Agradecimientos *

Tengo que dar las gracias a mucha gente por todo lo que me ha sucedido hasta hoy; en especial a mi familia por haberme ofrecido la posibilidad de ser educado. A mis amigos por su apoyo incondicional, y todas aquellas personas que me han hecho enriquecer a nivel personal y profesional.

Éste proyecto no hubiera sido posible sin la desinteresada colaboración de Álex Ballarín Latre, quien nos ha proporcionado parte de la información necesaria para poder llevar a cabo este trabajo y se ha comprometido en un gran número de horas de su tiempo libre.

También debo agradecer de forma especial a mi compañero Ferran Basora Roca, con quien he compartido tantas horas estudiando, trabajando y divirtiéndonos.

Nuestro director Ernest Teniente López también debe aparecer aquí, puesto que su aportación ha sido tan enriquecedora como la de Álex.

Quisiera dedicar este proyecto a todas aquellas personas que he conocido en la Facultad de Informática de Barcelona. Gracias a ellos he aprendido mucho más de lo que la propia facultad me ha enseñado.

La memoria ha sido realizado en \$\LaTeX$, y se ha utilizado software considerado Software Libre. Agradecer finalmente al trabajo de miles de personas que hacen posible que dispongamos de Software Libre de calidad.
Soporte informático adicional *

Además del documento que el lector tiene delante, se proporciona un CD-ROM con el siguiente contenido.

- **Subproyecto CRUD:**
  Todos los artefactos necesarios para probar el CRUD explicado en la memoria.
  - *ProfileCIM:*
    proyecto que contiene el profile del nivel CIM
  - *ProfilePIM:*
    proyecto que contiene el profile del nivel PIM
  - *ProfilePSM:*
    proyecto que contiene el profile del nivel PSM
  - *CrudCIMtoPIMTransformation:*
    proyecto que transforma modelos CRUD CIM a modelos PIM
  - *CrudPIMtoPSMTransformation:*
    proyecto que transforma modelos CRUD PIM a modelos PSM
  - **Subproyecto testCrud:**
    proyecto de prueba para el Crud

- **Subproyecto Login:**
  Todos los artefactos necesarios para probar el Login explicado en la memoria.
  - *ProfileCIM:*
    proyecto que contiene el profile del nivel CIM
  - *ProfilePIM:*
    proyecto que contiene el profile del nivel PIM
  - *ProfilePSM:*
    proyecto que contiene el profile del nivel PSM
- **LoginCIMtoPIMTransformation:**
  proyecto que transforma modelos Login CIM a modelos PIM

- **LoginPIMtoPSMTransformation:**
  proyecto que transforma modelos Login PIM a modelos PSM

- **Subproyecto testLogin:**
  proyecto de prueba para el Login

- **Configuración del entorno de pruebas Entorno de trabajo.launch:**
  Configuración que hemos diseñado para tener un entorno de pruebas reducido con el fin de probar las transformaciones de manera eficiente.
Sobre este proyecto

Esta memoria está estructurada de forma lineal, a medida que han surgido inquietudes y siguiendo una planificación inicial. Se explica el proceso llevado a cabo como si fuera un dietario, un histórico del orden en el que se han sucedido los hechos.

En este proyecto ha habido una cooperación entre Daniel Albert Sánchez y Ferran Basora Roca. Hemos contado con la inestimable colaboración de Álex Ballarín Latre y Ernest Teniente López. El primero de ellos ha proporcionado una visión empresarial, mientras que Ernest nos ha enriquecido gracias a su experiencia académica.

El motivo por el cual ha habido dicha cooperación es que el alcance de dicho proyecto es tan ambicioso que resulta inviable que un único estudiante lo pueda realizar de forma individual. El producto final que se propone como resultado del trabajo realizado es un método.

Puesto que es un trabajo de investigación, en numerosas ocasiones nos hemos enfrentado a múltiples caminos a avanzar. De todos ellos hemos realizado un análisis exhaustivo y cuantificando las diferentes alternativas hemos escogido la mejor de ellas.

En ocasiones este análisis no ha resultado lo suficientemente preciso a causa de ausencia de información. Esto ha provocado que ciertos imprevistos hayan retrasado la planificación inicial. La filosofía del equipo ha sido analizar la situación y valorar en ese punto si seguir adelante o rectificar (en función del esfuerzo en horas/recursos requeridos).

Es por ello que ha habido una cooperación activa en los siguientes puntos:

- Estudio de la tecnología MDA
- Estudio de la herramienta de desarrollo Rational Software Architect
- Metodología y planificación del proyecto
- Decisiones al respecto del mismo, objetivos deseables y división de trabajo
El proyecto se ha dividido en dos partes cada una de las cuales es responsabilidad de uno de los dos proyectistas. Cada subproyecto incluye una parte compartida y una parte propia de documentación. Para distinguir aquellas secciones que son exclusivas de cada proyecto, se ha incluido un asterisco (*) en su cabecera.

La estructura de ambos documentos son similares por motivos estéticos y para facilitar la lectura a aquel que desee profundizar en los temas expuestos por ambos integrantes.

Así mismo, a lo largo de este documento encontrará cajas de fondo gris con el distintivo Decisión. Es la forma en la que indicamos un punto de reflexión, donde hemos tomado una serie de decisiones respecto el abanico de opciones que se nos presentaban. El estilo de dichas cajas será el siguiente:

<table>
<thead>
<tr>
<th>Decisión:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ejemplo de decisión tomada durante el proyecto</td>
</tr>
</tbody>
</table>

Por otra parte, también necesitaremos un distintivo para los trozos de código de cualquier lenguaje de programación utilizado. El estilo de este distintivo será:

class Ejemplo {
    public static void main(String[] args) {
        System.out.println("Hola PFCMDA!");
    }
}

Información de contacto con los autores:

- **Daniel Albert Sánchez**: hyde.x86@gmail.com
- **Ferran Basora Roca**: fcsonline@gmail.com
Índice general

1. Motivaciones *

2. Introducción
   2.1. Contexto histórico .................................................. 3
   2.2. El desarrollo de software ............................................ 4
       2.2.1. Rational Unified Process ....................................... 5
       2.2.2. Roles en el diseño de software ............................... 6
   2.3. Modelos de desarrollo .............................................. 7
   2.4. Integración de MDA con el desarrollo basado en RUP ............ 9
   2.5. Beneficios de desarrollo MDA ..................................... 9
   2.6. Visión del proyecto ............................................... 11

3. Objetivos del proyecto *

4. MDA
   4.1. ¿Qué es MDA? ......................................................... 15
   4.2. Evolución histórica de MDA ...................................... 17
   4.3. Niveles de abstracción MDA ...................................... 17
       4.3.1. CIM: Computation Independent Model ....................... 18
       4.3.2. PIM: Platform Independent Model ............................ 18
       4.3.3. PSM: Platform Specific Model ............................... 19
   4.4. Transformación de modelos .................................... 20
4.4.1. Transformación del CIM al PIM ........................................ 21
4.4.2. Transformación del PIM al PSM ...................................... 21
4.5. Herramientas para la transformación de modelos ...................... 23
  4.5.1. MTL: Model Transformation Language ............................. 23
  4.5.2. MOF: Meta-Object Facility ........................................ 23
  4.5.3. ATL: ATLAS Transformation Language .............................. 24
  4.5.4. Eclipse / Rational Software Architect ............................. 25
4.6. Particularidades de MDA en este proyecto ................................ 26
  4.6.1. Representación de los niveles MDA en Rational Software Architect 27
  4.6.2. Adaptando MDA para aprovechar al máximo Rational Software Architect .............. 28

5. IBM Rational Software Architect ........................................ 29
  5.1. Introducción .................................................................. 29
  5.2. ¿Qué es? .................................................................... 30
  5.3. El entorno .................................................................... 30
        5.3.1. Perspectivas ....................................................... 30
        5.3.2. El Model Explorer ............................................... 32
  5.4. Requisitos Hardware ..................................................... 32
  5.5. Características software ............................................... 33
        5.5.1. Plugins ............................................................... 33
        5.5.2. Pluglets ............................................................... 33
        5.5.3. Profiles ............................................................... 33
        5.5.4. Motor de modelado ............................................... 34
        5.5.5. Patrones ............................................................... 35
        5.5.6. Estereotipos ......................................................... 36
        5.5.7. El entorno de pruebas .......................................... 37

6. Transformaciones * .......................................................... 39
  6.1. Transformación de modelos ........................................... 39
        6.1.1. Elementos de una transformación en RSA ....................... 40
              6.1.1.1. Plugin.java ............................................... 40
              6.1.1.2. Transformation.java ..................................... 40
              6.1.1.3. TransformationProvider.java ............................. 41
ÍNDICE GENERAL

6.1.1.4. Rule.java ................................................. 42
6.1.2. Cómo funciona la transformación ................................................. 42
6.2. UML2 ......................................................... 44
   6.2.1. Cómo crear elementos en UML2 ................................................. 44
      6.2.1.1. Eligiendo el destino ................................................. 44
      6.2.1.2. Modelos ......................................................... 45
      6.2.1.3. Packages ......................................................... 45
      6.2.1.4. Clases ......................................................... 45
      6.2.1.5. Interfaces ......................................................... 46
      6.2.1.6. Atributos ......................................................... 46
      6.2.1.7. Operaciones ......................................................... 47
      6.2.1.8. Asociaciones ......................................................... 48
      6.2.1.9. Herencias ......................................................... 48
      6.2.1.10. Aplicar un estereotipo a un elemento ................................................. 48
6.3. Librería ModelUtility ......................................................... 49

