

Arquitectura dirigida por modelos para J2ME

Luis Enrique Corredera de Colsa

Doctorando en Ingeniería del software

Universidad Pontificia de Salamanca en Madrid.

luisenrique@flagsolutions.net

1 Abstract

Una gran parte de los esfuerzos en el desarrollo de software actualmente consiste en la construcción de aplicaciones que además de una lógica de negocio no demasiado extensa, presentan interfaces para el manejo de las entidades que tiene que tratar el sistema en una estructura muy similar a la que se ofrece en los diagramas de clases que especifican los requisitos del sistema. Mediante el uso de MDA es posible reducir drásticamente los esfuerzos en el desarrollo de este tipo de aplicaciones. Es una tendencia en el software actual la distribución de las aplicaciones para agentes móviles en dispositivos de conectividad limitada, como son los PDA o los teléfonos móviles. Dado el aumento en capacidad computacional que han sufrido los teléfonos móviles en los últimos meses, y la innegable extensión en el uso de los mismos por parte de los usuarios, los teléfonos móviles son candidatos ideales para ejecutar clientes ligeros de las aplicaciones empresariales en terminales de bajo coste. En este artículo se aborda las características de MDA, las restricciones de la plataforma J2ME, el estudio de las principales herramientas de desarrollo MDA, y un posible enfoque para la transformación de modelos a la plataforma J2ME.

2 Keywords

Model Driven Architecture, Java 2 Platform Micro Edition, MIDP, CLDC, virtual machine, code generation.

3 Tabla de contenidos

1	Abstract.....	1
2	Keywords.....	2
3	Tabla de contenidos.....	2
4	Índice de figuras	3
5	Arquitectura dirigida por modelos para J2ME	4
5.1	Introducción a MDA.....	4
5.2	Conceptos de MDA.....	5
5.2.1	Sistema	5
5.2.2	Modelo.....	6
5.2.3	Dirigido por modelos.....	6
5.2.4	Arquitectura	6
5.2.5	Punto de vista.....	7
5.2.6	Vista	7
5.2.7	Plataforma	7
5.2.8	Aplicación	7
5.2.9	Independencia de la plataforma	7
5.2.10	Puntos de vista MDA.....	8
5.2.11	Modelo independiente de la computación	8
5.2.12	Modelo independiente de la plataforma.....	9
5.2.13	Modelo específico de la plataforma	9
5.2.14	Modelo de plataforma.....	9

5.2.15	Transformación de modelos	10
5.2.16	Servicios penetrantes	11
5.3	Usando MDA	11
5.3.1	Mapas de transformación	12
5.3.2	Transformaciones.....	15
5.4	Introducción a J2ME	17
5.4.1	La interfaz de usuario	21
5.4.2	Persistencia de la información.....	23
5.4.3	Comunicación.....	24
6	Revisión de herramientas	24
6.1	Características de las herramientas	24
7	Comparativa de herramientas.....	26
7.1	ArcStyler	27
7.2	AndroMDA.....	27
7.3	Codagen Architect 3.2	28
7.4	IQGen 1.4	29
7.5	OptimalJ	30
8	Conclusiones	31
9	Bibliografía.....	32
10	Referencias Web.....	32

4 Índice de figuras

Figura 1: Transformación de modelos	10
Figura 2: Transformación de metamodelos.....	13
Figura 3: Marcado de un modelo	14
Figura 4: Uso de información adicional y patrones en la transformación.....	15
Figura 5: JAVA 2ME dentro de las tecnologías JAVA.....	17
Figura 6: Perfiles y configuraciones.....	19

Figura 7: Ciclo de vida de un midlet.....	20
Figura 8: Jerarquía de clases MIDP para interfaz de usuario	22
Figura 9: Generación de código con herramientas MDA.....	24

5 Arquitectura dirigida por modelos para J2ME

5.1 Introducción a MDA

La arquitectura dirigida por modelos (Model Driven Arquitectura) es una especificación detallada por el OMG (Object Management Group) que integra diferentes especificaciones y estándares definidos por la misma organización con la finalidad de ofrecer una solución a los problemas relacionados con los cambios en los modelos de negocio, la tecnología y la adaptación de los sistemas de información a los mismos.

MDA nos permite el despliegue de aplicaciones empresariales, diseñadas sin dependencias de plataforma de despliegue y expresado su diseño mediante el uso de UML y otros estándares, potencialmente en cualquier plataforma existente, abierta o propietaria, como servicios web, .Net, Corba, J2EE, u otras.

La especificación de las aplicaciones y la funcionalidad de las mismas se expresa en un modelo independiente de la plataforma que permite una abstracción de las características técnicas específicas de las plataformas de despliegue. Mediante transformaciones y trazas aplicadas sobre el modelo independiente de la plataforma se consigue la generación automática de código específico para la plataforma de despliegue elegida, lo que proporciona finalmente una independencia entre la capa de negocio, y la tecnología empleada. De esta manera es mucho más simple la incorporación de nuevas funcionalidades, o cambios en los procedimientos de negocio sin tener que llevar a cabo los cambios en todos los niveles del proyecto. Simplemente se

desarrollan los cambios en el modelo independiente de la plataforma, y éstos se propagarán a la aplicación, consiguiendo por tanto una considerable reducción del esfuerzo en el equipo de desarrollo, en los errores que tienden a producirse en los cambios introducidos en las aplicaciones mediante otros métodos de desarrollo, y por consiguiente, la reducción de costes y aumento de productividad que conlleva, tan demandados tanto la industria de desarrollo de software como el resto de las empresas.

