13. Pruebas

En esta fase se realizan los juegos de prueba para detectar los posibles errores que puedan existir en la aplicación una vez compilada e implementada. Las pruebas que se han llevado a cabo para comprobar el correcto funcionamiento de la aplicación son las pruebas generales y las especificas.

13.1. Pruebas generales

Estas pruebas se han realizado para el control del buen funcionamiento de la aplicación una vez se ha implementado:

- Se muestran y activan los menús y las pantallas en la herramienta Poseidon en los momentos en que se producen eventos por parte del usuario.

- Se reconocen todos los elementos utilizados que se describieron en capítulos anteriores del esquema conceptual para poder traducirlos al esquema lógico y físico.

- Se valida correctamente el esquema conceptual propuesto por el usuario.

- Se comprueba que de las clases seleccionadas del esquema conceptual por el usuario para traducir al esquema lógico o físico, se seleccionan y traducen solo las asociaciones de las clases que ha seleccionado.

- Se traducen de forma correcta todas las asociaciones ya sea materializándolas o añadiendo las claves foráneas a una de las tablas que participan en las asociaciones. Esta operación es costosa debido a que se puede haber comenzado a traducir una asociación que posteriormente tendrá que ser modificada (añadiendo más atributos sobre la clave foránea de la asociación) debido a otras asociaciones relacionadas.

- Se comprueba que todos los casos posibles de traducción de las clases y asociaciones al esquema lógico están representados de forma correcta en la herramienta y que se corresponden con las reglas de traducción (ver capitulo 7).

- Se comprueba que todos los casos posibles de traducción de las clases y asociaciones al esquema lógico están traducidos de forma correcta en SQL estándar y PostgreSQL.

- Se comprueba que se generan los triggers en SQL de forma correcta para el control de las restricciones que el esquema lógico no puede controlar.

- Se comprueba que la creación de las vistas es correcta, creando vistas e incorporándolas en la base de datos de PostgreSQL para comprobar que funcionan.

13.2. Pruebas específicas

Las pruebas específicas se realizan a medida que se va implementando cada una de las partes de las funcionalidades. Con estas pruebas podemos ver que las partes que conforman la aplicación funcionan de manera correcta.

Validación del esquema conceptual

Para comprobar la correcta validación de un esquema, se han creado todas las posibles anomalías que pueden existir en el esquema conceptual (ver capitulo 5.6). Para cada anomalía se ha ejecutado el menú de validación incorporado a la herramienta para comprobar que el mensaje que se retorna es el correcto.

Generación del Esquema Lógico en UML Data Modeling Profile

Las pruebas de generación del esquema lógico constan de las siguientes fases:

- Generación automática de claves primarias para las clases que no tengan, a excepción de las clases que representan las asociaciones n-arias y las clases asociativas.
- Traducción de las asociaciones binarias para los tipos existentes ((1-*), (1-1) y (*-*)).
- Traducción de las asociaciones n-arias cuando:
  - Todos los enlaces de la asociación son del tipo *.
  - Todos los enlaces de la asociación son del tipo 1.
  - Existen más de dos enlaces con multiplicidad del tipo 1.
  - Existe solo un enlace con multiplicidad 1.
- Generalizaciones, composiciones y agregaciones con diferentes caminos.
- Deadlocks en asociaciones 1..*.
- Asociaciones reflexivas con o sin clase asociativa.
- Generación y representación en el esquema lógico de los triggers de cada tabla. En este punto se comprueba, además de que se ha creado todos los

triggers necesarios para controlar las restricciones, que el código generado se corresponde con la restricción a controlar, es decir:

- El trigger se crea sobre la tabla correcta.
- En cada condición de la sentencia SELECT, se corresponden los nombres de atributos de la parte izquierda de la igualdad con los de la parte derecha.

Y para cada tipo de asociación se realizan las pruebas referentes a las diferentes opciones de traducción (ver capitulo 7), es decir, se crean diferentes juegos de prueba para cada tipo de asociación, para comprobar que la traducción generada se corresponde con las reglas teóricas.