7. Desarrollo de una transformación con RSA ................................................. 55
   7.1. Estudio del problema a modelar ................................................. 55
   7.2. Definición de los profiles ......................................................... 56
      7.2.1. Creación de un profile ................................................. 57
   7.3. Definición del plugin ......................................................... 59
      7.3.1. Creación del plugin ......................................................... 59
   7.4. Implementación de la transformación ................................................. 63
   7.5. Generación de código ......................................................... 63

8. Plugins desarrollados * ......................................................... 65
   8.1. Características que cumplen todas las transformaciones ................................................. 65
   8.2. Modelo Create, Read, Update and Delete ................................................. 67
      8.2.1. Introducción ......................................................... 67
      8.2.2. Estudio del caso ......................................................... 67
      8.2.3. El modelo PIM ......................................................... 69
      8.2.4. Profile PIM ......................................................... 70
      8.2.5. Plugin CrudPIMtoPSM para Struts ................................................. 71
ÍNDICE GENERAL

8.2.6. El modelo CIM ........................................ 72
8.2.7. Profile CIM ........................................ 74
8.2.8. Plugin CrudCIMtoPIM ................................ 74
8.3. Modelo de sistema de login ................................ 75
  8.3.1. Introducción ........................................ 75
  8.3.2. Estudio del caso ..................................... 75
  8.3.3. El modelo PIM ....................................... 75
  8.3.4. Profile PIM .......................................... 78
  8.3.5. Plugin LoginPIMtoPSM para Java Swing .......... 79
  8.3.6. El modelo CIM ....................................... 81
  8.3.7. Profile CIM .......................................... 81
  8.3.8. Plugin LoginCIMtoPIM .............................. 81
8.4. Pruebas ................................................... 82
  8.4.1. Pruebas unitarias .................................. 82
    8.4.1.1. Pruebas unitarias del CRUD ................... 82
    8.4.1.2. Pruebas unitarias del Login .................. 83
  8.4.2. Pruebas de extremo a extremo .................... 84

9. Manuales de uso de los plugins *
  9.1. Manual de uso del CRUD ............................... 85
    9.1.1. Creando el CIM. Generación del PIM ............. 85
    9.1.2. Creando el PIM. Generación del PSM Struts ........ 86
  9.2. Manual de uso del Login .............................. 87
    9.2.1. Creando el CIM. Generación del PIM ............. 87
    9.2.2. Creando el PIM. Generación del PSM en Java Swing. 88
  9.3. Ejemplos aplicados ................................... 89
    9.3.1. CRUD .............................................. 89
    9.3.2. Login ............................................. 94

10. Metodología y Planificación ............................... 99
  10.1. Metodología ......................................... 99
    10.1.1. Alcance del proyecto ........................... 99
    10.1.2. Enfoque del proyecto .......................... 101
ÍNDICE GENERAL

10.1.2.1. Ingeniería del Software ........................................ 102
10.1.2.2. Project Management ............................................ 102
10.1.2.3. Programación .................................................. 103
10.1.3. Metodología de desarrollo ...................................... 103
  10.1.3.1. Proceso iterativo ........................................... 103
  10.1.3.2. Milestones .................................................. 104
  10.1.3.3. Bottom-up .................................................. 104
10.1.4. Iteraciones ........................................................ 105
10.1.5. Recursos .......................................................... 106
10.1.6. Gestión de riesgos .............................................. 108
10.2. Planificación * .................................................... 109
  10.2.1. Diagrama de Gantt ............................................ 109
  10.2.2. Planificación temporal * .................................... 110

11. Conclusiones .......................................................... 113
  11.1. Conclusiones del proyecto * .................................... 113
  11.2. Beneficios empresariales de implantar MDA .................... 114
  11.3. Trabajo futuro * .................................................. 115

12. Glosario ................................................................. 121
¿Qué nos ha llevado aquí? Quizá esta debería ser la primera pregunta que debiéramos hacernos. ¿Por qué surge este proyecto? ¿Qué factor diferenciador puede aportar al mundo?

Por todos es sabido que el desarrollo de software es un proceso complejo en el que se requiere de muchos recursos (horas/persona).

Poco a poco van surgiendo herramientas, formas de analizar y diseñar software, metodologías de trabajo, etc. Que facilitan dicho proceso. Así por ejemplo, los lenguajes Orientados a Objetos supusieron una revolución en el mundo de la programación.

Con sólo pensarlo un instante, todos llegaríamos a la conclusión que la Orientación a Objetos ha facilitado de manera notoria el trabajo en equipo. Al aumentar la modularidad de la aplicación, se favorece la escalabilidad del mismo. Es más sencilla la división de responsabilidades.

Metodologías como Rational Unified Process, de la que hablaremos más adelante, potencian aún más dicha división de responsabilidades. Su objetivo es proporcionarnos una guía de cómo debemos trabajar, de manera que maximicemos la productividad y minimicemos el tiempo de desarrollo.

En los breves inicios de dicho proyecto estos eran los conocimientos asentados de los que disponíamos.

En Junio del 2006, Álex Ballarín Latre (de IBM) se puso en contacto con un profesor de esta Universidad proponiéndole un Proyecto Final de Carrera para el cual se necesitarían tres proyectistas.

La propuesta inicial consistía en adaptar una metodología denominada Model Driven Architecture\(^1\) en la herramienta de desarrollo de software IBM Rational Software Architect.

\(^1\)Más adelante se explicará en profundidad qué es MDA y qué mejoras proporciona
¿Por qué resulta interesante este objetivo? En el mercado ya existen otras herramientas que permiten seguir un desarrollo MDA\(^2\). Aún así, estas son herramientas muy específicas que únicamente permiten modelar mediante MDA. Rational Software Architect es un software de gran embargadura y con un amplio abanico de posibilidades. Dotar a RSA de la capacidad de desarrollar mediante MDA, supone un *valor añadido* para el producto. Esta capacidad le posicionaría por delante de sus competidores.

La viabilidad técnica es posible, tal y como nos justificó en su día Álex, y como veremos a lo largo de este documento.

La propuesta inicial fue la siguiente:

- Un proyectista se encargaría de definir y documentar el método
- Un proyectista se encargaría de diseñar el modelo y las transformaciones en RSA
- Un proyectista se encargaría de transformar el modelo de máxima concreción tecnológica a código fuente

Finalmente únicamente Ferran Basora y yo decidimos seguir adelante. Es por ello que nos repartimos de la siguiente manera el trabajo:

- Daniel: Encargado de diseñar el modelo y las transformaciones. Definición y documentación del método
- Ferran: Encargado de transformar el modelo de máxima concreción tecnológica a código fuente. Definición y documentación del método

Como comentario final, es interesante remarkar que este proyecto es una *prueba de concepto*. Supone un estudio sobre la viabilidad de adaptación de MDA a RSA, y explica la forma en que debería hacerse dicha adaptación. Entendemos que afirmar que en este proyecto se adaptará MDA a Rational Software Architect es incorrecto. Es un software de proporciones gigantescas, desarrollado por un numeroso equipo de expertos durante un período prolongado de tiempo. Es por ello que el producto final de este proyecto es *investigar y describir de forma detallada el método de adaptación de MDA a RSA*.

\(^2\)Véase AndroMDA: http://www.andromda.org/

© FIB
Introducción

En este capítulo se hará una breve introducción sobre la historia del desarrollo de software, una explicación sencilla de qué es MDA para entrar en materia, qué aporta de nuevo respecto lo que había antes y qué beneficios aporta.

Además se explicará en qué consiste Rational Unified Process y qué relación tiene con los conceptos estudiados en las asignaturas de la rama de Ingeniería del Software impartidas en la carrera.

2.1. Contexto histórico

La informática ha evolucionado de manera significativa desde mediados del siglo XX, cuando surgió una de las primeras computadoras de uso general: ENIAC. ENIAC ocupaba 167 m², y contenía 17,468 tubos de vacío. Con 1,500 conmutadores electromagnéticos y 7,200 interruptores, programarla requería semanas de preparación. Las entradas y salidas de los programas se describían en tarjetas perforadas.

Desde entonces hasta la actualidad, los ordenadores han evolucionado de manera frenética. Se ha reducido el tamaño de estos, el consumo, aumentando la potencia de cálculo, la memoria, etc.

Pero no sólo ha evolucionado el hardware. Los lenguajes de programación también han evolucionado de manera notoria. Desde los lenguajes sin orientación a objetos que generan código máquina, hasta los lenguajes con orientación a objetos interpretados. Sin duda la aparición de lenguajes como Java ha ayudado a programar mejor y más rápido. El programador no debe preocuparse de punteros, gestión de memoria dinámica, no tiene que recompilar el código en función de la arquitectura sobre la que va a correr, etc. La aparición de lenguajes orientados a objetos (como C++ o Java), representó un significativo avance en la programación de aplicaciones. Y tras ellos surgieron metodologías y
CAPÍTULO 2. INTRODUCCIÓN

lenguajes para hacer mejores programas. En concreto, la aparición de UML\(^1\).

