             CD1{
187               RI I12
188             },
189             CD2{
190               RI I13
191             },
192             CD3{
193               RI I14
194             }
```

```
195
196           }
197           intComponents{
198             SC SH{
199               ports{
200                 SH1{
201                   RI I13
202                 }
203
204               }
205             },
206             SC SI{
207               ports{
208                 SI1{
209                   RI I14
210                 }
211
212               }
213             }
214
215           }
216           intConnectors{
217             Connector conector11{CD2 -> SH1},
218             Connector conector12{CE3 -> SI1}
219
220           }
221         }
222       }, //bloque 3
223       CC CF1{
224       ports{
225         CF1{
226           PI I9
227         },
228         CF2{
229           RI I10
230         },
231         CF3{
232           RI I11
233         },
234         CF4{
235           PI I18
236         }
237
238       }
239       intComponents{
240         SC SG{
241           ports{
242             SG1{
243               RI I9
244             },
245             SG2{
246               PI I12
247             }
248
249           }
250         },
251         CC CD1{
252           ports{
253             CD1{
254               RI I12
255             },
256             CD2{
257               RI I13
258             },
259             CD3{
```

```
260             RI I14
261           }
262
263         }
264         intComponents{
265           SC SH{
266             ports{
267               SH1{
268                 RI I13
269               }
270
271             }
272           },
273           SC SI{
274             ports{
275               SI1{
276                 RI I14
277               }
278
279             }
280           }
281
282         }
283         intConnectors{
284           Connector conector11{CD2 -> SH1},
285           Connector conector12{CF3 -> SI1}
286
287         }
288       }
289     },
290     SC SJ{
291       ports{
292         SJ1{
293           RI I16,
294         },
295         SJ2{
296           RI I17
297         },
298         SJ3{
299           RI I18
300         }
301
302       }
303     },
304     CC CG{
305       ports{
306         CG1{
307           RI I1,
308           PI I2
309         },
310         CG2{
311           PI I10
312         },
313         CG3{
314           PI I11
315         }
316
317       }
318       intComponents{
319         SC SA{
320           ports{
321             SA1{
322               RI I1,
323               PI I2
324             },
```

```
325                SA2{
326                  RI I3
327                }
328
329              }
330            },
331            SC SB{
332              ports{
333                SB1{
334                  PI I3
335                }
336              }
337            }
338          }
339          intConnectors{
340            Connector conector1{SB1 -> SA2},
341            Connector conector2{SA1 -> CA1}
342
343          }
344        }
345
346        intConnectors{
347          Connector conector9{CC1 -> SG1},
348          Connector conector10{SG2 -> CD1}
349
350        }
351      }
352
353    }
354    intConnectors{
355      Connector conector4{CB2 -> CC2},
356      Connector conector5{CB3 -> CC3},
357      Connector conector6{SD1 -> SE1},
358      Connector conector7{SE2 -> SF1},
359      Connector conector8{SF2 -> CB4},
360      Connector conector6{CC4 -> SJ1},
361      Connector conector7{CE4 -> SJ2},
362      Connector conector8{CF4 -> SJ3},
363
364      Connector conector7{CG2 -> SE3},
365      Connector conector8{CG3 -> SF3}
366
367    }
368  }
```

1. **Observando el modelo número uno, indique el grado de dificultad que cree que supondría implementar ese modelo con el editor.**

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

2. **¿Qué ventajas o desventajas cree que tendría el realizar el modelo con un editor de este tipo?**

3. **Observando el modelo número dos, indique el grado de dificultad que cree que supondría implementar ese modelo con el editor.**

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

4. **¿Qué ventajas o desventajas cree que tendría el realizar el modelo con un editor de este tipo?**

5. **Observando el modelo número tres, indique el grado de dificultad que cree que supondría implementar ese modelo con el editor.**

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

6. **¿Qué ventajas o desventajas cree que tendría el realizar el modelo con un editor de este tipo?**

A.2.3. Legibilidad o Comprensibilidad de los Editores

La siguiente prueba consiste en definir con la mayor claridad posible, el modelo que se presenta a continuación. Es necesario añadir todos los detalles que se entiendan del modelo.