MDA se apoya sobre los siguientes estándares para llevar a cabo su función:

- UML: empleado para la definición de los modelos independientes de la plataforma y los modelos específicos de las plataformas de destino. Es un estándar para el modelado introducido por el OMG.
- MOF: establece un marco común de trabajo para las especificaciones del OMG, a la vez que provee de un repositorio de modelos y metamodelos.
- XMI: define una traza que permite transformar modelos UML en XML para poder ser tratados automáticamente por otras aplicaciones.
- CWM: define la transformación de los modelos de datos en el modelo de negocio a los esquemas de base de datos.

5.2 Conceptos de MDA

De cara a entender MDA y sus características, su funcionamiento y su aplicación al proceso de desarrollo, revisaremos los conceptos básicos de MDA y su forma de uso.

5.2.1 Sistema

Los conceptos de MDA se definen centrados en la existencia o planteamiento de un sistema, que puede contener un simple sistema informático, o

combinaciones de componentes en diferentes sistemas informáticos, o diferentes sistemas en diferentes organizaciones, etc.

5.2.2 Modelo

Un modelo de un sistema es una descripción o una especificación de ese sistema y su entorno para desempeñar un determinado objetivo. Los modelos se presentan normalmente como una combinación de texto y dibujos. El texto se puede presentar en lenguaje de modelado, o en lenguaje natural.

5.2.3 Dirigido por modelos

MDA es un acercamiento al desarrollo de sistemas, que potencia el uso de modelos en el desarrollo. Se dice que MDA es dirigido por modelos porque usa los modelos para dirigir el ámbito del desarrollo, el diseño, la construcción, el despliegue, la operación, el mantenimiento y la modificación de los sistemas.

5.2.4 Arquitectura

La arquitectura de un sistema es la especificación de las partes del mismo, las conexiones entre ellos, y las normas de interacción entre las partes del sistema haciendo uso de las conexiones especificadas.

MDA determina los tipos de modelos que deben ser usados, como preparar dichos modelos y las relaciones que existen entre los diferentes modelos.

5.2.5 Punto de vista

Un punto de vista es una abstracción que hace uso de un conjunto de conceptos de arquitectura y reglas estructurales para centrarse en aspectos particulares del sistema, obteniendo un modelo simplificado.

5.2.6 Vista

Una vista es una representación del sistema desde un determinado punto de vista.

5.2.7 Plataforma

Una plataforma es un conjunto de subsistemas y tecnologías que aportan un conjunto coherente de funcionalidades a través de interfaces y determinados patrones de uso, que cualquier aplicación que se construya para esa plataforma puede usar sin preocuparse por los detalles de la implementación o como se lleva a cabo la misma dentro de la plataforma.

5.2.8 Aplicación

En MDA se define el término aplicación como una funcionalidad que tiene que ser desarrollada. Por tanto podemos definir un sistema en términos de la implementación de una o más aplicaciones, soportadas por una o más plataformas.

5.2.9 Independencia de la plataforma

La independencia de la plataforma es una cualidad que tienen que presentar los modelos. Lo que significa que un modelo es independiente de las

facilidades o características que implementan las plataformas, de cualquier tipo.

5.2.10 Puntos de vista MDA

MDA establece tres puntos de vista que se emplearán a lo largo del proceso de ingeniería:

- Punto de vista independiente de la computación: se centra en el entorno del sistema y los requisitos para el mismo. Los detalles de la estructura y procesamiento del sistema no se muestran, o aún no están especificados.
- Punto de vista independiente de la plataforma: se centra en las operaciones del sistema, mientras oculta los detalles necesarios para una determinada plataforma. Muestra aquellas partes de la especificación del sistema que no cambian de una plataforma a otra. En este punto de vista debe emplearse lenguaje de modelado de propósito general, o bien algún lenguaje específico del área en que se empleará el sistema, pero en ningún caso se emplearán lenguajes específicos de plataformas.
- Punto de vista de plataforma específica: combina el punto de vista independiente de la plataforma con un enfoque específico para su uso en una plataforma específica en un sistema.

5.2.11 Modelo independiente de la computación

Un modelo independiente de la computación (CIM) es una vista del un sistema desde el punto de vista independiente de la computación. En algunos casos, nos referimos al modelo independiente de la computación como el modelo del dominio, y se usa vocabulario propio de los expertos en el dominio para la especificación.

5.2.12 Modelo independiente de la plataforma

Un modelo independiente de la plataforma (PIM) es una vista del sistema desde el punto de vista independiente de la plataforma. Expone un carácter independiente de la plataforma sobre la que se desplegará, de modo que pudiera ser empleado en diferentes plataformas de carácter similar.

Una técnica común para alcanzar el grado de independencia de la plataforma necesario es definir un sistema basado en una máquina virtual que abstraiga los modelos particulares de las plataformas existentes y sea neutral respecto a las mismas.

5.2.13 Modelo específico de la plataforma

Un modelo específico de la plataforma (PSM) es una vista de un sistema desde el punto de vista dependiente de la plataforma. Combina las especificaciones del modelo independiente de la plataforma con los detalles que especifican el uso de una plataforma específica por parte del sistema.

5.2.14 Modelo de plataforma

Un modelo de plataforma expone un conjunto de conceptos técnicos que representan las diferentes formas o partes que conforman un sistema, y los servicios que provee. También expone, para su uso en los modelos específicos de la plataforma, conceptos que explican los diferentes elementos que provee una plataforma para la implementación de una aplicación en un sistema.

Una modelo de plataforma incluye también la especificación de requisitos en la conexión y uso de las partes que integran la plataforma, y la conexión de aplicaciones a la plataforma.