Cada vez, realizar aplicaciones de gran embargadura resulta más sencillo. Día a día las necesidades empresariales son superiores, más ambiciosas y disponen de menos tiempo de desarrollo. Con la aparición de los IDE\(^2\) y herramientas de generación de código, el programador cada vez se ha de centrar más en el código inteligente. El resto del código es posible autogenerarlo.

Disponemos hoy en día de herramientas como Rational Software Architect que nos permiten modelar, definir relaciones de clases, diseñar, definir bases de datos, persistencia de clases, etc. Hasta el punto que permite autogenerar el código. De este modo, el trabajo de los programadores radica en realizar lo que se denomina código inteligente.

Entonces, ¿hacia dónde tiende el futuro? La situación utópica para todo Ingeniero de Software, sería que únicamente debiésemos especificar programas, y que el resto se autogenerara totalmente. Evidentemente, estamos lejos de este deseo, pero MDA \(^3\) pretende acercarse un poco más a ello. MDA es una aproximación al diseño de software propuesta por la OMG \(^4\).

2.2. El desarrollo de software

Como se comenta en el apartado anterior, el desarrollo de software ha evolucionado mucho desde sus orígenes. Desde la creación de aplicaciones monolíticas en tarjetas perforadas hasta las aplicaciones de Orientación a Objetos que se ejecutan en servidores distribuidos el salto ha sido espectacular.

Centrándonos en el desarrollo de software basado en Orientación a Objetos, existe un conjunto de metodologías y procesos que nos indican cómo debemos hacer las cosas para maximizar las probabilidades de éxito.

Otra manera de diferenciar el tipo de desarrollo de software es según la estrategia seguida. Podemos diferenciar dos grandes grupos: el orientado hacia objetivos en cascada y el que sigue un proceso iterativo.

El diseño de software en cascada consiste en definir una serie de etapas que se van ejecutando de manera consecutiva. La regla básica es que no se puede comenzar una etapa \(i+1\) hasta que no se ha completado la etapa \(i\).

La segunda opción consiste en realizar un proceso cíclico en el que en cada ciclo se perfecciona cada una de las etapas.

El problema que plantea el primero de ellos es que si en una etapa se detecta un error de una fase anterior, obliga a rehacer todo el proceso desde el punto modificado.

\(^1\)Unified Modelling Language
\(^2\)Integrated Development Environment
\(^3\)Model Driven Architecture
2.2. EL DESARROLLO DE SOFTWARE

El desarrollo iterativo minimiza el problema anterior, puesto que de forma temprana se realizan pruebas de los elementos diseñados e implementados y permite detectar y corregir errores de forma muy temprana.

**Decisión:**
En la elaboración de este proyecto hemos optado por la fórmula del proceso iterativo, tal y como se justifica en el capítulo Metodología y planificación (consultar 10).

2.2.1. Rational Unified Process

RUP es una guía sobre cómo realizar un sistema software de manera correcta, ayudándose de UML. RUP es un refinamiento de un concepto más genérico denominado UP\(^5\) llevado a cabo por la empresa Rational Software.

Algunos de los elementos que contempla RUP son:

- Asignación de tareas y responsabilidades
- Implementa un conjunto de *Best Practices* en Ingeniería de Software: desarrollo iterativo, administración de requisitos, arquitectura basada en componentes, control de cambios, modelado visual de software, verificación de la calidad del software, etc.

Una de las principales características de RUP es que es un proceso iterativo e incremental. Los elementos que conforman el producto final propiamente dicho se denominan *arte-factos*, que representan elementos tales como el código fuente, modelo de casos de uso, modelo de análisis, etc. Por otra parte, en RUP se denomina *rol* al papel que juega una persona en un momento determinado.

En la tabla 2.2.1 podemos ver una lista de los artefactos más importantes de RUP.

RUP se compone de varios ciclos de desarrollo, proporcionando al final de cada ciclo un producto final tangible. Al final de cada ciclo se realiza un *Assessment*, punto en el que se deben tomar decisiones para el futuro.

Los ciclos de RUP son:

- **Inicio**, donde se realiza un plan de fases, se identifican los casos de uso principales y los riesgos
- **Elaboración**, etapa en la que se construye un plan de proyecto, se contemplan los casos de uso y se eliminan los riesgos

\(^5\)Unified Process
<table>
<thead>
<tr>
<th>Artefecto</th>
<th>Descripción</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Case</td>
<td>Contiene el diagrama de casos de uso que representa el modelo de negocio (sin tener en cuenta solución informática)</td>
</tr>
<tr>
<td>Use Case</td>
<td>Contiene el diagrama de casos de uso del sistema, de la solución tecnológica</td>
</tr>
<tr>
<td>Analysis Model</td>
<td>Contiene el modelo de análisis del sistema. Además contiene los diagramas de secuencia y estados</td>
</tr>
<tr>
<td>Design Model</td>
<td>Contiene el modelo de diseño del sistema. Incorpora patrones, diagramas de secuencia y estados</td>
</tr>
<tr>
<td>Implementation Model</td>
<td>Artefecto que contiene los ficheros físicos con la solución propuesta</td>
</tr>
<tr>
<td>Vision</td>
<td>Documento que describe qué es y qué no es el proyecto de forma concreta</td>
</tr>
<tr>
<td>Software Development Plan</td>
<td>Artefecto que describe cómo se llevará a cabo el desarrollo del proyecto, qué hacer si se materializan riesgos, etc.</td>
</tr>
<tr>
<td>Test Plan</td>
<td>Plan de pruebas a realizar para garantizar la calidad del producto</td>
</tr>
<tr>
<td>Supplementary Specifications</td>
<td>Artefecto en el que se describen los requisitos no funcionales del sistema</td>
</tr>
</tbody>
</table>

Cuadro 2.2.1: Lista de los artefactos básicos utilizados en RUP

- **Construcción**, donde se concreta la elaboración para crear un producto operativo y eficiente
- **Transición**, etapa en la que se implanta el producto en el cliente y se forma a los usuarios

**Decisión:**

Algunos de los *best-practices* que dictamina el proceso RUP se han llevado a cabo en la elaboración de este proyecto, tal y como se explica en el capítulo Metodología y Planificación (ver 10.1). Hay secciones de esta memoria que contienen apartados que se aconseja incluir en los artefactos de RUP. Así por ejemplo, análisis de riesgos, visión del proyecto, planes de contingencia, plan de pruebas, etc.

### 2.2.2. Roles en el diseño de software

Mostramos a continuación una tabla con los roles y funciones de cada uno de los integrantes de un proyecto de desarrollo de software.
2.3. MODELOS DE DESARROLLO

Figura 2.2.1: Los ciclos de desarrollo según RUP, en función del tiempo

<table>
<thead>
<tr>
<th>Rol</th>
<th>Función</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jefe de proyecto</td>
<td>Es el encargado de definir el alcance del proyecto, relación con el cliente, determinar requisitos funcionales y no funcionales, gestión de cuentas, etc.</td>
</tr>
<tr>
<td>Analista</td>
<td>El analista es el encargado de describir las funcionalidades que debe resolver la solución software</td>
</tr>
<tr>
<td>Diseñador</td>
<td>Es el encargado de decidir cómo se llevará a cabo la solución propuesta por el analista, especificando qué tecnología será utilizada</td>
</tr>
<tr>
<td>Programador</td>
<td>Un programador se encargará de implementar el código que corresponda con la solución informática propuesta por el diseñador</td>
</tr>
</tbody>
</table>

Cuadro 2.2.2: Roles en el desarrollo de software

2.3. Modelos de desarrollo

Aquí tenemos un esquema del ciclo de desarrollo tradicional. Como se puede observar, es un proceso cíclico que empieza en el análisis de requisitos del cliente. Una vez están identificados es necesario analizar el problema y describir una solución al mismo. Detectar qué elementos de software anterior se pueden reutilizar y cuáles no.

Figura 2.3.1: El ciclo de desarrollo tradicional

Las posteriores etapas corresponden al análisis, diseño, construcción e implantación del
CAPÍTULO 2. INTRODUCCIÓN

software.

Cuando hacemos desarrollo de software mediante una metodología UP\(^6\) seguimos los pasos indicados anteriormente. Hay que notar que es un proceso que deben seguir los arquitectos y diseñadores, la metodología simplemente les da indicaciones de cómo hacerlo de una manera ordenada y eficiente. Un analista daría la respuesta de qué debe solucionar el sistema informático. Un diseñador estudiaría la solución propuesta por el analista y buscaría una respuesta eficiente al cómo se va a solucionar.

Utilizando MDA se pretende facilitar el proceso anterior. Gracias al desarrollo MDA, conocido como MDD\(^7\), se pretende asistir al usuario tanto como sea posible, para llegar a obtener una solución software de forma más rápida y eficiente. Si conseguimos que el analista defina el problema de manera concreta independiente de la tecnología que habrá debajo (PIM \(^8\)), asistiéndolo podremos ayudarle a generar un modelo de entrada para el diseñador. Del mismo modo podemos ayudar al programador a producir el software final (eligiendo la tecnología más adecuada). Y no sólo eso: si la solución al problema está definida independientemente de la tecnología, seremos capaces de adaptar la solución al cambio tecnológico realizando pocas modificaciones.