```
CC CA{
  ports{
    CA1{
      RI final
    }
  }
  intComponents{
    CC CB{
      ports{
        CB1{
          RI I1,
          PI I2
        },
        CB2{
          PI I10
        },
        CB3{
          PI I11
        },
        CB4{
          RI I8
        }

      }
      intComponents{
        SC SA{
          ports{
            SA1{
              RI I1,
              PI I2
            },
            SA2{
              RI I3
            }

          }
        },
        SC SB{
          ports{
            SB1{
              PI I3
            }
          }
        },
        SC SC1{
          ports{
            SC1{
              RI I4
            }
          }
        }
      }
      intConnectors{
        Connector conector1{SB1 -> SA2},
        Connector conector2{SA1 -> CA1},
        Connector conector3{CB4 -> SC1}

      }
    },
    SC SD1{
      ports{
        SD1{
          PI I5,
          RI I6
```

```
65          }
66        }
67      },
68      SC SE1{
69        ports{
70          SE1{
71            PI I5,
72            RI I6
73          },
74          SE2{
75            PI I7
76          }
77        }
78      },
79      SC SF1{
80        ports{
81          SF1{
82            RI I7
83          },
84          SF2{
85            PI I8
86          }
87        }
88      },
89      CC CC1{
90        ports{
91          CC1{
92            PI I9
93          },
94          CC2{
95            RI I10
96          },
97          CC3{
98            RI I11
99          }
100
101       }
102       intComponents{
103         SC SG{
104           ports{
105             SG1{
106               RI I9
107             },
108             SG2{
109               PI I12
110             }
111
112           }
113         },
114         CC CD1{
115           ports{
116             CD1{
117               RI I12
118             },
119             CD2{
120               RI I13
121             },
122             CD3{
123               RI I14
124             }
125
126           }
127           intComponents{
128             SC SH{
129               ports{
```

```
130                 SH1{
131                   RI I13
132                 }
133
134               }
135             },
136             SC SI{
137               ports{
138                 SI1{
139                   RI I14
140                 }
141
142               }
143             }
144
145           }
146           intConnectors{
147             Connector conector11{CD2 -> SH1},
148             Connector conector12{CD3 -> SI1}
149
150           }
151         }
152       }
153       intConnectors{
154         Connector conector9{CC1 -> SG1},
155         Connector conector10{SG2 -> CD1}
156
157       }
158     }
159
160   }
161   intConnectors{
162     Connector conector4{CB2 -> CC2},
163     Connector conector5{CB3 -> CC3},
164     Connector conector6{SD1 -> SE1},
165     Connector conector7{SE2 -> SF1},
166     Connector conector8{SF2 -> CB4}
167
168   }
169 }
```

A.2.4. Impresión del Usuario

- **Tras haber realizado los diferentes ejercicios propuestos, indique a continuación cualquier observación que no haya comentado en los ejercicios anteriores. Se pide entre otras cosas que comente ventajas y desventajas que cree que tiene el editor. Indique que cosas le han gustado o le han resultado comodas a la hora de trabajar con el editor y también aquellas que presenten dificultades/incomodidades.**

- **Para afrontar las posibles incomodidades o dificultades que presente el editor, indique qué características haría que el editor resultara más sencillo de utilizar.**

Apéndice B

Editor Gráfico. Guía de Usuario y Ejercicios

B.1. Guía de Usuario

En esta guía se va a explicar como manejar el editor gráfico de modelos. Antes de explicar como utilizar el editor es necesario conocer el tipo de modelos que se pueden construir utilizando el editor.

El editor permite construir modelos basados en componentes. Cada componente tendrá un número determinado de puertos. Se pueden interpretar los puertos como servicios que el componente va a proveer o requerir. Esto se indica mediante la inclusión de interfaces en los puertos. Por lo tanto, un puerto contendrá interfaces de tipo 'Required' o 'Provided'. Los componentes se unirán entre ellos por medio de conectores. La relación entre componentes será posible dependiendo de si se cumples ciertas condiciones con los puertos de los componentes a unir. Las condiciones son las siguientes:

- Los puertos cuyos componentes se encuentran en distinto nivel, para unirse deben tener interfaces con el mismo nombre y éstas deben ser del mismo tipo (por ejemplo, 'provided' con 'provided').
- Los puertos cuyos componentes se encuentran en un mismo nivel, para unirse deben tener interfaces con el mismo nombre y éstas deben ser de distinto tipo.