5.2.15 Transformación de modelos

La transformación de modelos es el proceso de convertir un modelo en otro modelo del mismo sistema.

La Figura 1 ilustra la transformación del modelo independiente de la plataforma en un modelo específico para una plataforma mediante el uso de información añadida que permita trazar ambos modelos.

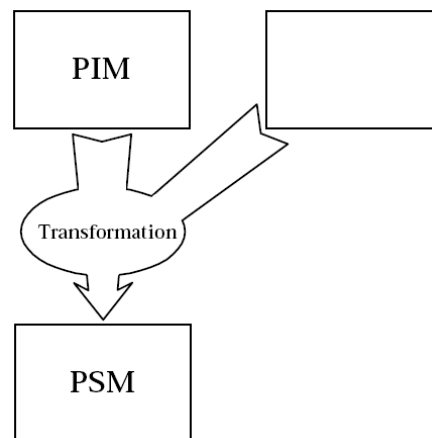


Figura 1: Transformación de modelos

La transformación de un modelo independiente de la plataforma en un modelo dependiente de la plataforma no es necesaria para PIM basados en una máquina virtual. En este caso hay que transformar el PIM correspondiente a la máquina virtual en un modelo de plataforma específico.

5.2.16 Servicios penetrantes

Los servicios penetrantes son servicios que están disponibles un amplio catálogo de plataformas. MDA proveerá servicios penetrantes comunes e independientes de la plataforma y dispondrá de trazas para la transformación de los modelos, que permitirá la transformación de los servicios expuestos en los PIMs a las plataformas de destino.

5.3 Usando MDA

Expuestos los conceptos básicos de MDA, es necesario conocer como se relacionan unos modelos, su modo de uso y las transformaciones entre PIM y PSM.

El modelo de origen es el CIM, con el que se modelan los requisitos del sistema, describiendo la situación en que será usado el sistema y que aplicaciones se espera que el sistema lleve a cabo, sirviendo tanto como ayuda para entender el problema como una base de vocabulario para usar en los demás modelos.

Los requisitos recogidos en el CIM han de ser trazables a lo largo de los modelos PIM y PSM que los implementan.

El CIM puede consistir en un par de modelos UML que muestren tanto la distribución de los procesos (ODP, Open Distributed Processing) como la información a tratar. También puede contener algunos modelos UML más que especifiquen en más detalle los anteriores.

A partir del CIM, se construye un PIM, que muestra una descripción del sistema, sin hacer referencia a ningún detalle de la plataforma. Debe presentar

especificaciones desde el punto de vista de la empresa, información y ODP. Un PIM se puede ajustar a un determinado estilo de arquitectura, o a varios.

Después de la elaboración del PIM, el arquitecto debe escoger una plataforma (o varias) que satisfagan los requisitos de calidad.

5.3.1 Mapas de transformación

Mediante mapas, MDA especifica las reglas de transformación de un PIM a un PSM para una plataforma en concreto. Estos mapas incluyen la transformación de tipos de valores para la transformación desde el PIM al PSM, los metamodelos y sus reglas para la transformación en tipos de valores existentes en las plataformas.

Cuando se prepara un modelo haciendo uso de un lenguaje independiente de la plataforma, especificado en un metamodelo y posteriormente se elige una plataforma para el despliegue, debe existir una especificación de transformación entre el metamodelo independiente de la plataforma y el metamodelo que describe la plataforma. Este requisito se ilustra en la Figura 2.

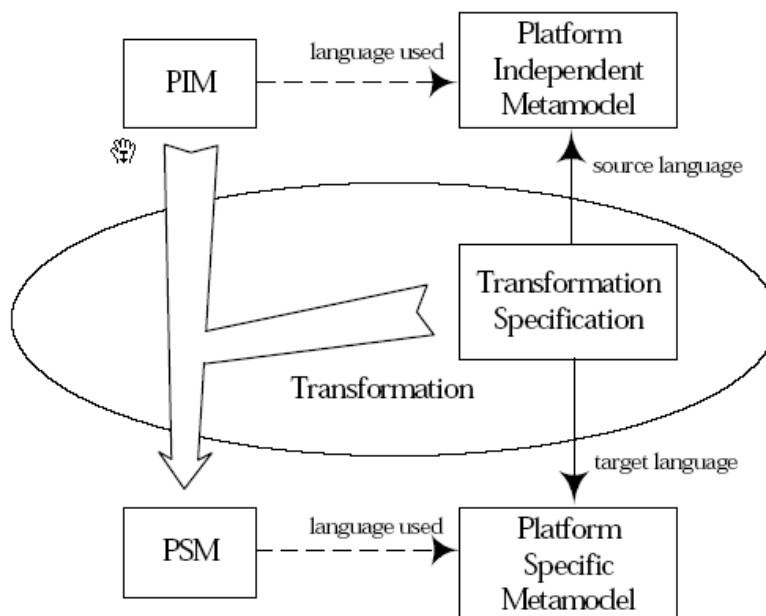


Figura 2: Transformación de metamodelos

Una forma de facilitar la transformación de modelos es la identificación de elementos en el PIM que deben transformarse de una manera concreta para la plataforma de destino, y marcarlos como tal. Una marca expresa un concepto del PSM en el PIM; las marcas alejan el PIM de la independencia de la plataforma, por lo que un arquitecto debe mantener un PIM limpio, marcarlo para su adaptación a una plataforma en concreto. Las marcas deben concebirse como una capa transparente que se pone sobre el modelo.