Por otra parte, el objetivo de MDA es asistir al MDE\(^9\). MDE se refiere al uso sistemático de modelos como artefactos primarios de la ingeniería a través del ciclo de vida de la ingeniería. De este modo conseguimos librar a las personas de trabajo, puesto que se puede programar más rápido y con menos errores.

Imaginemos un similitud con la arquitectura. Un arquitecto podría hacer los planos de una casa, teniendo en cuenta únicamente los habitáculos, puertas y ventanas que hubiera en cada una de ellas. Todo ello sin considerar cómo debe ser la instalación eléctrica, ni qué tipo de materiales va a haber en cada habitación, ni los enchufes, etc. Ahora el arquitecto podría decidir qué función va a tener cada una de las habitaciones: un lavabo, una cocina, etc. Esto sería el similitud de PIM\(^10\), puesto que está considerando una solución al diseño de un hogar sin tener en cuenta el cómo se construirá. Únicamente decide cuántos lavabos tendrá, cuál será la comunicación entre habitaciones, etc. Con los planos tal y como los tenemos ahora mismo, el lector estará de acuerdo que hay infinidad de implementaciones de la casa: con parquet o no, varias distribuciones del inodoro, ducha o bañera, etc. ¿Y si tuviéramos un software que nos permitiera diseñar e implementar, al estilo MDA, una casa? Etiquetaríamos una habitación como lavabo, y un proceso asistido nos preguntaría si queremos ducha o bañera, dónde situaríamos el inodoro, etc. De tal manera que ayudaríamos de manera considerable al arquitecto.

Es más, siguiendo con el hipotético caso del software de arquitectura, con el mismo diseño PIM que representa una casa podríamos realizar cambios en nuestro hogar sin tener

---

\(^6\)Unified Process
\(^7\)Model-Driven Development, se refiere al desarrollo orientado al modelo
\(^8\)Platform Independent Model
\(^9\)Model-Driven Engineering
\(^10\)Más adelante trataremos en profundidad los niveles de MDA
que rediseñar todo el modelo.

Figura 2.3.2: Transformaciones de un arquitecto de infraestructuras haciendo un símil con MDA

2.4. Integración de MDA con el desarrollo basado en RUP

MDA se integra perfectamente con el desarrollo basado en Rational Unified Process. Como más adelante veremos y justificaremos, podemos identificar cada uno de los niveles que conforma un modelado del paradigma MDA con algún artefacto de RUP.

Utilizando un desarrollo basado en el paradigma MDA somos capaces de realizar soluciones software de manera mucho más rápida que de la forma tradicional. Esto significa que si identificamos el artefacto Use Case de RUP con uno de los modelos que representan MDA, e identificamos el artefacto Analysis Model con otro modelo que representa MDA, la transformación de un modelo a otro nos facilita enormemente el trabajo.

En la figura 2.4.1 podemos ver qué nivel de MDA se corresponde con qué artefacto de RUP.

2.5. Beneficios de desarrollo MDA

A continuación detallaremos una lista de beneficios que presenta el desarrollo MDA.

1. **Aumenta la productividad**: El desarrollo mediante Model-driven development supone un aumento de productividad, puesto que permite generar código y otros artefactos a partir de los modelos.

2. **Mantenibilidad**: Gracias al desarrollo orientado al modelo, y al independizar totalmente la solución tecnológica, se aumenta la mantenibilidad de las aplicaciones. La migración a nuevas tecnologías es mucho más sencilla.

3. **Adaptabilidad**: Gracias al desarrollo MDD, es mucho más fácil añadir o modificar procesos de negocio. El hecho de añadir un proceso de negocio implica sólo diseñar
esa nueva funcionalidad. El código común a otros componentes lo implementan las transformaciones de modelo.

4. **Consistencia**: El código generado está libre de errores. Al autogenerar código libramos al programador de trabajo y el producto es más consistente desde el principio.

5. **Repetición**: El hecho de aplicar transformaciones entre modelos ayuda a que las organizaciones reduzcan el factor sorpresa a la hora de desarrollar software. Con la experiencia de algo aprendido es más fácil evitar la materialización de riesgos.

6. **Claridad**: El hecho de diseñar modelos independiza de estos factores irrelevantes. De este modo la comprensión de una solución mediante un modelo ayuda a entender la solución propuesta por su carácter conciso.

7. **Retrasar la decisión tecnológica al final**: Diseñando de este modo, permite decidir una solución a un problema retrasando hasta el último momento la decisión tecnológica. De este modo, es posible determinar cuál será la mejor solución dadas las características.
2.6. Visión del proyecto

En este proyecto se pretende investigar y proponer una solución sobre cómo debería llevarse a cabo una adaptación de la metodología MDA en IBM Rational Software Architect.

Puesto que el trabajo que se plantea no es una receta que debamos seguir paso a paso, sino que es un proyecto de investigación en el que en determinados puntos se plantean diversas alternativas, debemos proponer un método de actuación en el momento en que debamos tomar una decisión y justificarla adecuadamente.

En cada punto en el que se pueda avanzar por diversos caminos se analizarán las alternativas en busca de la mejor de ellas. Analizaremos los parámetros que consideremos adecuados en función del resultado deseado. Tendremos en cuenta el coste temporal que puede repercutir cada una de las opciones.

El principal objetivo del proyecto es el de encontrar elementos susceptibles de ser modelizados con el paradigma MDA, estudiar la forma en que podría adaptarse a RSA, documentarlo e implementarlo. Esto formará parte de un proceso iterativo (consultar el capítulo 10.1.3) en el que en cada iteración se madurará el proceso de implantación. Al final del proceso, obtendremos un método sólido de cómo un equipo de desarrolladores debería actuar para realizar una implementación del paradigma MDA de forma sencilla.

El objetivo final del proyecto será desarrollar un conjunto de plugins que se puedan adosar a la plataforma IBM Rational Software Architect, que demuestren que se puede implementar el paradigma MDA en dicha herramienta. Además se pretende observar los beneficios aportados: la reutilización de código, independencia tecnológica del diseño, etc.
Objetivos del proyecto *

En este capítulo trataremos los objetivos que se pretenden conseguir con la elaboración de este proyecto.

Como se ha visto en el capítulo anterior (consultar apartado 2.5) MDA proporciona gran ventaja respecto el desarrollo tradicional.

Si observamos el mercado, hay varias aplicaciones comerciales que implementan MDA. Como ejemplo de ellas tenemos AndroMDA \(^1\) y OpenMDX \(^2\).

Aún así, el principal objetivo de este proyecto es estudiar el método de creación de plugins \(^3\) para IBM Rational Software Architect quedando en un plano secundario el producto final. De este modo, el resultado de la elaboración de este trabajo no es más que un \emph{método}, un manual de cómo se deben hacer las cosas para que alguien pueda trabajar en ello.

\begin{tabular}{|l|}
\hline
\textbf{Decisión:} \\
Es importante destacar que IBM Rational Software Architect es una aplicación increíblemente grande y que ha requerido de muchas horas de trabajo de un grupo elevado de personas. El principal objetivo de este proyecto no es el de dotar a RSA de la capacidad de incorporar MDA; el objetivo consiste en realizar una prueba piloto de cómo debería hacerse dicha incorporación. \\
\hline
\end{tabular}

Por otra parte, se proporcionará como producto final un conjunto de plugins que permitan a un usuario de Rational Software Architect llevar a cabo un proceso asistido MDA.

Enumeramos a continuación una lista de los principales objetivos del proyecto, en orden cronológico:

\begin{itemize}
\item \(^1\)http://www.andromda.org/
\item \(^2\)http://www.openmdx.org/
\item \(^3\)Módulos de una aplicación
\end{itemize}
1. **Análisis de la herramienta de desarrollo Rational Software Architect a nivel de usuario** y realización de tutoriales de aprendizaje. Es imprescindible conocer bien la herramienta de modelado para poder tener éxito en la adaptación de MDA a la misma. Conociendo en profundidad cómo debe trabajar un usuario RSA, será más fácil proponer una herramienta usable.

2. **Análisis de la herramienta de desarrollo Rational Software Architect a nivel de desarrollador.** Realización de tutoriales de aprendizaje de cómo crear plugins, realizar transformaciones, etc. Cumpliendo este objetivo, conseguiremos conocer en profundidad las posibilidades que nos ofrece RSA y de entre ellas escoger la que mejor se adapte a MDA.

3. **Investigación de la tecnología MDA y su estándar definido por la OMG.** Éste es un objetivo esencial, que complementado con los anteriores permitirá identificar los elementos adecuados para representar un desarrollo MDD en RSA. Además resulta un objetivo esencial por definición, puesto que sin conocer en profundidad la metodología no es posible realizar nada.

4. **Investigación de utilización de la librería UML2**, que servirá para realizar transformaciones modelo-modelo.

5. **Estudio del método de adaptación de los niveles MDA en Rational Software Architect.** Identificar de entre las posibilidades de RSA cómo representar cada nivel MDA. (Consultar capítulo 4.3).

6. **Diseño del método de modelado.** Proporcionar una receta para que cualquier persona que haya leído este documento, pueda crear sus propios modelos del paradigma MDA.

7. **Proceso iterativo de desarrollo de plugins**, tal y como se describe en el capítulo 10.1 en el que aprovechando el método que se explicará más adelante de bottom-up, se garantiza un trabajo orientado a objetivos. Esto proporcionará una mejora de eficiencia, puesto que el trabajo está condicionado a resultados.