Existen además, dos tipos de componentes, el componente simple, que tendrá los puertos de los que se ha hablado anteriormente y el componente compuesto. El componente compuesto podrá albergar en su interior a otros componentes (simples y/o compuestos) y a los conectores.

Además de las condiciones anteriores sobre nombres y tipo de interfaces, hay otras reglas a tener en cuenta a la hora de construir los modelos.

- Todos los componentes deben tener nombres distintos.
- Las interfaces contenidas dentro de un mismo puerto, deben tener nombres distintos.

- Cada componente debe tener al menos un puerto definido.

- Cada componente compuesto debe tener en su interior al menos dos componentes y dos conectores.

Conociendo estas reglas básicas, es hora de comenzar a conocer como se pueden construir nuestros modelos.

En la figura B.1 se puede ver el area de trabajo del editor gráfico. En la zona en blanco será donde comencemos a añadir los componentes y construiremos nuestros modelos. A la derecha de la figura B.1 encontramos la paleta de herramientas, donde se pueden ver los iconos con los que podremos añadir componentes simples, componentes compuestos, conectores, puertos, etc.

Figura B.1: Area de trabajo del editor gráfico.

Para añadir un componente simple, haremos click en el icono 'SC' y al dibujarlo en el area de trabajo veremos lo que aparece en la figura B.2. En la zona derecha del componente es donde se añadirán los puertos del componente. Para ello tendremos que seleccionar el icono 'P' de la paleta y hacer click en el area derecha del componente. Una vez hayamos añadido un puerto, podremos añadir las interfaces que sean necesarias.

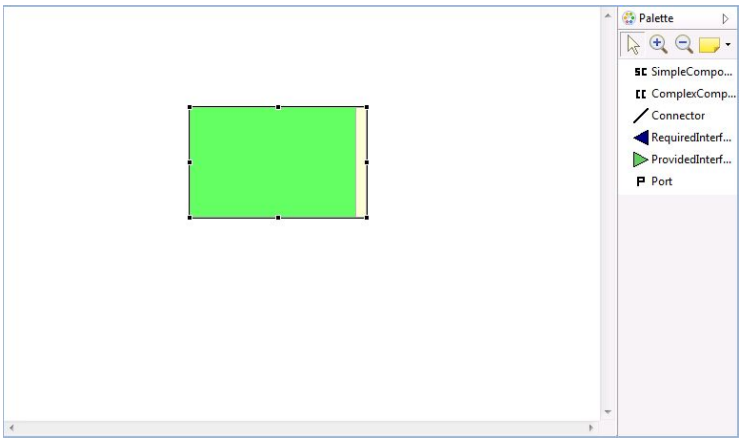

Figura B.2: Componente simple en el editor gráfico.

Añadiendo otro componente con su puerto y su interfaz, podemos ver en la figura B.3 que ya se han conectado los componente a través de sus respectivos puertos. A la hora de conectar los componentes con un conector, hay que hacer click en el recuadro que hace referencia al puerto, no en el dibujo de las interfaces o del componente, ya que otra forma, el conector no se 'enganchará'.

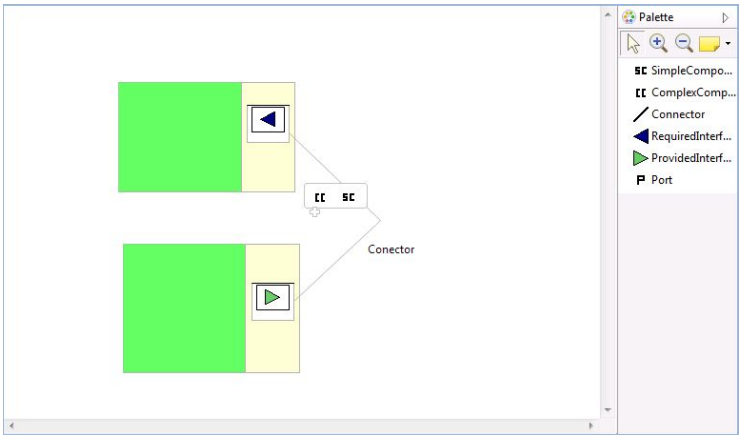

Figura B.3: Componente simple en el editor gráfico.