Creación de las vistas

En las pruebas de creación de vistas se comprueba que solo se muestran los atributos referentes a las tablas seleccionas por el usuario que forman parte la vista, que las condiciones de la vista hacen referencia a las claves primarias de la tabla principal y las claves foráneas de las otras tablas.

Además, se comprueba que la representación generada en UML Data Modeling, se corresponde con el código SQL y éste no contiene errores.

Generación de código SQL Estándar y PostgreSQL

En esta fase se comprueba que el código sql generado se corresponde con el esquema lógico. Además, se comprueba que no contiene errores, validándolo y realizando pruebas reales sobre la base de datos de PostgreSQL con los triggers y las tablas creadas para comprobar que funcionan. Esta fase es bastante costosa debido a que se ha de analizar la creación de las tablas y triggers de manera manual y contrastarlo con el generado por la aplicación. Además, para cada triggers se realizan juegos de prueba sobre la base de datos de PostgreSQL para comprobar que los triggers producen las excepciones cuando no se cumplen las restricciones del esquema.

14. Gestión Final del Proyecto

14.1. Justificación

La planificación que se realiza al inicio de un proyecto suele variar respecto a la duración final de un proyecto, aunque las etapas de un proyecto sean las mismas, pueden surgir aspectos no hallados al principio del proyecto que aumentan o disminuyan la duración de algunas de las etapas definidas. Las situaciones que ha hecho diferir la planificación inicial de este proyecto se justifican a continuación:

- Recuperar y representar la información de los esquemas en la herramienta Poseidon a partir de las clases Java que implementan el metamodelo UML[18] ha resultado ser más complicado de la previsión inicial a la hora de determinar el funcionamiento de estas clases de Poseidon, debido a que muchas de la funciones contenían parámetros que no se sabia como recuperar para poder utilizarlas. Como por ejemplo, como recuperar la información de las asociaciones en los esquemas conceptuales y representar las asociaciones y sus multiplicidades en los esquemas lógicos.

- La falta de restricciones a la hora de representar los esquemas conceptuales en la herramienta Poseidon, provoca que la funcionalidad tenga que realizar muchas comprobaciones con el fin de detectar posibles anomalías en los esquemas representados, de modo que se ha requerido mayor tiempo para desarrollar esta función. Como por ejemplo, comprobar la existencia de ciclos en agregaciones o generalizaciones.

- La traducción de los esquemas lógico a los esquemas físico en SQL, ha resultado ser más compleja debido al análisis que se tiene que realizar sobre los esquemas lógicos para generar las restricciones en el esquema físico. Por ejemplo, generar las restricciones de multiplicidad de las asociaciones mediante triggers.

- La potencia para definir esquemas conceptuales así como las muchas posibilidades de traducción a los esquemas lógicos y físicos, provoca que se tengan que realizar una gran cantidad de pruebas para comprobar que todas las traducciones posibles se corresponden con las reglas teóricas. Por ejemplo, comprobar que se traducen de forma correcta los diferentes tipos de asociaciones binarias, las generalizaciones según el tipo de especificación y que se generan los triggers necesarios para controlar las restricciones.

14.2. Planificación real

Después de exponer en el apartado anterior, cuales ha sido los diferentes aspectos no detectados al inicio del proyecto, se describe a continuación la duración real de las fases que han diferido de la planificación inicial:

- La duración prevista en la etapa para obtener la información de los esquema conceptuales ha resultado mayor debido a que se ha requerido mayor tiempo en determinar que clases y funciones eran necesarias para acceder al metamodelo UML de los esquemas conceptuales y como retornaba la información para posteriormente formatear los datos para facilitar su interpretación. La previsión inicial fue de 75 horas y la duración real ha sido de 78 horas.

- La etapa de validación de los esquemas ha resultado más duradera debido a la gran cantidad de errores que pueden existir en los esquemas conceptuales y que la herramienta Poseidon no controla. La previsión inicial fue de 30 horas y la duración real ha sido de 40 horas.