8. **Documentación del propio proceso** con el fin de que un usuario pueda realizar aplicaciones gracias al material proporcionado.
En este capítulo se tratará de forma detallada qué es MDA y cuál ha sido su evolución histórica.
Así mismo trataremos en detalle qué opciones hay en el mercado y por qué hemos tomado determinadas decisiones. Veremos la forma de adaptar MDA a RSA, teniendo en cuenta las ventajas y limitaciones de ambos.
Hemos decidido que es interesante documentar este capítulo conjuntamente, puesto que así establecemos una visión estandarizada de cómo enfocamos el paradigma MDA en este proyecto. Como se han tomado particularidades respecto lo que determina el estándar definido por el OMG, es conveniente describir dichas particularidades con la suficiente claridad.

4.1. ¿Qué es MDA?

MDA es una metodología de diseño de software propuesta por el OMG\(^1\) en el año 2001. MDA consta de tres niveles de abstracción, que describiremos a lo largo de este capítulo. Antes de explicar en detalle en qué se basa MDA, presentaremos la nomenclatura básica para familiarizar al lector:

- Al desarrollo orientado en el modelo se le denomina MDD\(^2\).
- A la ingeniería de procesos basándose en MDA se le denomina MDE\(^3\).
- La definición de los niveles se hace en un lenguaje DSL\(^4\). DSL engloba lenguajes que nos sirven para definir los niveles.

---
\(^1\)Object Management Group
\(^2\)Model-Driven Development
\(^3\)Model-Driven Engineering
\(^4\)Domain Specific Language
- Llamamos PDM\textsuperscript{5} al metamodelo que describe cómo debe ser una transformación de un modelo a otro de MDA.

- QVT\textsuperscript{6} es un estándar definido por el OMG para describir cómo deben ser las transformaciones entre niveles.

En la sección 2.3 del capítulo Introducción, justificamos la principal mejora por la cual surge Model Driven Architecture. Como mencionan en la web del OMG, MDA es la arquitectura a escoger en un mundo que cambia constantemente.

El mundo de la informática evoluciona de manera frenética. Constantemente surgen nuevos lenguajes, arquitecturas, sistemas y formas de trabajar. Al igual que en otros sectores de la sociedad, muchas veces el mercado se rige por aquella tecnología que en el momento está de moda.

El principal beneficio que proporciona MDA que no disponíamos anteriormente es la versatilidad al cambio de tecnología. Porque teniendo un problema correctamente definido -sin tener en cuenta la tecnología en la que finalmente esté implementado- la capacidad de reacción al cambio gracias a un proceso MDD es tremendamente sencillo.

Según la definición estricta de OMG, gracias a MDA y combinándolo con otros estándares definidos por la misma organización, podemos generar elementos tales como Web-Services, .NET, Corba, J2EE, etc.

![Figura 4.1.1: Modelo conceptual de MDA](image)

Aún así, como veremos al final de este capítulo, nuestra visión de MDA ha sido ligeramente rebajada respecto la definición del estándar, puesto que así ha sido necesario para poder integrarlo en Rational Software Architect.

\textsuperscript{5}Platform Definition Model
\textsuperscript{6}Queries/Views/Transformations
4.2. Evolución histórica de MDA

MDA surge como propuesta del OMG. El OMG (Object Management Group) es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI y CORBA. Es una organización no lucrativa que promueve el uso de tecnología orientada a objetos mediante guías y especificaciones para tecnologías orientadas a objetos. El grupo está formado por cientos de compañías y organizaciones de software de entre las que destacan:

- Hewlett-Packard
- IBM
- Sun Microsystems
- Apple Computer


En el año 1997 el OMG adopta UML y MOF\(^7\) como técnicas de modelado. Como bien es sabido, un modelo UML puede ejecutarse sobre cualquier plataforma. Aún así, con el tiempo se determinó que el hecho de modelar sin tener en cuenta la plataforma sobre la que va a ser ejecutado el producto final es la clave para la estabilidad y el retorno de inversión.

La aproximación MDA tal y como la conocemos podemos bautizarla en el año 2001. Algunos de los conceptos clave de la aproximación MDA ya fueron enunciados en el método propuesto por Sally Shlaer y Stephen Mellor (Shlaer-Mellor\(^8\)), a finales de los 80.

4.3. Niveles de abstracción MDA

MDA se descompone en tres grandes niveles:

- CIM: Computation Independent Model
- PIM: Platform Independent Model
- PSM: Platform Specific Model

\(^7\)MetaObject Facility - http://www.omg.org/mof/

\(^8\)Para más información consultar el libro *Object-Oriented Systems Analysis: Modelling the World in Data* (Sally Shlaer y Stephen Mellor)
4.3.1. CIM: Computation Independent Model

El primer nivel, de mayor abstracción de MDA se denomina CIM. Un modelo CIM es una primera aproximación a la solución del problema, totalmente abstracta: todavía no se piensa en una tecnología concreta. En la metodología RUP sería el símil con el Use Case. Contiene una lista de requisitos que debe cumplir la solución software.

El objetivo de una definición de un problema en este nivel, es el de esclarecer los detalles del problema que queremos modelar. De esta manera, un cliente podría observar si la solución propuesta responde a sus necesidades. Por otra parte, para el desarrollador es mucho más fácil pasar de este nivel a una solución informática. Así pues, el CIM sirve de puente entre el cliente y el desarrollador. Además, favorece la distribución del trabajo, priorización de tareas y sirve como elemento estructural.

Considerando el clásico ejemplo del Terminal de Punto de Venta, en este nivel de abstracción consideraríamos:

1. El cliente inicia la secuencia de acciones, presentando los productos que desea comprar al empleado
2. El empleado suma la cantidad que debe abonar el cliente
3. El empleado indica la cantidad al cliente, y le pregunta sobre el método de pago
4. El cliente elige pagar al contado o con tarjeta de crédito
5. El empleado cobra al cliente

En la solución anterior, no se tiene en cuenta la intervención de ninguna tecnología concreta, sólo se detalla una solución genérica de cómo cobrar productos a un cliente en cualquier establecimiento. Es este el nivel de abstracción que se detalla en el nivel CIM. Detalla aquello que se debe realizar.

4.3.2. PIM: Platform Independent Model

El siguiente nivel que especifica MDA es el nivel PIM. Para llevar a cabo un proceso de desarrollo mediante MDA es necesario que se defina un modelo PIM. Un modelo PIM es una especificación del problema sin tener en cuenta ningún detalle tecnológico, aunque conceptualmente ya se piensa en una solución informática. Un ejemplo de esta independencia tecnológica: HTML define cómo debe ser una página web sin tener en cuenta qué sistema operativo y qué navegador va a interpretarlo.

Cuando alguien pretende hacer un desarrollo utilizando MDA deberá especificar el problema sin tener en cuenta si utilizará un MOO9, si implementará la lógica en un sistema gestor de base de datos relacional, etc.

9Modelo Orientado a Objetos
4.3. NIVELES DE ABSTRACCIÓN MDA

Volviendo al ejemplo del Terminal de Punto de Venta, en este nivel de abstracción consideraríamos:

1. El cliente inicia la secuencia de acciones, presentando los productos que desea comprar al empleado

2. El empleado indica al terminal de punto de venta que un cliente quiere realizar una compra, e introduciría los productos que quiere adquirir

3. El TPV descontaría las unidades que adquiere el cliente del sistema central, a fin de mantener consistentes los inventarios

4. Una vez introducidos todos los productos, el empleado pregunta al cliente sobre el método de pago

5. El cliente paga, y el TPV guarda la transacción a fin de ayudar a mantener las cuentas del establecimiento

Nuevamente indicamos la correspondencia con artefactos RUP. En esta ocasión, el nivel PIM se corresponde con el Analysis Model de RUP.

4.3.3. PSM: Platform Specific Model

El último de los niveles de MDA es el PSM. PSM representa un modelo basándose en una tecnología concreta, en una plataforma concreta. Entendemos que puede haber varios modelos PSM que se correspondan con el mismo PIM. En el diseño de software basándose en un proceso UP¹⁰ esta distinción entre PIM y PSM no es tan clara, puesto que en fases tempranas ya se toman decisiones al respecto de la tecnología. Cuando se está haciendo el análisis del problema, el analista toma decisiones que influirán claramente en la tecnología utilizada.

Así podríamos afirmar que en un modelo PSM se empiezan a contemplar aspectos de la tecnología qué dará soporte a nuestra aplicación: OO, J2EE, .NET, servlets, hibernate, etc.

Una vez mas, indicamos el nivel de abstracción en el ejemplo del TPV:

1. El cliente inicia la secuencia de acciones, presentando los productos que desea comprar el empleado

2. El empleado indica al terminal de punto de venta que un cliente quiere realizar una compra

3. El empleado utiliza el lector de código de barras para introducir los productos

¹⁰Unified Process
4. El TPV descuenta las unidades de una base de datos Oracle centralizada a través de un sistema de red

5. Una vez introducidos todos los productos, el empleado pregunta al cliente sobre el método de pago

6. El cliente paga, y el TPV guarda la transacción en el ERP\textsuperscript{11} SAP para mantener las cuentas del establecimiento

El artefacto RUP con el que se corresponde el nivel PSM es el Design Model.