La Figura 3 ilustra el uso de marcas como ayuda para la transformación, y su situación intermedia entre la independencia y la dependencia de la plataforma. Una vez es elegida la plataforma, existe un mapa para la transformación de modelos. Este mapa incluye un conjunto de marcas, que usamos para marcar los elementos del PIM como guía para la transformación del modelo.

Las marcas pueden definir tipos del modelo, especificación de clases, asociaciones, roles, estereotipos,... las marcas también pueden especificar

características cualitativas que después en la transformación se convertirá en el objetivo apropiado para el cumplimiento de los requisitos.

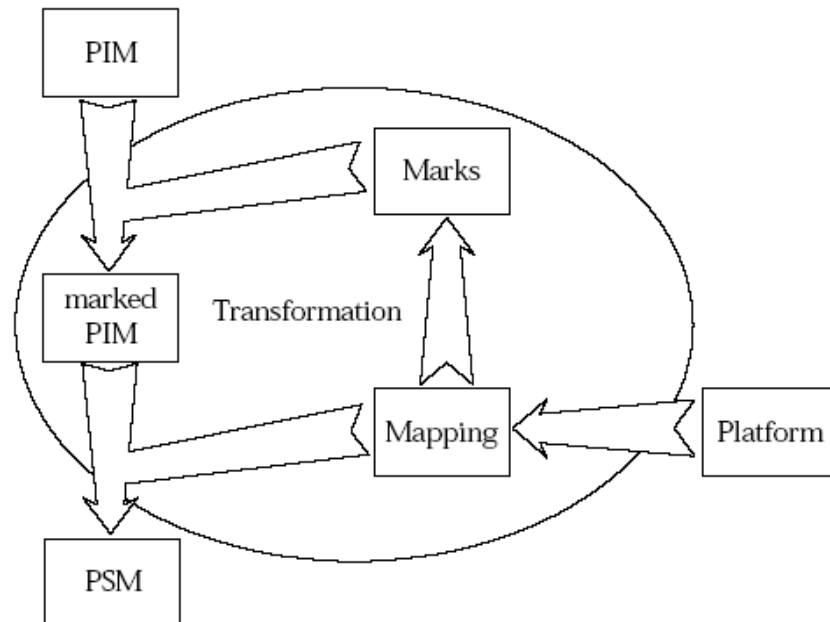


Figura 3: Marcado de un modelo

Las marcas deben tener un modelo y una estructura que permita mantener la coherencia, que impida el marcado de un elemento con marcas mutuamente excluyentes.

Los mapas de transformación pueden mantener también plantillas, que son modelos parametrizados que especifican tipos concretos de transformaciones. Podemos asociar un conjunto de marcas a una plantilla para identificar instancias en un modelo que deben ser transformados de acuerdo a las indicaciones de la plantilla.

Un elemento en un modelo puede ser marcado varias veces, empleando marcas procedentes de diferentes plantillas, y de distintos mapas de

La transformación es el proceso que toma como entrada el PIM marcado y da como resultado el PSM del sistema, y el registro de transformación.

Algunas herramientas pueden hacer una transformación del PIM directamente a código desplegable en la plataforma de destino o incluso conceptos como MDR (Model-Driven Runtime Environment) que propone el uso de un entorno de ejecución para los PIM, directamente, sin generación de PSM ni código para la plataforma. En cualquier caso el OMG sostiene que la transformación debe emitir un PSM para ayudar al entendimiento del código y la depuración del mismo.

El registro de transformación incluye una traza desde cada elemento del PIM a los elementos correspondientes en el PSM, especificando que parte de los mapas de transformación fueron empleados para derivar los elementos del PSM desde los elementos del PIM. Una herramienta que mantenga los registros de transformación debe ser capaz de sincronizar de forma automática los cambios producidos en un modelo al otro.

El PSM producido por una transformación es un modelo del mismo sistema que ha sido especificado en el PPIM. También especifica como el sistema hace uso de una determinada plataforma.

Un PSM será una implementación del sistema si proporciona toda la información necesaria par construir el sistema y ponerlo en marcha.

5.4 Introducción a J2ME

J2ME es una tecnología basada en Java que permite el desarrollo de aplicaciones para teléfonos móviles y dispositivos con capacidad de proceso limitada. En la **figura 2** se aprecia claramente dónde se enmarca la tecnología que se ha usado en este proyecto, dentro de la “maraña” de tecnologías que es JAVA.