Por último, el editor nos permite ver nuestro modelo con un editor tree-editor, como se puede ver en la figura B.4 que en alguna ocasion puede ser de utilidad para modificar alguna propiedad de uno de los componentes sin necesidad de buscarlo en el editor gráfico.

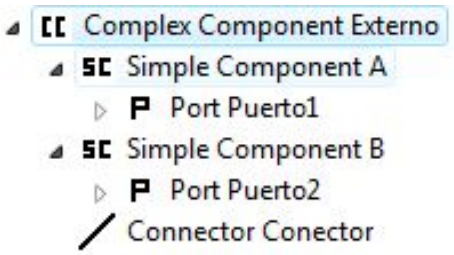

Figura B.4: Tree-editor del modelo realizado con el editor gráfico.

B.2. Ejercicios

A continuación se presentan los ejercicios que los usuarios tendrán que realizar para realizar la evaluación del editor gráfico. Entre los ejercicios a realizar se evaluarán:

- Curva de Aprendizaje.
- Escalabilidad.
- Legibilidad o Comprensibilidad del lenguaje.
- Impresión del Usuario.

B.2.1. Curva de Aprendizaje

1. **Ejercicio 1**.

   El primer ejercicio consiste en un componente compuesto de nombre «compuestoA» que contendrá en su interior dos componentes simples: «simpleA» y «simpleB». El componente 'simpleA' tendrá dos puertos en su interior «SA1» y «SA2», el puerto «A1» con una interfaz de tipo «provided» y nombre «I1» y el puerto «A2» con una interfaz de tipo «required» y nombre «I2». El componente «simpleB» tendrá un único puerto «SB1» con una interfaz de tipo «provided» y nombre «I2». El componente «compuestoA» tendrá también un puerto «CA1» con una interfaz de tipo «provided» y nombre «I1». El componente «compuestoA» tendrá dos conectores. El conector «conectorCA1» que unirá el puerto «SB1» con el «SA2» y el conector «conectorCA2» unirá el puerto «SA1» con el «CA1».

   No olvide rellenar el cuestionario, indicando el tiempo de comienzo y finalización del ejercicio 1.

2. **Ejercicio 2.**

   En el segundo ejercicio, vamos a comenzar añadiendo algunas modificaciones a lo hecho anteriormente y añadir más componentes. Comenzaremos añadiendo un componente simple más al componente compuesto «compuestoA» cuyo nombre será «simpleC» y tendrá un puerto «SC1» con una interfaz de tipo «provided» llamada «I3». A continuación añadiremos un puerto al componente «simpleA», que llamaremos «SA3» y que tendrá una interfaz de tipo «required». Añadiremos un conector al componente «compuestoA» con el que uniremos los puertos «SC1» con el «SA3». El componente «compuestoA» tendrá un nuevo puerto «CA2» con una interfaz de tipo «required» llamada «I4». Añadiremos al modelo un nuevo componente simple «simpleD» con un puerto «SD1» que tendrá una interfaz de tipo «provided» llamada «I4». Se creará un conector «conectorE1» que unirá los puertos «SD1» con «CA2». Además, añadiremos otro nuevo componente compuesto «compuestoB» que tendrá su interior dos componentes simples, «simpleE» y «simpleF» con un puerto cada uno, «SE1» y «SF1» respectivamente, y cada puerto una interfaz de tipo «required», «I5» e «I6» respectivamente. «compuestoB» tres puertos: «CB1», «CB2» y «CB3». Cada uno de los puertos tendrá una única interfaz de tipo «required» y nombres «I5», «I6» e «I1», además, tendrá dos conectores, «conectorCB1» que unirá los puertos «CB1» con «SE1» y «CB2» con «SF1». Añadiremos otro conector «conectorE2» que unirá el puerto «CA1» con el puerto «CB3».

   No olvide rellenar el cuestionario, indicando el tiempo de comienzo y finalización del ejercicio 2.

3. **Ejercicio 3.**

   En el tercer y último ejercicio, se quiere encapsular el componente «simpleD» y «compuestoB» en un componente compuesto que llamaremos «compuestoC». Para ello, con la metodología que se ha seguido en los dos ejercicios anteriores, se deberán añadir los puertos pertinentes al componente compuesto nuevo de forma que las interfaces de los componentes internos queden conectados con el componente «compuestoA» de la misma forma que estaban en el ejercicio 2.