**Figura 4.3.1:** Separación de niveles MDA dependiendo de su dependencia tecnológica

---

### 4.4. Transformación de modelos

El proceso de transformación de modelos consiste en pasar de un modelo A a un modelo B ayudándose de los metamodelos mA y mB. Se dice que es una transformación endogénea si mA = mB. En caso contrario se le denomina transformación exogénea. Para representar los modelos es necesario disponer de un DSL\textsuperscript{12}.

En la figura 4.4.1 podemos contemplar los diferentes niveles por los que pasa el desarrollo de software tradicional. Iniciado en el modelo de casos de uso y finalizado en la generación de código, se deben crear un conjunto de artefactos que definan el sistema a modelar.

En esta sección trataremos desde una visión abstracta cómo se realizan las transformaciones entre modelos de los niveles MDA.

\textsuperscript{11}Enterprise Resource Planning

\textsuperscript{12}En nuestro caso utilizaremos UML como DSL para nuestras transformaciones
### 4.4. TRANSFORMACIÓN DE MODELOS

#### 4.4.1. Transformación del CIM al PIM

El proceso de transformación entre el modelo CIM al modelo PIM es complejo. Es tal el nivel de abstracción en el primero que gran parte de la información debe estar incluida en la lógica de la transformación. Actualmente es uno de los temas en los que más se está investigando sobre MDA.

Si el lector está familiarizado con la metodología RUP, el simbol de pasar de un nivel CIM a otro PIM es el hecho de generar elementos desde el Use Case al Analysis Model. La información que se proporciona en el primero de ambos casos (CIM y Use Case) es tan abstracta que resulta sumamente complejo transformarlo al siguiente nivel.

En las transformaciones realizadas en este proyecto se ha conseguido realizar transformaciones del CIM al PIM a base de relajar el nivel de abstracción.

#### 4.4.2. Transformación del PIM al PSM

Como se ha comentado anteriormente la relación entre PIM y PSMs es 1 a N. Esto significa que, a priori, no es un problema determinista. En algunos casos puede suceder que una solución tecnológica sea óptima, pero sería necesario decidir qué criterios determinan que una solución sea mejor que otra: eficiencia, minimización de usos de la red, delegando la lógica al SGBD...
¿Cómo se lleva a cabo la transformación de un nivel a otro? Para ello nos ayudamos de un PDM. Un PDM es un metamodelo que representa la definición de un modelo. Un usuario que quiera pasar de un modelo PIM a otro PSM, deberá proporcionar información adicional al primero para poder generar correctamente el segundo. Para ello es necesario un proceso documentado, o una GUI\textsuperscript{13} lo suficientemente intuitiva como para llevar a cabo el proceso satisfactoriamente.

\textsuperscript{13}Graphics User Interface
4.5. Herramientas para la transformación de modelos

4.5.1. MTL: Model Transformation Language

El OMG ha definido un estándar de transformación de modelos denominado MOF/QVT\(^{14}\), porque disponer de un sistema de transformación de modelos es una parte esencial de la especificación MDA.

Existe en el mercado diferentes lenguajes que cumplen en parte este estándar de OMG. Algunos de estos lenguajes son ATL\(^{15}\) o VIATRA\(^{16}\). En el estándar MOF/QVT se define una forma genérica de transformación de modelos basado en metamodelos MOF\(^{17}\).

4.5.2. MOF: Meta-Object Facility

Para definir metamodelos que nos servirán para una posterior transformación de modelos, OMG ha definido un estándar llamado MOF.

MOF es un lenguaje para definir lenguajes de modelado. Permite a los usuarios definir totalmente nuevos lenguajes a partir de metamodelos.

La alineación del metamodelo UML2.0 con el metamodelo MOF simplifica la interoperabilidad cruzada entre herramientas.

No lo veremos más en detalle, puesto que no tiene mayor relevancia.

\(^{14}\)Queries/Views/Transformations
\(^{15}\)ATLAS Transformation Language desarrollado por INRIA
\(^{16}\)Visual Automated model TRAnformations
\(^{17}\)Meta-Object Facility
4.5.3. ATL: ATLAS Transformation Language

ATL es un lenguaje desarrollado en INRIA\textsuperscript{18} que implementa el estándar QVT comentado anteriormente. Este lenguaje, como especifica el estándar, es capaz de transformar modelos. ATL puede recibir como fuente formatos como Ecore, KM3, EMOF, etc.

Veremos un ejemplo de cómo funciona una transformación ATL. No tiene gran relevancia con la solución aportada en éste proyecto, de modo que si el lector lo cree conveniente puede pasar a la siguiente sección.

En este ejemplo haremos una transformación de un modelo de clases a un modelo relacional.

Si queremos transformar un modelo que representa una conjunto de clases, el primer paso de todos sería obtener el metamodelo que representa a cualquier diagrama de clases.

En un metamodelo de clases podemos observar que hay la definición de 	extit{Class}, la definición de 	extit{Attribute} y la definición de 	extit{DataType}. Todos ellos heredan de una clase que les proporciona a todos un atributo de texto que será un identificador. También vemos que una 	extit{Class} puede tener un conjunto ordenado de 	extit{Attribute}, que pueden ser multi- evaluados o no.

Entonces, una vez obtenido el metamodelo origen 	extit{MMa}, necesitamos el metamodelo 	extit{MMb} que representa al modelo de destino. En nuestro caso será una base de datos relacional. En la figura 4.5.3 lo podemos observar.

En este metamodelo podemos observar que está compuesto por 	extit{Table}, y ésta por 	extit{Column}.

\textsuperscript{18}Institut national de recherche en informatique et en automatique
de un cierto *Type*. En este caso todos también heredan de una clase superior que les proporciona un identificador de texto.

El último paso es determinar un mapeo entre el metamodelo origen y el destino, que en este ejemplo se puede deducir rápidamente, y decidir las correspondientes reglas que lo lleven a cabo.

Cuándo tenemos determinado el mapeo, sólo falta escribir en lenguaje ATL dicha transformación. Un programa ATL lo forman una cabecera que contempla los modelos de entrada y de salida, un conjunto de funciones llamadas helpers y un conjunto de reglas que determinan los cambios que ha de sufrir el modelo *Ma* para obtener el modelo *Mb*.

### 4.5.4. Eclipse / Rational Software Architect

Dentro del abanico de herramientas de transformación de modelos también existe la plataforma de desarrollo Rational Software Architect. Esta potente herramienta tiene un motor de transformación de modelos integrado de forma nativa.

Igual que el resto de herramientas de transformación de modelos, RSA utiliza internamente un DSL para trabajar. El DSL de RSA es UML, que por razones históricas (consultar 5.1) se integra perfectamente con el entorno.
RSA permite hacer transformaciones de varios tipos:

- De modelo a modelo
- De modelo a texto

La transformación de modelos está integrada de forma nativa con Rational Software Architect. De hecho podemos desarrollar una transformación y ejecutarla en un entorno de pruebas reducido. La primera instancia de Rational Software Architect nos permite hacer debugging de la transformación que se ejecuta en el entorno de pruebas.

<table>
<thead>
<tr>
<th>Decisión:</th>
</tr>
</thead>
<tbody>
<tr>
<td>El fabricante (IBM) recomienda en una lista de best-practices realizar transformaciones utilizando su sistema de transformación de modelos. La mayor ventaja es que de forma nativa ya están incluidas. Es por ello que hemos decidido trabajar con las transformaciones de RSA, en vez del resto de herramientas de transformación de modelos que hemos enumerado. Además el hecho de trabajar con las transformaciones RSA, permite que los modelos transformados se integren perfectamente con el resto de funcionalidades del software.</td>
</tr>
</tbody>
</table>

La transformación de modelos proporcionada por Rational Software Architect se verá en detalle en posteriores capítulos, puesto que es el elemento con el que hemos trabajado.

4.6. Particularidades de MDA en este proyecto

El principal objetivo de este proyecto es proporcionar a Rational Software Architect de la capacidad de llevar a cabo un MDD. Para ello hemos tomado las siguientes decisiones para obtener un mejor resultado:

- Aprovechar al máximo las funcionalidades proporcionadas por RSA
- Creación de plugins instalables con las transformaciones MDA
- Desarrollo sobre Rational Software Architect 6.0.1.1 (actualizado hasta la Interim Fix 003)
- Utilización de la librería UML2 (versión 1.0.3)
- Utilización de la librería JET 1.0
4.6. PARTICULARIDADES DE MDA EN ESTE PROYECTO

Por otra parte la propia herramienta nos marca una serie de restricciones. Rational Software Architect está diseñado para trabajar con modelos UML. Es por ello que los modelos CIM, PIM y PSM deben estar representados en UML. Además, RSA está inicialmente pensado para generar código Java y C++, por lo que la decisión de crear una transformación a cualquier otro lenguaje puede suponer algún problema adicional. De este modo, consideramos que una de nuestras prioridades será crear plugins para tecnología J2EE, puesto que tanto el estándar como la herramienta lo permiten.

La mayor dificultad a nivel decisional de nuestro caso de estudio, es establecer una correspondencia entre los niveles MDA y las funcionalidades proporcionadas por RSA que respeten fielmente a la definición de MDA proporcionada por el OMG. En el siguiente apartado hablaremos sobre ello.

