Ingeniería inversa de GUIs

Existen numerosos sistemas en funcionamiento que fueron desarrollados en los años 90 utilizando entornos RAD (Rapid Application Development), tales como Delphi, Visual Basic u Oracle Forms. Es interesante la migración de este tipo de aplicaciones para aprovechar las funcionalidades que las tecnologías más modernas ofrecen, como distribución, transaccionalidad, eficiencia, usabilidad, etc. Un aspecto importante a considerar es la migración de la interfaz gráfica (GUI).

Para construir la interfaz gráfica, este tipo de entornos se basan en arrastrar componentes gráficos desde una paleta a un lienzo. La descripción de la interfaz normalmente se almacena en un fichero textual, que puede procesarse para obtener información de utilidad y automatizar parte de la migración.

El proyecto se compone de tres tareas.

1. Creación de un inyector para la descripción de una GUI desarrollada en Delphi o Visual Basic. Se utilizarán algunas de las herramientas vistas en clase, como xText o Gra2MoL.
2. Generación de una GUI equivalente en alguna tecnología moderna, por ejemplo Swing o HTML/CSS. No se requiere detección de layout.
3. Obtención de un modelo de datos aproximado a partir de los datos pedidos al usuario. Se requiere implementar alguna heurística simple.

Será suficiente considerar un subconjunto de la funcionalidad de Delphi o Visual Basic, es decir, algunos componentes básicos como etiquetas, cajas de texto, calendario, o frames.

Derivar megamodelos a partir de tareas ANT

La herramienta ANT es ampliamente utilizada para describir las tareas que deben realizarse para automatizar la compilación de un proyecto de software. Por ejemplo, es habitual escribir ficheros ANT para compilar un proyecto java, utilizando la tarea *javac* para expresar la ejecución del compilador de Java. Así, existen tareas especializadas para ejecutar transformaciones de modelos, generadores de código, etc.

Por otra parte, al aplicar DSDM es común construir cadenas de transformación donde la salida de un proceso se convierte en la salida de otro. Cuando las cadenas de transformación son complejas el mantenimiento del proyecto se hace difícil.

Un modelo que describe la relación entre los artefactos que componen un proyecto se denomina *megamodelo*. Su interés radica en que describe de manera abstracta las relaciones entre los tipos de artefactos de un proyecto. Obsérvese que una especificación ANT es concreta ya que indica explícitamente qué fichero es la entrada o salida de una tarea. En cambio un megamodelo indica relaciones entre tipos de artefactos.

El proyecto se compone de tres tareas.

- Construcción de un metamodelo para describir megamodelos simples con al menos los siguientes conceptos: inyector, transformación M2M, generador de código y cadena de transformación.
- Descubrimiento de un megamodelo para algunos ficheros ANT simples que serán proporcionados por el profesor. Para ello se proponen dos alternativas:
  - Utilizar ETL para procesar el fichero ANT (que es XML)
  - Utilizar la herramienta de EMF para convertir ficheros XML en modelos

Lenguaje para procesar hojas de cálculo

Las hojas de cálculo se usan a menudo como bases de datos sencillas para almacenar y compartir información que sigue un esquema concreto en cada caso (la información se organiza en filas/columnas según el diseño de la hoja de cálculo hecho por el usuario). En ocasiones esta información puede ser para automatizar alguna tarea o como punto de partida para generar el esqueleto de algún sistema en construcción. En este sentido a veces la hoja de cálculo funciona como una especie de DSL orientado a usuarios no especialistas.

La cuestión que se plantea es cómo extraer la información de la hoja de cálculo para que pueda ser procesado por herramientas de automatización como transformaciones de modelos o generadores de código. Una opción es utilizar alguna de las librerías disponibles para procesar hojas de cálculo, por ejemplo Jakarta POI (http://poi.apache.org/trans/es/index.html) o Ruby Spreadsheet (http://spreadsheet.rubyforge.org/). Dado un diseño de hoja de cálculo se crearía el programa correspondiente que extrae la información.

El principal problema de esta aproximación es que es complicado escribir programas robustos y que sean capaces de detectar los pequeños fallos o inconsistencias que el usuario puede cometer al introducir los datos en la hoja de cálculo (por ejemplo, escribir una cadena donde se espera un número, o no introducir un dato que es obligatorio).