- La etapa de pruebas sobre el esquema lógico generado por la aplicación ha resultado más duradera de la previsión inicial debido a la gran cantidad juegos de prueba genéricos sobre diferentes esquemas conceptuales que se deben hacer para comprobar todas las posibles traducciones al esquema lógico. La previsión inicial fue de 16 horas y la duración real ha sido de 20 horas.

- La etapa de obtención del esquema lógico para traducirlo al esquema físico ha resultado mas fácil de desarrollar, debido ha que se utiliza el mismo procedimiento que el realizado para obtener los esquemas lógico a partir de los esquemas conceptuales. La previsión inicial fue de 10 horas y la duración real ha sido de 3 horas.

- La etapa de traducción de los esquemas lógico a los esquemas físico en SQL ha resultado muy compleja debido al análisis que se tiene que realizar para generar los diferentes tipos de restricciones mediante triggers y restricciones integridad referencial, de los esquemas lógicos. La previsión inicial fue de 100 horas y la duración real ha sido de 115 horas.

- La fase de pruebas sobre el esquema físico generado ya sea en PostgreSQL o SQL estándar ha resultado más duradera de la previsión inicial debido a las fases que

requiere la validación (comprobación de que el SQL creado se corresponden con el esquema lógico, los triggers controlan realmente las restricciones del esquema lógico). La previsión inicial fue de 15 horas y la duración real ha sido de 28 horas.

La previsión inicial de horas necesarias para elaborar el proyecto fueron 778 horas. Una vez revisada la planificación y habiendo expuesto las dificultades encontradas en algunas de las etapas del proyecto el número de horas requeridas han sido 808 horas, de modo que se ha producido una desviación de 30 horas más de dedicación a su desarrollo a causa de las situaciones expuestas anteriormente.

Una vez justificadas las etapas que han hecho variar la previsión inicial, el diagrama de Gantt real resultante se muestra en la siguiente pagina.