4.6.1. Representación de los niveles MDA en Rational Software Architect

Según el OMG un modelo CIM es la descripción de la solución a un problema independientemente de la tecnología. Puesto que un diagrama de clases se podría considerar una solución tecnológica, creemos que el mejor elemento que nos proporciona RSA para representar el modelo CIM es un diagrama de casos de uso. Un modelo de casos de uso nos permite definir los requisitos necesarios para solventar un problema sin tener en cuenta el cómo.

Una vez pasamos al modelo PIM, ya tenemos claro que utilizaremos una solución informática para solventar el problema dado. La OMG define un modelo PIM como la descripción de una solución técnica, sin especificar qué tecnología informática será utilizada para solventarlo. Este es el punto más delicado al que nos hemos enfrentado. Porque ¿dónde está realmente la frontera entre independencia tecnológica y dependencia tecnológica?

Según la definición teórica estricta del OMG, el hecho de representar el nivel PIM en un diagrama UML es incorrecto. Cuando diseñamos un diagrama UML, estamos considerando que la solución será un producto desarrollado en Orientación a Objetos. Pero eso no tiene por qué ser así: también existen soluciones monolíticas, soluciones implementadas en un SGBD, etc.

<table>
<thead>
<tr>
<th>Decisión:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Siendo conscientes de ello, hemos decidido que dadas nuestras necesidades, un modelo UML es el mejor método para representar el modelo PIM. Consideramos que un modelo de casos de uso es demasiado abstracto como para representar toda la información que necesitamos en este nivel.</td>
</tr>
</tbody>
</table>

Finalmente, representaremos el modelo PSM en un diagrama UML.

Mostramos a continuación la correspondencia con RUP.
- CIM: Se corresponde con el Use Case de RUP
- PIM: Se corresponde con el Analysis Model de RUP
- PSM: Se corresponde con el Design Model de RUP

4.6.2. Adaptando MDA para aprovechar al máximo Rational Software Architect

Debemos tener muy presente que Rational Software Architect es una herramienta de diseño de software muy completa y funcional. Las posibilidades que tiene un desarrollador para generar software son extraordinarias. Es por ello que consideramos que sería muy beneficioso que pudiéramos aprovechar al máximo las posibilidades de RSA en nuestra adaptación para MDA.

<table>
<thead>
<tr>
<th>Decisión:</th>
</tr>
</thead>
<tbody>
<tr>
<td>De este modo hemos tomado la decisión de no restringir nuestro proceso a las transformaciones que realicemos. Nuestro objetivo es que alguien que se aproveche de nuestro proceso, pueda continuar más allá y generar una solución informática completamente funcional.</td>
</tr>
</tbody>
</table>

De esta manera, si alguien quiere desarrollar una aplicación Struts que contenga un CRUD se puede aprovechar de nuestro proceso asistido. En el modelo que representa el nivel CIM, puede modelar el resto de casos de uso del sistema. En el modelo que representa el nivel PIM, puede añadirle elementos UML que modelicen el resto de la aplicación que desee desarrollar. Los plugins que hemos desarrollado replicarán todos aquellos elementos UML que no sean propios del CRUD, y en la transformación de PSM a código se cederá la generación de código de éstas clases al generador de Rational Software Architect.
IBM Rational Software Architect

A lo largo de este capítulo veremos el entorno de modelado/desarrollo IBM Rational Software Architect. Describiremos las características del software, sus capacidades y elementos que lo componen.

5.1. Introducción

La plataforma de desarrollo de software IBM Rational Software Architect (RSA) surgió como evolución de un proyecto anterior de la empresa Rational, la cual fue adquirida por IBM en el año 2003.

La empresa Rational Machines fue fundada por Paul Levy y Mike Devlin en 1981 la cual apostaba por un modelo de desarrollo de software basado en prácticas modernas, desarrollo modular e iterativo.

Uno de los productos de más éxito de la empresa Rational es RUP\(^1\) que ofrece una metodología de desarrollo de software, en la que se define una lista de best practices que ayudan al desarrollo de software. Entre estas best practices destacan el desarrollo de software en iteraciones, gestión de requisitos, verificación de la calidad del software, desarrollo con modelos, control de versiones, etc. RUP determina la lista de modelos que se tienen que llevar a cabo para modelar todo un sistema software y especifica su utilidad dentro del diseño.

Otro de los productos que desarrollaron los integrantes de Rational es UML\(^2\), el cual es un intento de reducción de la fragmentación de maneras de modelización que existe entre los diferentes integrantes de la empresa. En 1996 entregaron la especificación del UML 1.0 a la Object Management Group.

\(^{1}\)Consultar el capítulo 2.2.1
\(^{2}\)Unified Modeling Language
El predecesor de RSA es *Rational Rose*, producto desarrollado para la plataforma Windows que se integra perfectamente con la metodología RUP. *Rose* forma parte de una suite llamada *Rational Suite* en la cual también hay integrados productos como *Requisite-PRO* o *ClearCase*, adquiridos por la empresa Rational. Esta suite de modelización fue puesta al mercado en el año 1999. Posteriormente en el año 2003 la empresa IBM adquirió Rational por el módico precio de 2.100 millones de dólares.

Por esta razón histórica, IBM RSA, RUP y UML se integran perfectamente.

5.2. ¿Qué es?

Actualmente IBM RSA, ha sido reconstruido desde cero y soportado encima de la potente plataforma de desarrollo llamada Eclipse\(^3\). En él se han querido plasmar todas las ideas heredadas de las aplicaciones Rational.

Eclipse es una plataforma desarrollada íntegramente en Java. Forma parte de un proyecto *Open Source* y goza de gran popularidad en el mundo del desarrollo de software. Existen gran cantidad de paquetes y plugins para Eclipse para todo tipo de entornos.

RSA permite desarrollar y modelar todo tipo software. Aquí tenemos un listado de las capacidades que tiene RSA:

- Desarrollar, modelar diseños software con UML 2.0
- Cualquier tipo de requerimiento Java. Aplicaciones simples o complejas
- Cualquier tipo de requerimiento Web como Web Services, JSP, XHTML, etc.
- Entorno integrado de desarrollo para Java, C++, PHP, etc.
- Entorno de desarrollo para dispositivos móviles
- Tratamiento de infinidad de ficheros como XML, XSLT, DTD, etc.

Como podemos ver en el listado IBM RSA es una de las herramientas para el desarrollo de software más potentes que existen en la actualidad.

La versión con la que se ha trabajado en el proyecto ha sido la versión 6.0 pero a día de hoy se conoce que IBM está a punto de presentar la versión 7.0.

5.3. El entorno

5.3.1. Perspectivas

El conjunto de perspectivas es una de las herramientas más útiles de esta plataforma de desarrollo. Mediante el intercambio entre perspectivas, el programa obtiene o pierde

\(^3\)http://www.eclipse.org/
capacidades de desarrollo en cierto campo.

A continuación se muestra una lista con varias de las perspectivas que hay en RSA:

- **Resource:**
  Con ésta perspectiva se activa la capacidad en la gestión de ficheros físicos.

- **Java:**
  Se activan capacidades de visualización de paquetes, compilación, gestión de librerías, dependencias, etc.

- **Debug:**
  Se activan capacidades de debuguación de código, logs, la pila de llamadas a métodos, variables y otros datos en tiempo de ejecución de los programas.

- **Web:**
  En ésta perspectiva se activan capacidades en el desarrollo de temas relacionadas del mundo web. La edición de ficheros HTML, edición de scripts y organización de aplicaciones dinámicas mejora sustancialmente con ésta perspectiva.

- **Plugin Development:**
  Gran parte de la potencia de la plataforma reside en el desarrollo de plugins. Por ésta razón tener una perspectiva que facilite el desarrollo de plugins proporciona valor añadido a toda la plataforma.
5.3.2. El Model Explorer

El Model Explorer que es un interfaz en forma de árbol para poder navegar a través de los proyectos que se están desarrollando.

En este interfaz podemos ver todos los elementos que componen cada uno de los proyectos; ya sean paquetes, clases, atributos, métodos o cualquier elemento que interviene en el desarrollo del proyecto.

Gracias a las perspectivas el Model Explorer puede cambiar su aspecto ofreciendo más o menos información. Por ejemplo, el model explorer con la perspectiva Java muestra los paquetes y clases. Si cambiamos a la perspectiva Modelado el model explorer cambia y se pueden ver casos de uso, diagramas de actividad, etc.

Figura 5.3.2: Vista del Model Explorer en RSA

5.4. Requisitos Hardware

Uno de los problemas que tiene el desarrollo de aplicaciones con RSA es el consumo excesivo de recursos. Recursos como el uso del procesador o el consumo de memoria.

A continuación se listan los requisitos de hardware recomendados para el buen funcionamiento del entorno:
5.5. CARACTERÍSTICAS SOFTWARE

- Pentium 3.0 Ghz
- Memoria principal de 2 GB
- Espacio en el disco duro aproximadamente de 9 GB

5.5. Características software

5.5.1. Plugins

Una de las características software más potentes que dispone esta plataforma son los plugins. Los plugins son módulos que se pueden instalar en la aplicación para dotarla de nueva características y capacidades.

Pueden modificar cualquier aspecto de la plataforma, ya sean aspectos gráficos o capacidades internas. En el entorno RSA, los plugins son elementos esenciales para extender las características básicas.