En este proyecto se propone construir un DSL que disponga de construcciones especiales para examinar una hoja de cálculo y generar un modelo con la información recogida. Posibles construcciones serían: recorrer por filas hasta encontrar una celda con un determinado valor (o con la carencia de él), dada una fila *mapear* las columnas a propiedades de un objeto, o introducir aserciones para verificar que la hoja de cálculo tiene los valores esperados.

La ejecución del DSL tendría como entrada una hoja de cálculo, y como salida un modelo conforme a cierto metamodelo. Por tanto, el DSL construido se comporta como un inyector que permitirá manipular los datos de una hoja de cálculo utilizando herramientas del DSDM.

El proyecto se compone de tres tareas.

- Diseño del lenguaje, basándose en alguno de los ejemplos proporcionados por el profesor.
- Construcción de una sintaxis concreta con xText
- Implementación del motor de ejecución. Para la implementación se recomienda utilizar Jakarta POI o Ruby Spreadsheet como librerías de procesamiento de hojas de cálculo, pero otras pueden ser igualmente válidas. Hay dos opciones básicas para la implementación:
  - Construir un compilador utilizando EGL para generar código Java o Ruby.
  - Construir un intérprete en Java o Ruby, que ejecute el DSL a medida que lo analiza.

Dependiendo de la complejidad del lenguaje y de su funcionalidad, este proyecto podría ser realizado por más de una persona.

Lenguaje textual para especificar meta-modelos con múltiples niveles de instanciación

MetaDepth es una herramienta de meta-modelado que permite especificar sistemas con un número de meta-niveles arbitrario. Soporta el concepto de "potencia" visto en clase. Por ejemplo, uno podría especificar un sistema en tres meta-niveles mediante:

```
Model Store@2{
  Node Product{
    VAT@1 : double = 7.5;
    price : double = 10;
  }
}

Store Library{
  Product Book{ VAT = 7;  }
  Product CD  { VAT = 15; }
}

Library MyLibrary{
  Book mobyDick{ price=14;}
  CD   NovenaSinfonia { price=20; }
}
```

donde *Store* es un modelo de potencia 2, que es instanciado sucesivamente por *Library* y *MyLibrary*. MetaDepth es una herramienta hecha en Java, que funciona con un *shell* de línea de comandos, y que puede descargarse aquí: http://astreo.ii.uam.es/~jlara/metaDepth/

El proyecto consistiría en crear un editor textual con xText, que permita editar modelos MetaDepth con la sintaxis anterior desde Eclipse. Así mismo, se desea construir un "puente" entre MetaDepth y EMF, de tal manera que a partir de un meta-modelo (como *Store* en el listado anterior) se genere un meta-modelo ecore.

Este trabajo es **ampliable a un trabajo de fin de máster**, por ejemplo implementando transformaciones que agrupen sistemas que tengan meta-modelos arbitrarios en dos meta-niveles, y al revés.

Bibliografía:

Juan de Lara, Esther Guerra: Deep Meta-modelling with MetaDepth. TOOLS (48) 2010: 1-20

Generación automática de editores gráficos a partir de meta-modelos

Graphiti (http://www.eclipse.org/graphiti/) es un framework para la construcción rápida de editores diagramáticos en Eclipse. En particular, permite la instanciación de meta-modelos ecore (o de otro tipo) durante la edición del diagrama. Este framework es muy útil a la hora de definir lenguajes visuales de dominio específico.

EMF permite la definición de meta-modelos, y su instanciación mediante un editor en forma de árbol. No obstante, esta forma de editar modelos es extremadamente tediosa y propensa a errores. Más aún, no consigue uno de los objetivos del modelado, que es un mejor entendimiento del sistema bajo estudio mediante modelos intuitivos.

En este proyecto, se plantea la generación automática de editores visuales basados en Graphiti a partir de un meta-modelo EMF. El proyecto se plantea en varias fases, pudiéndose realizar por dos personas dependiendo del grado de sofisticación resultante. Para una sola persona, los requisitos mínimos son las fases 1 y 2.