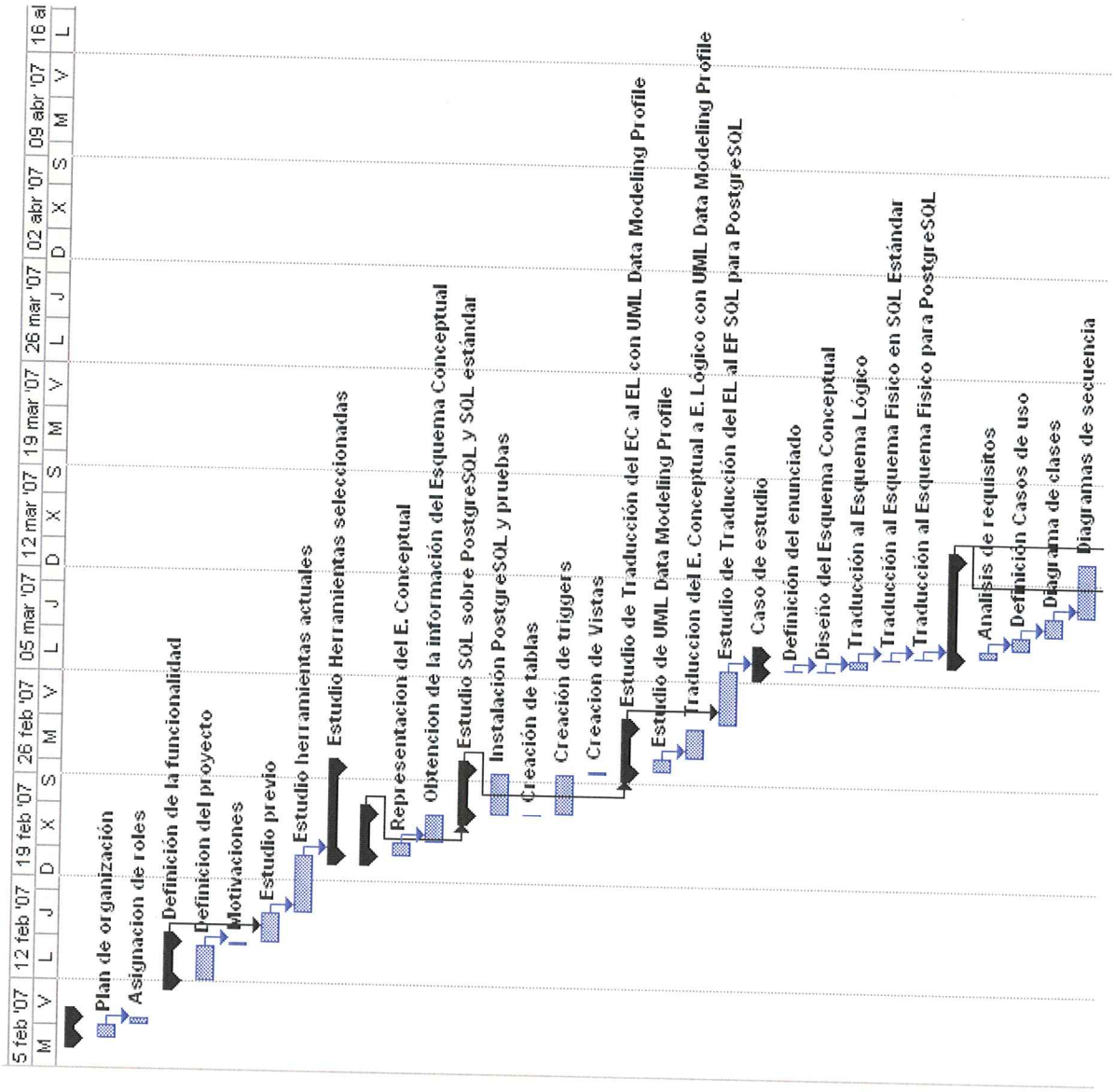

Ilustración 14-1: Planificación real (I)