   No olvide rellenar el cuestionario, indicando el tiempo de comienzo y finalización del ejercicio 3.

1. **Respecto al primer ejercicio, indique el grado de dificultad que ha supuesto su realización.**

   Anote la hora de inicio del ejercicio:.....

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

   Anote la hora de finalización del ejercicio:.....

2. **Respecto al primer ejercicio, comente las dificultades que ha encontrado durante su realización.**

3. **Respecto al primer ejercicio, una vez finalizado, observando el resultado obtenido, si tuviera que explicar el modelo mostrándoselo a otra persona, valore qué grado de dificultad tendría dicha acción.**

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

4. **Respecto al segundo ejercicio, indique el grado de dificultad que ha supuesto su realización.**

   Anote la hora de inicio del ejercicio:.....

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

   Anote la hora de finalización del ejercicio:.....

5. **Respecto al segundo ejercicio, comente las dificultades que ha encontrado durante su realización.**

6. **Respecto al segundo ejercicio, una vez finalizado, observando el resultado obtenido, si tuviera que explicar el modelo mostrándoselo a otra persona, valore qué grado de dificultad tendría dicha acción.**

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

7. **Respecto al tercer ejercicio, indique el grado de dificultad que ha supuesto su realización.**

   Anote la hora de inicio del ejercicio:.....

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

   Anote la hora de finalización del ejercicio:.....

8. **Respecto al tercer ejercicio, comente las dificultades que ha encontrado durante su realización.**

9. **Respecto al tercer ejercicio, una vez finalizado, observando el resultado obtenido, si tuviera que explicar el modelo mostrándoselo a otra persona, valore qué grado de dificultad tendría dicha acción.**

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

B.2.2. Escalabilidad

Partiendo de la experiencia previa obtenida con los ejercicios anteriores con el editor y observando los modelos que se presentan a continuación:

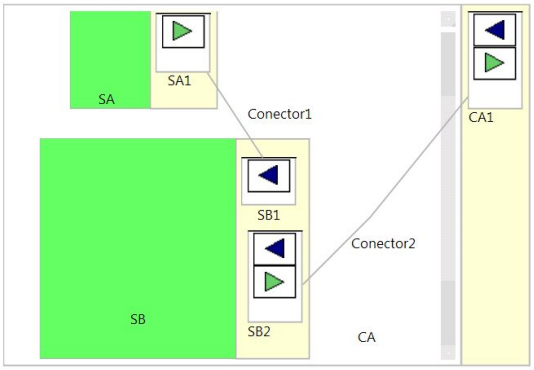

Figura B.5: Modelo 1.

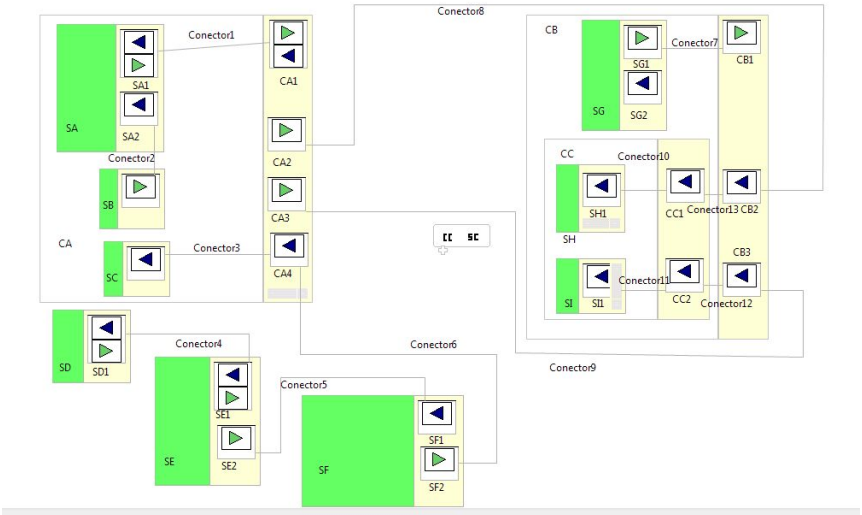

Figura B.6: Modelo 2.