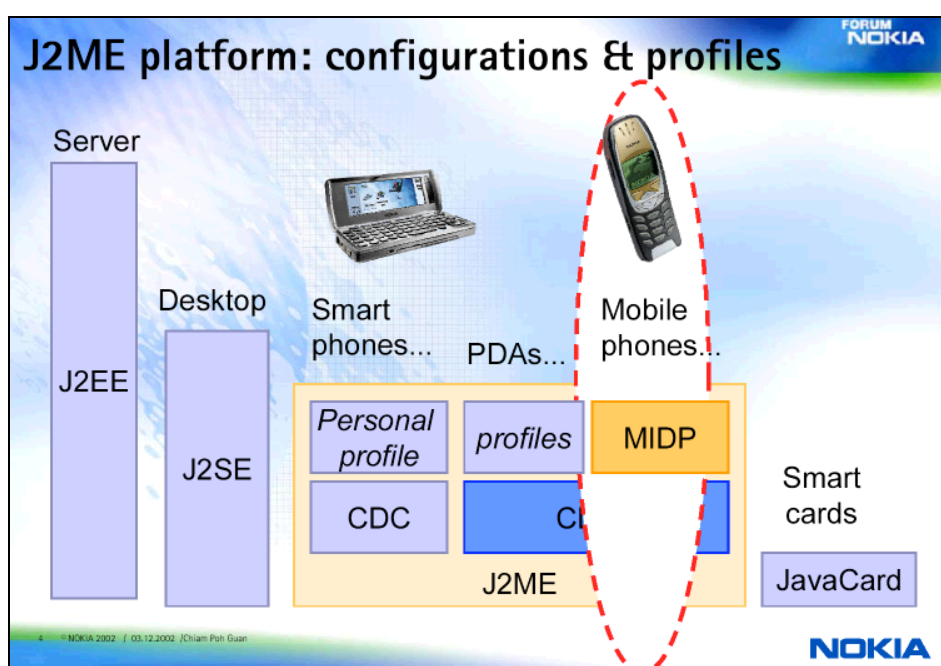


Figura 5: JAVA 2ME dentro de las tecnologías JAVA

MIDP (*Mobile Information Device Profile*) es un conjunto de APIs de JAVA, que junto con CLDC (*Connected Limited Device Configuration*), constituyen un entorno de ejecución de aplicaciones J2ME completo, orientado a dispositivos de información móvil de bajo coste, como son por ejemplo los teléfonos móviles. El perfil MIDP provee de un entorno estándar de ejecución que

permite desplegar nuevas aplicaciones y servicios de forma dinámica en los terminales móviles de los usuarios.

El ya mencionado CLDC, define el lenguaje JAVA y las peculiaridades de la máquina virtual usada. De hecho está íntimamente ligado a ésta. Especifica la memoria ROM para uso de la máquina virtual y de las clases del sistema (128Kb), y exige un mínimo 32Kb de RAM para las clases de aplicaciones. Define así mismo el núcleo central de bibliotecas que la máquina virtual debe de contener. Éstas son:

- APIs fundamentales: *JAVA.lang*
- Métodos de Entrada/Salida y manejo de flujos de datos: *JAVA.io*
- Calendario, Fecha, generación de números aleatorios, estructuras básicas de datos: *JAVA.util*
- Métodos de red básicos: *JAVAx.microedition.io*

MIDP se asienta sobre CLDC. MIDP es una plataforma que nos proporciona un completo entorno de ejecución para un tipo específico de dispositivo. Define APIs para componentes de interfaz de usuario, entrada, manejo de eventos, persistencia de datos, acceso a redes y temporizadores, pero siempre teniendo en consideración las limitaciones de memoria y pantalla de los dispositivos móviles.

MIDP tiene las siguientes APIs:

- Interfaz de Usuario: *JAVAx.microedition.lcdui*
- HTTP1: *JAVAx.microedition.io.HttpConnection*
- Persistencia de Datos: *JAVAx.microedition.rms*
- Ciclo de vida de la Aplicación: *JAVAx.microedition.midlet*

- Temporizadores: `JAVA.util.Timer` & `JAVA.util.TimerTask`
- Un tipo nuevo de excepción: `JAVA.lang.IllegalStateException`

En la podemos ver un esquema de cómo se organizan estos conceptos:



Figura 6: Perfiles y configuraciones

La JVM es la *JAVA Virtual Machine* mientras que las siglas KVM es la Kilobyte Virtual Machine. Como se puede deducir, la máquina virtual JAVA que se utiliza para la configuración CLDC, está limitada a 1Kbyte, de ahí sus numerosas limitaciones. La más significativa de las consecuencias de una máquina virtual tan pequeña, es que no pueda realizar una verificación de las clases para que no realicen operaciones no permitidas. Es por ello que cuando se programa en J2ME hay una fase nueva en el ciclo de vida del software que viene justo después de la etapa de compilación. Ésta es la **preverificación**.

Otro concepto importante es el de MIDlet. Las aplicaciones escritas para la plataforma J2ME/MIDP se llaman MIDlets.

Un MIDlet es un conjunto de clases que colaboran para realizar una cierta acción. Siempre una de esas clases deberá heredar de la clase **javax.microedition.midlet.MIDlet**. Esa clase será la encargada de “cumplir” con el CICLO DE VIDA de un MIDlet que se muestra en el siguiente gráfico:

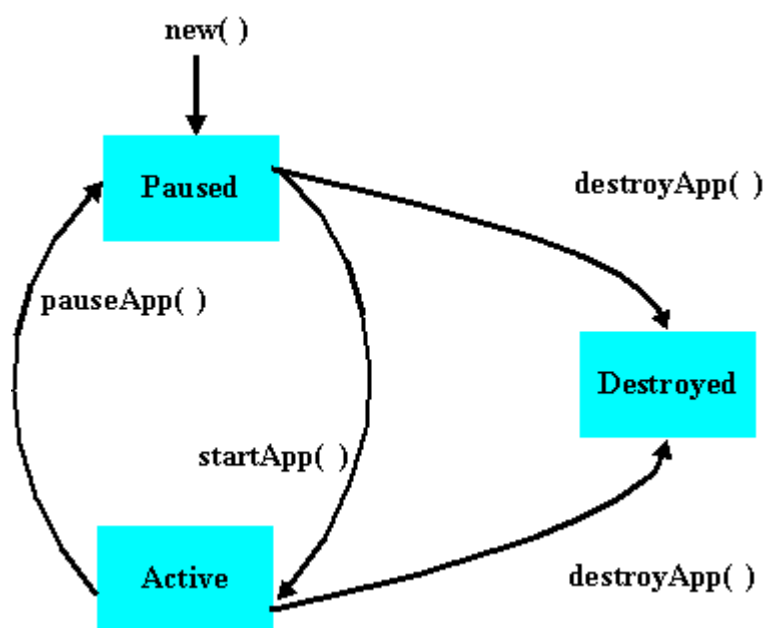


Figura 7: Ciclo de vida de un midlet

En todos los teléfonos que soporten JAVA siempre deberá haber un **gestor de aplicaciones** que sea el que se encargue de controlar el ciclo de vida de los MIDlets que se ejecutan. Un ciclo de vida típico de un MIDlet sería:

- El Gestor de aplicaciones crea una nueva instancia del MIDlet (`new ()`). El MIDlet estaría ahora en estado *Paused*.
- El Gestor de aplicaciones decide ceder los recursos del teléfono al MIDlet, así que ejecuta su método *startApp ()*. El MIDlet entra en estado

Active. El MIDlet puede ahora realizar el servicio para el que fue programado.