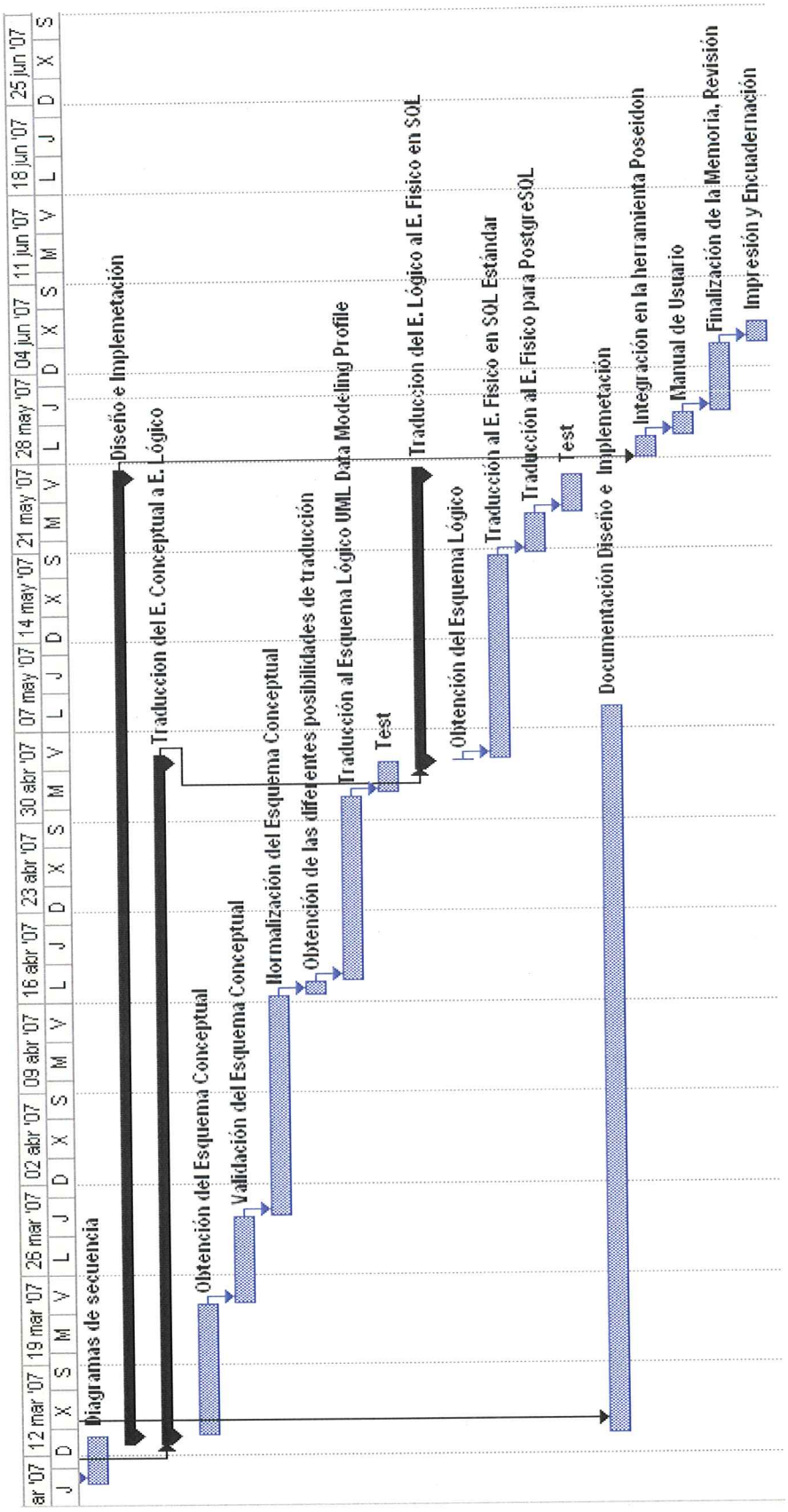

Ilustración 14-2: Planificación real (II)

14.3. Análisis Económico

En este apartado se estima el coste total del proyecto a partir de las horas trabajas según la planificación real descrita anteriormente, los recursos software y hardware requeridos para el desarrollo del proyecto.

El precio hora de los recursos humanos según el tipo de rol son los siguientes:

- Jefe de proyecto 30€/hora
- Analista 20€/hora
- Arquitecto 10€/hora

Los recursos software que se requieren para desarrollar el proyecto son los siguientes:

- MS Office 250 €
- MS Project 300 €

Los recursos hardware que se requieren para desarrollar el proyecto son los siguientes:

- 1 PC Pentium IV 1200 €
- 1 Impresora tinta 250 €

Las horas de trabajo reales para cada uno de los roles son los siguientes:

- Jefe de Proyecto 27 horas
- Analista 326 horas
- Arquitecto 455 horas

De acuerdo con la planificación real definida y el costo de cada uno de los recursos el importe total del proyecto asciende a **13.880€.**

15. Conclusiones

15.1. Exportación de la funcionalidad

En la actualidad, muchas de las herramientas de modelado UML existentes en el mercado no ofrecen la posibilidad de integrar nuevas funcionalidades, esto genera una gran dependencia hacia las empresas que las han desarrollado.

A pesar de que la gran tendencia es desarrollar herramientas de modelado UML con principios de extensibilidad, la gran mayoría de organizaciones desarrollan sus herramientas de forma cerrada a posibles extensiones por parte de los usuarios.

La funcionalidad de intercambio de modelos con otras herramientas de modelado UML se convierte en una meta alcanzable, teniendo en cuenta la estandarización sintáctica y semántica que presenta el lenguaje UML. El uso de esta estandarización por parte de las herramientas permite que los usuarios puedan intercambiar diagramas formas de representación de modelos UML de forma independiente a las herramientas mediante el uso de XMI [18].

Por ejemplo, si un grupo de desarrolladores utiliza una herramienta CASE o una herramienta de modelado visual, debiera existir la facilidad de transportarla a otra herramienta, independientemente del proveedor o fabricante de cada una de ellas.

Un hecho importante es hacer uso de los principios del metamodelo UML para la construcción de las herramientas, esto conllevaría poder hacer adaptaciones o realizar ampliaciones de la herramienta aumentando la portabilidad de la misma.