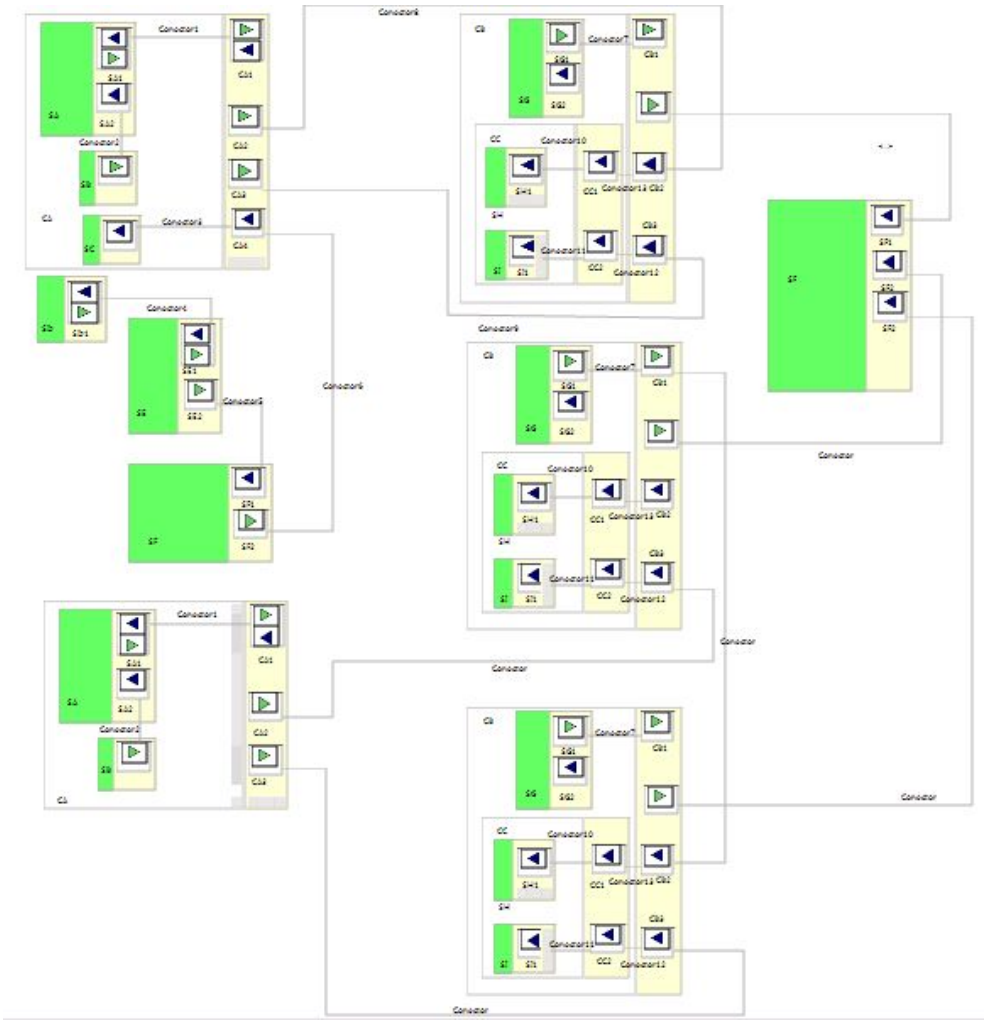

Figura B.7: Modelo 3.

1. **Observando el modelo número uno, indique el grado de dificultad que cree que supondría implementar ese modelo con el editor.**

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

2. **¿Qué ventajas o desventajas cree que tendría el realizar el modelo con un editor de este tipo?**

3. **Observando el modelo número dos, indique el grado de dificultad que cree que supondría implementar ese modelo con el editor.**

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

4. **¿Qué ventajas o desventajas cree que tendría el realizar el modelo con un editor de este tipo?**

5. **Observando el modelo número tres, indique el grado de dificultad que cree que supondría implementar ese modelo con el editor.**

   ◯ Muy fácil.
   ◯ Fácil
   ◯ Normal
   ◯ Difícil
   ◯ Muy difícil

6. **¿Qué ventajas o desventajas cree que tendría el realizar el modelo con un editor de este tipo?**

B.2.3. Legibilidad o Comprensibilidad de los Editores

La siguiente prueba consiste en definir con la mayor claridad posible, el modelo que se presenta a continuación. Es necesario añadir todos los detalles que se entiendan del modelo.