- El Gestor de aplicaciones decide desactivar la ejecución del MIDlet (porque hay una llamada entrante por ejemplo), y ejecuta el método *pauseApp()*. El MIDlet entra en estado *Paused* y debe liberar todos los recursos que estaba utilizando.
- El Gestor de aplicaciones envía una señal *destroyApp()* al MIDlet para que este termine su ejecución y se libere de memoria. El MIDlet entra en estado *Destroyed*.

5.4.1 La interfaz de usuario

Una vez que ya introducida la tecnología y comentado qué es un MIDlet, es interesante centrarse en la **interfaz de usuario** de un MIDlet, pues el presente proyecto está basado enteramente en una de esas interfaces. Cuando se afronta un proyecto en el que hay que desarrollar MIDlets la primera decisión que hay que tomar es si se van a utilizar **Interfaces de usuario de Alto nivel o de Bajo Nivel**. Esta decisión afectará sin lugar a duda a muchos aspectos del desarrollo del software, pues entre otras cosas provocará que la aplicación pierda la **portabilidad** que promulgan las tecnologías de JAVA. Una interfaz de usuario de Alto nivel proporcionará la mayoría de *widgets* y formularios necesarios para que un usuario pueda manejar nuestra aplicación, siempre y cuando ésta no tenga que mostrar gráficos de forma dinámica. En este caso, que es el desarrollo de sistemas de información para teléfonos JAVA, podemos olvidarnos casi por completo del bajo nivel y centrarnos en el uso de los formularios estándar para la interacción con el usuario, que nos permiten mantener una presentación homogénea cualesquiera que sean los terminales en los que se despliegue la aplicación.

Debe evitarse en todo momento el uso de la clase *Canvas*, ya que su característica de bajo nivel hará muy probable que se haya perdido la portabilidad de la aplicación a otros teléfonos con distintas posibilidades gráficas que el que estamos probando.

Además, el *look&feel* de la aplicación hecha a bajo nivel ya no será el que el fabricante del teléfono eligió para su interfaz de alto nivel, que debería ser idéntico al del resto de menús y opciones del teléfono, con los que el usuario ya está familiarizado. Es ésta, por tanto, otra gran desventaja de no usar interfaces de usuario de Alto nivel.

En la Figura 8 se observa toda la jerarquía de clases de MIDP, de la que la única clase que se puede utilizar para una interfaz de usuario de bajo nivel es la clase *Canvas*, y que evitaremos siempre en nuestros desarrollos de sistemas de información.

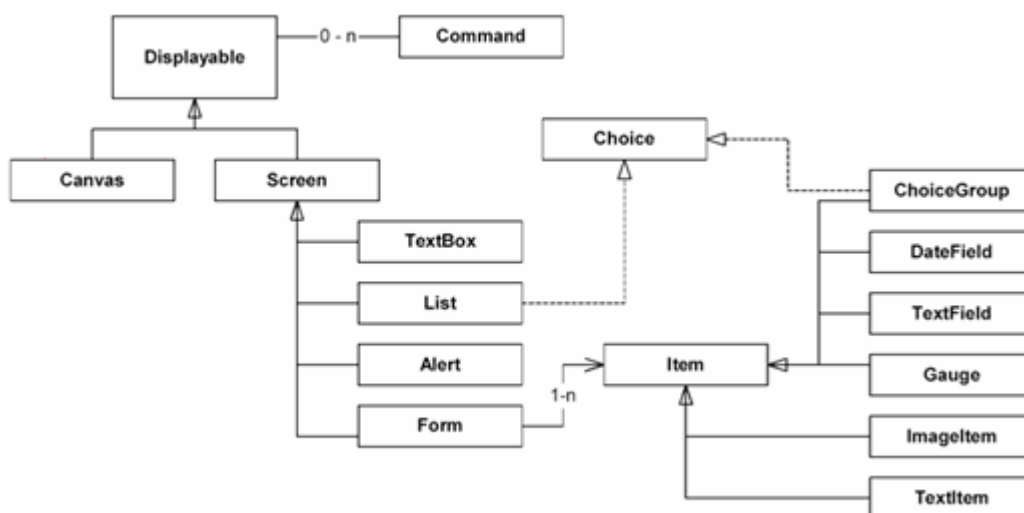


Figura 8: Jerarquía de clases MIDP para interfaz de usuario

5.4.2 Persistencia de la información

De vital importancia para poder implementar sistemas de información en dispositivos móviles que soportan Java es la capacidad de los mismos para almacenar información y comunicarla con otros componentes del sistema.

Desde el punto de vista del almacenamiento, los dispositivos J2ME proveen un pequeño espacio de almacenamiento persistente para las midlet suites, compartido solo por los midlets que pertenecen a una misma suite. Este mecanismo se denomina Record Management System (RMS), y está entendido como una base de datos orientada a registros simples.