5.5.2. Pluglets

Un pluglet es una componente que se adosa al entorno de modelado/desarrollo RSA de una forma similar a los plugins. Consiste en una clase normal de Java que tiene acceso a la API estándar de Java.

Se diferencia respecto el anterior que puede ser conectado al entorno de forma instantánea, sin necesidad de ejecutar un entorno de pruebas reducido.

Se utilizan normalmente para realizar operaciones sencillas dentro del entorno de desarrollo. Se comportan como scripts.

Un pluglet es un componente que se adosa al entorno de modelado/desarrollo RSA con la misma finalidad que un plugin. La diferencia esencial respecto el primero, es que los pluglets son scripts que dado una entrada o input generan una salida (output).

Ofrecen menos flexibilidad que los anteriores.

5.5.3. Profiles

RSA dispone de una herramienta llamada profile o perfil. Los perfiles son una manera rápida y controlada de proporcionar a los modelos un contexto dentro de todo el diseño de una aplicación. Podremos restringir al diseñador de qué opciones y decisiones quiere proporcionar al modelo. De esta manera evitaremos errores de diseño.

En la definición de un profile podemos encontrar diferentes elementos:

1. Estereotipos: conjunto de los posibles estereotipos que podemos aplicar a los elementos de un modelo. Más información en el apartado 5.5.6
2. **Enumeraciones**: son el conjunto de listas de valores que pueden tomar los atributos que tiene cada uno de los estereotipos disponibles.

En el apartado 7.2 existe un manual de la creación de un *profile*.

### 5.5.4. Motor de modelado

Una de las herramientas más potentes de RSA es su motor de modelado. Con él podemos diseñar gráficamente modelos que representen cualquier parte de la aplicación.

Por ejemplo, gracias al motor de modelado, se puede llevar a cabo el diseño de:

- **Diagramas de clases**: se puede modelar toda la estructura de clases, visualizar relaciones, herencias, interfaces, representación de patrones e infinidad de otros elementos típicos del diseño de una aplicación.

- **Diagramas de secuencia**: podremos modelar todas las interacciones entre objetos que intervienen en la aplicación. Se pueden ver y editar las llamadas entre cada una de las partes implicadas en el diagrama de secuencia.

- **Diagramas de estados**: podremos modelar el comportamiento dinámico que tiene un objeto dentro de una aplicación mediante la definición de todos los estados posibles y las transiciones posibles desde el inicio al fin.
5.5. CARACTERÍSTICAS SOFTWARE

5.5.5. Patrones

En el diseño de aplicaciones aparecen necesidades comunes a todas ellas que se tienen que resolver. Una práctica extendida en la Ingeniería del Software es la utilización de patrones de diseño. Estos patrones explican, de una manera genérica, cómo resolver dicho problema.

En RSA existe un panel de gestión de patrones (figura 5.5.2). En él podemos ver un listado de los patrones disponibles entre los que podemos encontrar entre muchos otros los siguientes:

- **Observador**: en el que para completar el patrón se define un Sujeto que será la clase que enviará la llamada avisando que algún evento a sucedido. También tendremos que definir el conjunto de Observadores que recibirán dicha llamada y podrán percatarse de que ha ocurrido un evento.

- **Estado**: en este patrón se definirá una clase que en un determinado momento tiene un estado y dinámicamente podemos cambiar el estado obteniendo nuevas características particulares de cada uno de los estados. Definiremos la clase de original y el conjunto de los diferentes estados posibles.

![Diagrama de patrones de diseño](image)

**Figura 5.5.2**: Vista de los patrones aplicables en RSA

Este panel también proporciona una pequeña documentación de cómo aplicar dicho patrón de diseño a nuestros modelos. RSA utiliza estos patrones para tomar decisiones a la hora de generar diagramas de secuencia, en la generación de código, etc.
5.5.6. Estereotipos

Como ya hemos comentado anteriormente, a la hora de trabajar con modelos en RSA, una parte de vital importancia es la definición de los *profiles*. Estos aportan la capacidad de aportar más énfasis a cada uno de los modelos diseñados.

Una de las partes más significativas a la hora de definir un *profile* es la definición de los estereotipos. Los estereotipos son elementos aplicables a cualquier parte del diseño de un modelo que nos aportará información más allá de la que aporta el propio modelo.

Por ejemplo, podemos hacer lo siguiente, gracias a los estereotipos:

- Atribuir el estereotipo de *ControlStruts* a una clase de un modelo UML
- Atribuir el estereotipo de *CRUDView* a un caso de uso de un modelo UML
- Atribuir el estereotipo de *Restrictor* a una relación entre clases de un modelo UML
- Atribuir el estereotipo de *RegExp* a un atributo de tipo *String* de un modelo UML

Cada uno de estos estereotipos podría contener atributos modificables que aportarían información adicional al elemento al cual se ha aplicado. Por ejemplo el estereotipo *ControlStruts* podría contener los atributos:

- **Variable dispatch**: Variable que se utilizará a la hora de controlar qué vista llama al controlador Struts
- **Validación de entradas**: En el cliente o en servidor. Es decir, si la llevará a cabo el cliente o el servidor

O en el caso del estereotipo *RegExp* que hemos aplicado a un atributo de una clase, podría contener los atributos:

- **Expresión Regular**: Expresión regular que deberá cumplir el atributo *String* al cual le hemos aplicado el estereotipo mencionado cada vez que se modifica. Podríamos validar de esta manera entradas como *DNI's*, direcciones de correo, etc.
- **Color error**: Color de error con el que se va a mostrar en las vistas dicho atributo si no cumple la expresión regular

Como se puede ver, las ventajas que nos pueden aportar la atribución de estereotipos a los elementos de un modelo son significativas.

<table>
<thead>
<tr>
<th>Decisión:</th>
</tr>
</thead>
<tbody>
<tr>
<td>El uso de estereotipos es la base en el desarrollo del proyecto. Toda la información que tendrán los modelos en cada uno de los niveles de abstracción será a causa de la atribución de unos u otros estereotipos a los elementos que conforman los modelos.</td>
</tr>
</tbody>
</table>
5.5. CARACTERÍSTICAS SOFTWARE

5.5.7. El entorno de pruebas

En el momento de desarrollar plugins para RSA, una de las características más deseables es la posibilidad de probar inmediatamente los nuevos elementos que hemos creado. Por esta razón, en RSA se puede lanzar un entorno de desarrollo exactamente igual al que disponemos. La particularidad de este nuevo entorno es que no contiene los proyectos que estamos desarrollando ni las configuraciones de nuestro entorno.

En este entorno virtual tenemos los plugins que queremos probar. De esta manera podemos llevar a cabo depuraciones de código, control de errores, inspección de variables, trazabilidad de llamadas y cualquier tipo de consulta típica del entorno de depuración.

Uno de los problemas encontrados a la hora de probar nuestros propios plugins fue que el lanzamiento del entorno virtual con la configuración por defecto se lanza con todos los plugins instalados en la máquina.

El proceso de lanzamiento por defecto puede tardar perfectamente entre diez y veinte minutos, razón por la cual probar cada uno de los cambios que se llevaban a cabo era muy costoso. A consecuencia de esto se hizo un estudio de los paquetes necesarios que necesitaba el entorno virtual para funcionar. Al final se consiguió reducir el tiempo de carga a un par de minutos; gracias a ello se facilitó enormemente el trabajo.

Como software adicional entregado, se incluye el fichero Entorno de trabajo.launch, que corresponde a la configuración necesaria para lanzar dicho entorno, en el que únicamente se cargan los plugins necesarios para probar y debugar las transformaciones desarrolladas.

Para poder probar ésta configuración, se deben seguir los pasos siguientes:

1.Entrar en el directorio:
   `<carpeta de trabajo>\.metadata\.plugins\org.eclipse.debug.core\.launches`

2. Copiar dentro de ella la configuración entregada

3. Rearranzar el entorno de trabajo de Rational Software Architect

4. Comprobar que existe la nueva configuración en la lista de configuraciones de lanzamiento

5. Lanzar dicho entorno de pruebas y trabajar con las transformaciones importadas
Figura 5.5.3: Configuraciones de lanzamiento
Transformaciones *

A lo largo de este capítulo veremos en detalle cómo se realizan transformaciones de modelos en Rational Software Architect, qué elementos las componen y qué función tiene cada uno de ellos.

También veremos cómo se utiliza la librería UML2 utilizada en nuestras transformaciones, y una librería propia de utilidades.

6.1. Transformación de modelos

Rational Software Architect nos proporciona una funcionalidad esencial para este proyecto: las transformaciones de modelos. Véamos a continuación qué elementos conforman una transformación, y qué papel juega cada uno de ellos.

Decisión:
Para la realización de este proyecto, nos hemos ayudado del asistente de creación de un plugin para la transformación de modelos. Así mismo, hemos seguido la guía de best-practices que proporciona el fabricante respecto la transformación de modelos. Este es el motivo por el que hemos descartado realizar transformaciones aprovechando MTL, ATL, etc.

Uno de los puntos que recomienda IBM para el desarrollo de transformaciones, es la creación de una estructura intermedia en la transformación. A la parte de la transformación que pasa del input al modelo intermedio se le denomina front-end. Por otra parte, la parte que pasa del modelo intermedio al output se le denomina back-end.