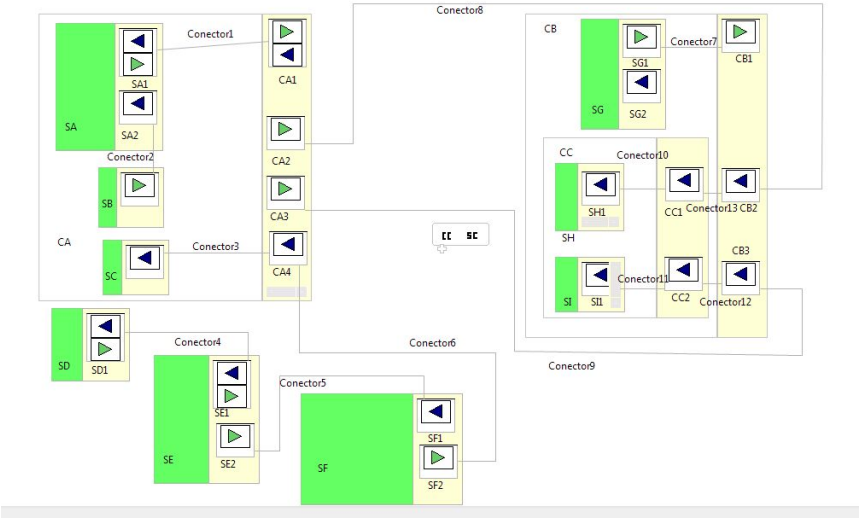

Figura B.8: Modelo de comprensibilidad del lenguaje.

B.2.4. Impresión del Usuario

- **Tras haber realizado los diferentes ejercicios propuestos, indique a continuación cualquier observación que no haya comentado en los ejercicios anteriores. Se pide entre otras cosas que comente ventajas y desventajas que cree que tiene el editor. Indique que cosas le han gustado o le han resultado comodas a la hora de trabajar con el editor y también aquellas que presenten dificultades/incomodidades.**

- **Para afrontar las posibles incomodidades o dificultades que presente el editor, indique qué características haría que el editor resultara más sencillo de utilizar.**

Bibliografía

[FB03] Ed Merks Raymond Ellersick Timothy J. Grose Frank Budinsky, David Steinberg. *Eclipse Modeling Framework: A Developer's Guide*. Addison Wesley, 2003.

[Fer96] M García Ferrando. La encuesta. *El análisis de la realidad social. Métodos y técnicas de investigación (3ª ed.), Alianza Universidad Textos, Madrid*, pages 167–201, 1996.

[Ger01] Jean Bézivin Olivier Gerbé. Towards a precise definition of the omg/mda framework. In *Proceedings*, page 273. IEEE, 2001.

[Gro09] Richard C Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley Professional, 2009.

[MGR+08] B Mora, F García, F Ruiz, M Piattini, A Boronat, A Gómez, JÁ Carsí, and I Ramos. Jisbd2007-08: Software generic measurement framework based on mda. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 6(4):363–370, 2008.

[mof05] *Meta Object Facility (MOF) Specification Version 1.4.1*. Number formal/05-05-05. Object Management Group, 2005.

[Om02] G. Om. XML Metadata Interchange (XMI) Specification. Technical report, Object Management Group, 2002.

[omg03] *MDA Guide Version 1.0.1*. Object Management Group, 2003.

[P+09] Terence Parr et al. Antlr parser generator. *Online Stand Dezember*, 2009.

[PGP10] Claudia Pons, Roxana Giandini, and Gabriela Pérez. Desarrollo de software dirigido por modelos. conceptos teóricos y su aplicación práctica. *1er. Edición. EDULP & McGraw-Hill Educación*, 2010.

[Rob02] Colin Robson. *Real world research: A resource for social scientists and practitioner-researchers*, volume 2. Blackwell Oxford, 2002.

[Spe06] OMG Available Specification. Object constraint language, 2006.

[uml11] *OMG Unified Modeling LanguageTM (OMG UML), Superstructure Version 2.4.1*. Number formal/2011-08-06. Object Management Group, 2011.

[WRH+12] Claes Wohlin, Per Runeson, Martin Hst, Magnus C Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in software engineering*. Springer Publishing Company, Incorporated, 2012.

[Yin09] Robert K Yin. *Case study research: Design and methods*, volume 5. Sage, 2009.