Como se expuso en el apartado 1.2, existen algunas herramientas que permiten añadir nuevas funcionalidades, pero solo si son proporcionadas por la propia empresa que las desarrolla, como por ejemplo Rational Rose [10], que implementa su propio estándar para acceder al metamodelo UML. Esta herramienta imposibilita el acceso a usuarios externos para saber el funcionamiento de la estructura interna de los patrones que utiliza la aplicación.

Otras herramientas como por ejemplo ArgoUML [9] o Visual Paradigm for UML [22], permiten la incorporación de nuevas funcionalidades mediante plug-ins, permitiendo a los usuarios realizar sus propias ampliaciones y mejoras sobre la herramienta de forma libre.

En el caso Visual Paradigm se utiliza el lenguaje Java [21] para su implementación, pero no se sigue el estándar definido por OMG para acceder al metamodelo UML [18]. Por ejemplo, para crear un plug-in deberíamos importar las librerías com.vp.plugin.VPPlugin y com.vp.plugin.VPPluginInfo, disponer de la librería de VP-UML *%VP_SUITE%/lib/openapi.jar* y para acceder al metamodelo UML tendríamos que usar la librería com.vp.plugin.model.

Visual Paradigm permite la importación/exportación de los modelos en XMI, de modo que los esquemas definidos y generados en Poseidon a partir de la funcionalidad pueden ser exportados a esta herramienta.

En el caso de ArgoUML, también se utiliza el lenguaje Java para su implementación y utiliza el estándar de OMG para acceder al metamodelo UML. Por ejemplo para acceder al metamodelo UML tendríamos que usar la librería org.argouml.model. ARGOUML además permite importar y exportar esquemas en formato XMI, de modo que también podríamos exportar a esta herramienta los esquemas generados por nuestra funcionalidad.

Herramientas como las citadas anteriormente, provocan que si se quiere crear una nueva funcionalidad, exista una gran dependencia de la herramienta y no pueda ser exportada fácilmente a otras herramientas existentes debido a que no siguen el estándar de acceso al metamodelo UML.

Respecto a la funcionalidad realizada, la posibilidad de exportarla a otras herramientas de modelado no es posible de forma directa, debido a que muchas herramientas no siguen el estándar de acceso al metamodelo UML, a excepción de la herramienta ArgoUML. En el caso de se quisiera incorporar la funcionalidad desarrollada a esta herramienta, se tendrían que realizar cambios con respecto a las librerías utilizadas y la forma de representar gráficamente los esquemas.

Para incorporar a Visual Paradigm la funcionalidad, se tendrían que realizar cambios en cuanto a la obtención de los esquemas y la representación gráfica. En el caso de Rational Rose, seria imposible ya que las posibles extensiones están sujetas a las proporcionadas por la propia empresa.

15.2. Posibles ampliaciones

La aplicación desarrollada en este proyecto puede tener diferentes ampliaciones, en este caso, sobre la herramienta Poseidon. Estas posibles ampliaciones afectan al vínculo existente entre el modelado UML y las bases de datos. Las posibles ampliaciones que podrían llevarse a cabo sobre este proyecto son las siguientes:

- La generación de los esquemas lógicos en la funcionalidad está representada en SQL estándar, de modo que además de la traducción de los esquemas lógicos a esquemas físicos para PostgreSQL realizada en este proyecto, se podrían incluir traducciones a esquemas físicos para otros tipos de SGBD, como por ejemplo, Oracle, MySQL, Informix o SQL Server.

- Las diferentes opciones de traducción de un esquema conceptual a un esquema lógico, se presentan cada vez que el usuario quiere traducir un esquema siempre y cuando el esquema conceptual haya variado, una posible ampliación sería generar automáticamente un fichero en el que se almacenarán la opciones de traducción por defecto de los esquemas y que cada vez que se tradujese el esquema conceptual, la aplicación mostrase un formulario con las opciones por defecto, pudiendo éste modificarlas cuando quisiese.