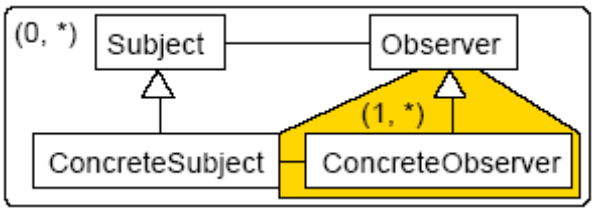
Fases del proyecto:
1: Generar un editor básico a partir de un meta-modelo, sin posibilidad de configuración. Así, cada clase del meta-modelo se dibujará en forma de caja con el nombre del tipo dentro. Las asociaciones se dibujarán como flechas.
2: Permitir editar los atributos de los objetos.
3: Diseñar un lenguaje de dominio específico para configurar el editor resultante (p.ej. el color de los objetos o la forma). Este lenguaje puede ser un lenguaje externo, o implementarse mediante anotaciones en meta-modelos ecore (al estilo del framework EUGENIA).

Este trabajo es **ampliable a un trabajo de fin de máster**, por ejemplo generando un editor con acciones sofisticadas (copy-paste, layouts, etc), así como definiendo lenguajes de configuración sofisticados, por ejemplo que permitan asociar relaciones espaciales a las asociaciones del meta-modelo (dentro de, adyacente a, etc).

Lenguaje textual para especificar patrones de diseño

El proyecto se compone de tres tareas.

1. Creación de un meta-modelo ecore que permita definir patrones de diseño. Un patrón de diseño estará formado por un conjunto de clases y asociaciones que describen la estructura del patrón, sus roles, y posibles regiones de variabilidad que indiquen cuántas veces puede aparecer un determinado rol o roles (en general un conjunto de elementos) en una instancia del patrón. La siguiente imagen muestra un ejemplo de patrón de diseño.



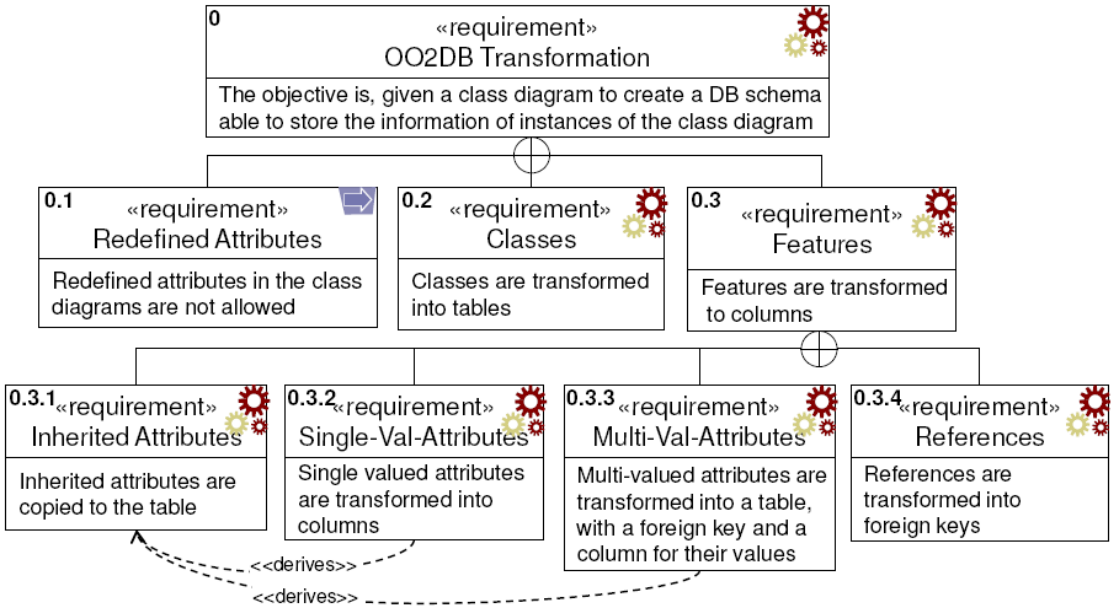
2. Creación de un editor textual con xText para definir patrones de diseño conformes al meta-modelo del punto 1.
3. Creación de un generador de código que obtenga las instancias de un determinado patrón dentro de un modelo, y las anote con los roles del patrón. El generador de código se realizará con un lenguaje de plantillas como Jet o EGL. El código generado que se usará para detectar instancias de patrones deberá ser EOL.

Lenguaje visual de especificación de requisitos para transformaciones de modelos

Creación de un editor visual para un lenguaje de especificación de requisitos para transformaciones de modelos. El alumno recibirá el meta-modelo ecore del lenguaje y una definición de su sintaxis concreta. El editor se desarrollará con Graphiti.

El proyecto se compone de dos tareas.