La implementación de RMS asegura que todas las operaciones individuales sobre un registro son atómicas, sincronizadas y serializadas para evitar corrupción en los registros en acciones de acceso múltiple. No obstante, cuando un Midlet emplea diferentes hilos que acceden al RMS es obligación del Midlet controlar la concurrencia. Si o lo hace y varios hilos intentan escribir sobre un mismo registro de manera concurrente, RMS serializará las operaciones y se ejecutarán una detrás de otra, sin riesgos de corrupción para la base de datos, pero con efectos imprevisibles para el midlet, ya que los valores almacenados serían sobrescritos en las sucesivas llamadas.

RMS provee de timestamps para los almacenes de registros, y control de versiones en forma de un número entero que se autoincrementa en cada operación de modificación del registro. Estas dos características pueden ser de mucha utilidad para la sincronización de información con otras aplicaciones del sistema.

5.4.3 Comunicación

Para la integración de los terminales móviles con otros componentes del sistema de información es necesario disponer de unas capacidades mínimas de comunicación, que faciliten a los midlets la posibilidad de conectar con sistemas de sincronización, y comunicar los registros almacenados en el RMS para evitar la saturación de los almacenes de registros.

Midp proporciona una interfaz de comunicación que, aunque es muy restringida es muy flexible, permitiendo establecer conexiones mediante sockets tcp, envío de datagramas, conexiones a puertos serie, conexiones a ficheros y un conector de alto nivel http, para realizar peticiones web.

6 Revisión de herramientas

6.1 Características de las herramientas

El concepto de herramienta que soporta MDA es el de una herramienta que provee de automatización el proceso de desarrollo MDA anteriormente descrito, y que visto en síntesis puede resumirse en la representación de la Figura 9, donde las flechas entre las cajas indican las sucesivas transformaciones que han de llevarse a cabo entre modelos siguiendo unas determinadas reglas, como se ha expuesto anteriormente en este texto.



Figura 9: Generación de código con herramientas MDA

Las herramientas que soportan MDA pueden ofrecer diferentes características, entre las que son deseables al menos las siguientes:

- Creación de modelos de datos entidad-relación partiendo del modelo CIM:
 - o Conservando todas las restricciones especificadas en el modelo: claves primarias y claves externas, multiplicidad de las relaciones (n:m, 1:n, 0:n, ...), longitud de atributos y precisión de los mismos.
 - o Manejo de disparadores.
- Generación y modelado de PIM desde modelos entidad-relación:
 - o Modelos de clases con objetos estereotipados.
 - o Uso de diagramas de actividad y otros elementos que demuestren comportamiento (diagramas de colaboración, secuencia, estados,...)
 - o Uso de OCL para la especificación de restricciones.
 - o Soporte para valores etiquetados en las clases de datos, para facilitar la internacionalización del producto.
- Generación de PSM:
 - o Elección de la plataforma para la que generar el PSM (EJB/JDO, .NET, Corba...) y elección de la tecnología de cliente (Struts, JSP, XMLC, Swing, SWT, ...).
 - o División del modelo de datos y la interfaz gráfica de usuario en paquetes diferentes.
 - o Generación de ficheros de propiedades (I18n).
 - o Generación correcta de los diagramas de actividad apropiados.
 - o Generar clases de prueba.
- Adaptación del PSM:
 - o Poder introducir código suplementario al generado que permita completar la aplicación. Este código debe ser mantenido en un

repositorio para evitar tener que reescribirlo en sucesivas generaciones del modelo.

- Generación de la aplicación:
 - o Generación del esquema de base de datos.
 - o Generación de código.
 - o Generación de Makefiles, ANT, u otros scripts para la construcción del código.
 - o Producción de ficheros de configuración.
 - o Producción de documentación.

Como MDA es un conjunto de estándares alineados para coadyuvar en un punto de vista particular para el desarrollo de software, la implementación de herramientas MDA permite establecer características desde un punto de vista subjetivo para los productores de herramientas, permitiendo interpretaciones diversas sobre la aplicación de los estándares.

En este momento no existen muchas herramientas que soportan MDA para el desarrollo de software, aunque es previsible que el número crezca con el paso de no demasiado tiempo.

7 Comparativa de herramientas

Aunque las herramientas que soportan MDA no son muchas, su número es suficiente para elegir las herramientas más populares de cara a un análisis comparativo de las mismas. Más detalle acerca de las herramientas aquí presentadas y otras que no se citan en este documento se pueden obtener en la página web de OMG.

7.1 ArcStyler

Es un sistema basado en uso de cartuchos para descripción de transformaciones que permite generar aplicaciones de n capas codificadas en java/J2EE y c#/.NET a partir de diagramas UML y la especificación de los procesos del negocio.

Permite extender las capacidades de transformación, generando nuevos cartuchos a partir de UML, cuyo objetivo sea cualquier plataforma o lenguaje.

Usa Rose como herramienta de modelado UML, soportando directamente los ficheros MDL. Admite como entrada UML v1.3, con los siguientes elementos:

- Diagramas de clases
- Restricciones
- OCL
- Casos de uso
- Diagramas de actividad
- Diagramas de estado
- Diagramas de secuencia
- Diagramas de colaboración.

No soporta diagramas de componentes ni diagramas de despliegue, pero admite código propio para la generación de código.

7.2 AndroMDA

AndroMDA es un sistema también basado en cartuchos, que admite como entrada descripciones XMI de diagramas UML, y usa XDoclet como tecnología de marcado para el acceso a datos desde las clases Java.