- Otra posible ampliación podría ser traducir los esquemas lógicos a los esquemas físicos de la base de datos sin necesidad de realizar el paso previo de traducción del esquema conceptual al esquema lógico. Para realizar esta ampliación seria necesario guardar todo el contenido del esquema lógico, por ejemplo en un fichero XML. Esta ampliación conllevaría realizar validaciones de los esquemas lógicos guardados en formato XML que incluirían la validación de las restricciones (como por ejemplo las restricciones textuales, de integridad referencial, de multiplicidad y de las generalizaciones) definidas en el esquema.

- Este proyecto genera las restricciones del esquema lógico expresadas en SQL estándar y SQL para PostgreSQL, una posible ampliación podría ser generar las restricciones en OCL (Object Constraint Language) y permitir al usuario definir más restricciones en OCL sobre el esquema lógico generado. La definición de las restricciones por parte del usuario se realizarían en una serie de formularios incorporados en la propia herramienta que conllevaría validarlas para posteriormente ser traducidas al SQL específico.

- Otra nueva incorporación seria poder trabajar con las bases de datos desde la herramienta de modelado UML. Es decir, que la herramienta dispusiera de formularios que permitiesen crear operaciones de inserción, actualización, eliminación y consulta sobre la base de datos, trabajando desde la misma herramienta de modelado.

- Otra posible ampliación que resultaría interesante realizar, seria llevar a cabo la operación inversa a lo que se ha realizado en este proyecto, es decir, a partir de un esquema físico de la base de datos generar el esquema lógico correspondiente y a partir de éste, generar el esquema conceptual al que corresponde. Es decir, aplicar ingeniería inversa a una base de datos. La idea principal de aplicar ingeniería inversa a una base de datos es poder obtener un esquema logico partir de una base de datos ya implementada.

16. Bibliografía

16.1. Libros

[1] Larman, "UML y Patrones 2ª Edición", Prentice Hall, 2003.

[2] Dolors Costal Costa - M. Ribera Sancho Samsó - Ernest Teniente López, "Especificación de sistemas software en UML", Edicions UPC, 2003.

[3] Cristina Gómez - Enric Mayol - Antoni Olivé - Ernest Teniente, "Diseño de sistemas software en UML", Edicions UPC, 2003.

[4] A. Abelló, E. Rollón, "Apunts Disseny i Administració de Bases de Dades" (DABD), Apuntes, 2005.

[5] Peter Gulutzan - Trudy Pelzer, "SQL-99 Complete, Really", R&D Books, 1999.

[6] John C. Worsley - Joshua D. Drake, "Practical PostgreSQL", O'Reilly, 2002.

[7] Tomasz Szmuc - Krzysztof Zielinski, "Software Engineering: Evolution and Emerging Technologies", IOS Press, 2005.

16.2. Enlaces Web

[8] Sybase, www.sybase.com, 2007.

[9] CollabNet, argouml.tigris.org, 2006.

[10] IBM, www-306.ibm.com/software/rational, 2007.

[11] Gentleware, www.gentleware.com, 2007.

[12] Scott W. Ambler, www.agiledata.org/essays/umlDataModelingProfile.html, 2006.

[13] D. Gornik, www.jeckle.de/files/RationalUML-RDB-Profile.pdf, 2005.

[14] PostgreSQL, www.postgresql.org, 2007.

[15] Sun Microsystems, java.sun.com/j2se/1.5.0/docs/guide/jdbc, 2004.

[16] Mimer SQL, developer.mimer.com/validator/parser200x/index.tml#parser, 2007.

[17] Roland Bouman, rpbouman.blogspot.com/2006_02_01_archive.html, 2006.

[18] OMG, www.omg.org, 2007.

[19] Scott W. Ambler, www.agiledata.org/essays/mappingObjects.html, 2006.

[20] Apache Software Foundation, ant.apache.org, 2007.

[21] Sun Microsystems, java.sun.com, 2007.

[22] Visual Paradigm, www.visual-paradigm.com, 2007.