1. Creación del editor visual con Graphiti. La apariencia de los diagramas será la siguiente:



   El editor numerará los requisitos automáticamente (número en la esquina superior izquierda de cada caja).

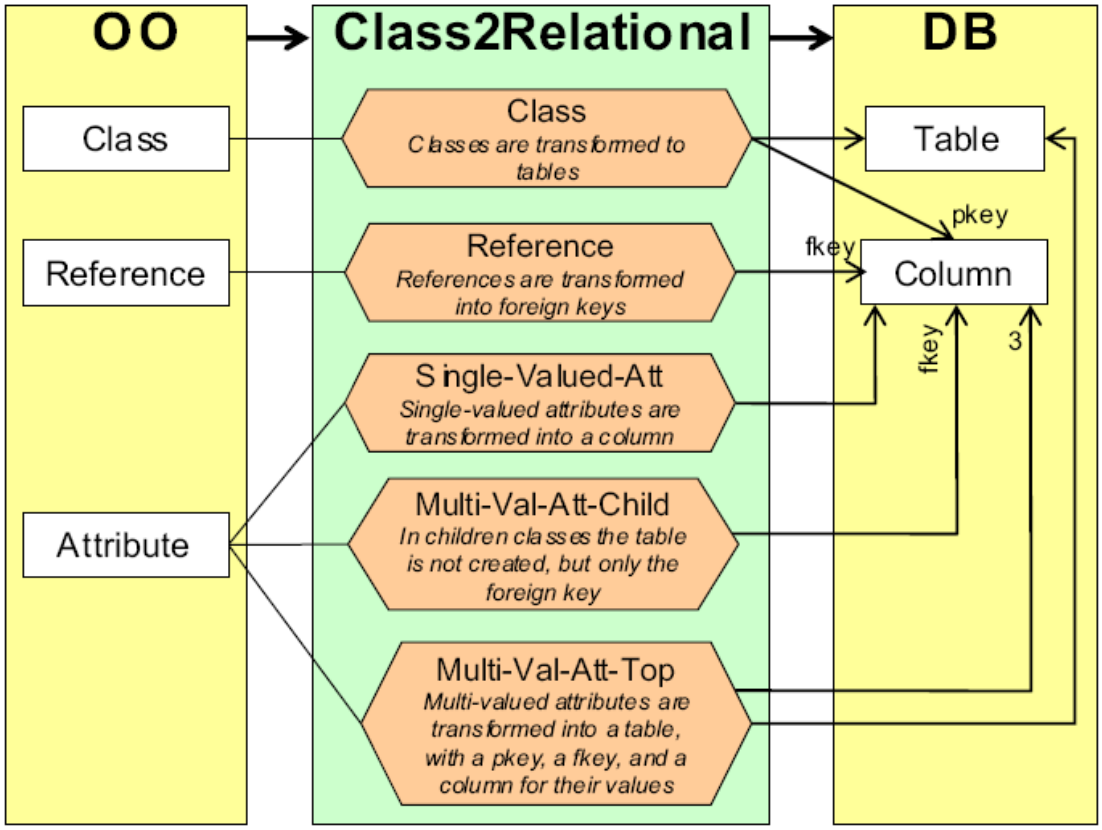
2. Creación de un generador de código que genere documentación html a partir de un diagrama de requisitos. El generador se realizará con un lenguaje de plantillas como Jet o EGL.

Lenguaje visual de especificación de relaciones (*mappings*) entre meta-modelos

Creación de un editor visual para un lenguaje que permite definir relaciones entre meta-modelos. El objetivo de este lenguaje es diseñar con un alto nivel de abstracción transformaciones de modelos. El alumno recibirá el meta-modelo ecore del lenguaje y una definición de su sintaxis concreta. El editor se desarrollará con Graphiti.

El proyecto se compone de dos tareas.

1. Creación del editor visual con Graphiti. La apariencia de los diagramas será la siguiente:



   Cada caja amarilla podrá configurarse con el nombre de un meta-modelo ecore, cuyo nombre se mostrará en la parte superior. Dentro podrán añadirse elementos que representen nombres de clases del meta-modelo. En el centro se mostrará una caja verde para definir relaciones entre las clases de los meta-modelos a la izquierda y la derecha.

2. Añadir al editor un validador que compruebe que las clases que se añaden en las cajas amarillas realmente pertenecen al meta-modelo indicado.