Admite como entrada ficheros XMI versión 1.1, y como herramienta de modelado la comunidad de desarrollo aconseja el uso de Poseidon for UML, de Gentleware.

Admite UML v1.4, con los siguientes elementos:

- Diagramas de clases
- Restricciones
- OCL
- Casos de uso
- Diagramas de actividad
- Diagramas de estado
- Diagramas de secuencia
- Diagramas de colaboración
- Diagramas de componentes
- Diagramas de despliegue

Hace uso de diversos estándares abiertos para llevar a cabo sus funciones: Struts, Netbeans MDR, ant, Hybernate, Velocity, XDoclet y Maven.

Admite cualquier lenguaje de programación como salida, y admite código propio para la generación de código.

Dadas sus características se puede considerar un Framework para el desarrollo con MDA más que una herramienta en sí misma.

7.3 Codagen Architect 3.2

Esta herramienta de la empresa Codagen permite la transformación de modelos y la automatización de la generación de código partiendo de los modelos CIM.

Como integración con herramientas de modelado, Codagen incorpora plugins para Microsoft Visio, Rational Rose y Borland Together Control.

Admite como entrada ficheros XMI versión 1.1 y ficheros MDL de Rational Rose.

Admite como elementos UML en la entrada los siguientes:

- Diagramas de clases
- Casos de uso
- Diagramas de actividad
- Diagramas de estado

Entre los estándares libres soportados tenemos únicamente Struts, pero como contrapartida, los lenguajes de salida que admite son Java, C#, C++ y Visual Basic, admitiendo también la incorporación de código propio al código generado. También admite valores etiquetados (para ficheros de propiedades), JDO, JMS y EJB.

7.4 IQGen 1.4

El generador de modelos y código IQGen, dela empresa InnoQ está desarrollado en Java.

Incorpora un entorno de modelado muy pobre, que se suple con la admisión de entrada de modelos en formato XMI.

La versión de UML que admite en la entrada es UML 1.3, que contenga los siguientes elementos:

- Diagramas de clases

- Restricciones
- OCL
- Casos de uso
- Diagramas de actividad
- Diagramas de estado
- Diagramas de secuencia
- Diagramas de colaboración
- Diagramas de componentes
- Diagramas de despliegue

Puede generar salida en cualquier lenguaje (J2EE/EJB, C#, Cobol...).

7.5 *OptimalJ*

Este producto de la compañía compuware genera aplicaciones J2EE partiendo de los modelos. Implementa completamente la especificación MDA.

Está desarrollado en Java, lo que le hace portable a cualquier plataforma para su ejecución. Dispone de plugins para los entornos de desarrollo integrado Eclipse y NetBeans.

Admite XMI versión 1.1 tanto para la importación de ficheros como para su salida.

La versión de UML admitida por la herramienta de modelado es UML 1.3, con los siguientes elementos:

- Diagramas de clases
- Restricciones
- OCL
- Casos de uso

- Diagramas de actividad
- Diagramas de estado
- Diagramas de secuencia
- Diagramas de colaboración
- Diagramas de componentes
- Diagramas de despliegue

8 Conclusiones

MDA rompe con el concepto de producción de software artesanal, automatizando las tareas de diseño, desarrollo y en buena parte despliegue de aplicaciones. Gracias a esta automatización, la ingeniería del software puede participar de las ventajas de que gozan otras ingenierías al usar herramientas de soporte a su disciplina.

Centrando la atención sobre la aplicación de MDA a la plataforma Java 2 Micro Edition, en el momento de desarrollo de este documento, no existe ningún soporte específico para esta plataforma en las herramientas MDA existentes. Si bien implícitamente al soporte de Java por las herramientas analizadas, los tipos de datos de J2ME están soportados por las mismas.

La evolución de las herramientas MDA para la plataforma J2ME pasa por el desarrollo de extensiones que restrinjan las transformaciones de modelos PIM a PSI atendiendo a las restricciones de tipos J2ME, la persistencia y la sincronización, en forma de plantillas y mapas de transformación.

Las herramientas MDA que presentan mejores aptitudes para esta tarea serían en este momento ArcStyler y AndroMDA, que gracias a sus sistemas de

cartuchos y plantillas permiten incorporar los cambios necesarios con relativa sencillez.

9 Bibliografía

1. Object Management Group: "Model Driven Architecture (MDA)".
2. Paul Harmon: "MDA, An idea whose time has come."
3. Soley, Mark: "Model Driven Architecture: An Introduction."
4. Johann Oberleitner: "Model Driven Architecture. Model Driven Development".
5. David S. Frankel: "Model Driven Architecture. Reality and Implementation".
6. Pleumann and Haustein: "A model-driven runtime environment for web applications".
7. Lasheras: "Cartuchos MDA en Arcstyler".

10 Referencias Web

1. <http://www.mark-hofmann.de>. Accesible el día 13/04/2005.
2. <http://www.omg.org/mda>. Accesible el día 13/04/2005.
3. <http://www.code-generation.net>. Accesible el día 15/04/2005.
4. <http://www.modelbased.net>. Accesible el día 08/05/2005.
5. <http://www.forum.nokia.com>. Accesible el día 06/05/2005.
6. <http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html>. Accesible el día 06/05/2005.
7. <http://jcp.org/aboutJava/communityprocess/final/jsr172/index.html>. Accesible el día 06/05/2005.
8. <http://www.arcstyler.com>. Accesible el día 12/05/2005.
9. <http://www.andromda.org>. Accesible el día 15/04/2005.
10. <http://www.gentleware.com>. Accesible el día 12/05/2005.

11. <http://www.omg.org/mda/committed-products.html> Accesible el día 12/05/2005.