Desarrollo de software dirigido por modelos: ¿quién quiere escribir código?

Antonio Vallecillo
Universidad de Málaga

Ciudad Real, Abril 2006

Un recorrido por nuestra "historia"

- **Ensamblador**

Registros: AX, BX, …
Segmentos: DS, SS, …
NOP
JMP
CALL
RETURN
Direcciones de memoria
…

- Demasiado bajo nivel
- Poca expresividad
- Programas muy complejos

Luego surgió la prog. estructurada

- Ensamblador
- **Prog. Estructurada**

Estructuras de control:
- **if**
- **while**
- **…**

Abstracción de procedimientos

Lenguajes
- **Fortan**
- **Pascal**
- **C**
- **…**

- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

Aparecieron los objetos

- Ensamblador
- Prog. Estructurada
- **Prog. O. Objetos**

Encapsulacion de datos y comportamiento
Interacciones mediante intercambio de mensajes
Mecanismos:
- **Herencia**
- **Vinculación dinámica**
- **Polimorfismo, …**

Lenguajes:
Eiffel, Smalltalk, C++, Java,…
Analisis Orientado a Objetos
Diseño Orientado a Objetos

- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

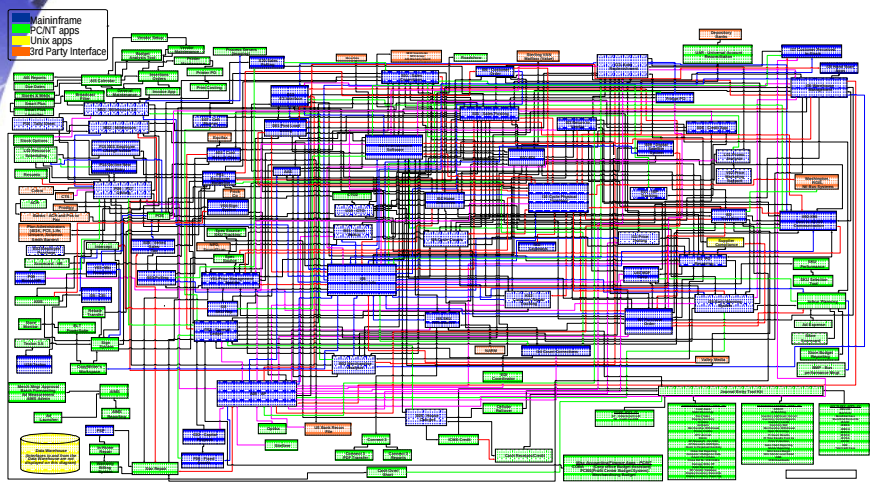
Aparecen los componentes

- Ensamblador
- Prog. Estructurada
- Prog. O. Objetos
- **Prog. O. Componentes**

Distribución
Heterogeneidad
Packaging
Mecanismos:
  Reflexión y Metadata
  Polimorfismo paramétrico
  "Home", Contenedores,…
Lenguajes (IDLs), IDEs
Modelos y plataformas
  J2EE, CORBA/CCM, .NET
CBSE!

- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

El problema es la complejidad



Diseño de una Aplicación Real (Retail)

Otra variante de la POO: los aspectos

- Ensamblador
- Prog. Estructurada
- Prog. O. Objetos
- Prog. O. Componentes
- **Prog. O. Aspectos**

Crosscutting concerns
Nuevos conceptos:
- Aspecto
- Joint point
- Weaving
- ...

Lenguajes O. aspectos
- AspectJ, ...

AOSD!
- Early aspects
- Aspectos y componentes

- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

Y ahora los servicios

- Ensamblador
- Prog. Estructurada
- Prog. O. Objetos
- Prog. O. Componentes
- Prog. O. Aspectos
- **Prog. O. Servicios**

Mayor interoperabilidad
Menor acoplamiento
Alta disponibilidad
Nuevos conceptps
- Web Services
- WSDL, SOAP, UDDI,…
- Semantic Web Services
- BPEL
- “Servicio”
- Service Bus

SOA!

- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

Y después?

- Ensamblador ...
- Prog. Estructurada ...
- Prog. O. Objetos ...
- Prog. O. Componentes ...
- Prog. O. Aspectos ...
- Prog. O. Servicios
- **Prog. O. Eventos**
- **Prog. O. X???**
- **Prog. O. Y???**
- **Prog. O. Z???**

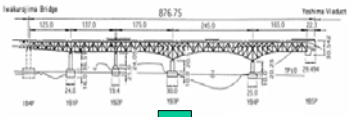
- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

¿Qué hacemos con esto?

- **Es preciso romper ese nudo "Gordiano"**
- **La programación no debe ser el centro de atención. Hay que elevar NOTABLEMENTE el nivel de abstracción**
- **¿Cómo se hace en otras ingenierías más maduras?**
  - Ingenierías civiles (caminos, canales, puertos, …)
  - Arquitectura y construcción
  - Ingeniería aeronáutica y del espacio
  - …

Las ingenierías tradicionales usan "modelos"

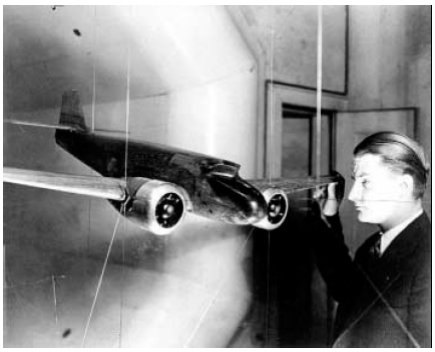
- **Tan antiguos como las Ingenierías (p.e. Vitruvius)**
- **Los ingenieros tradicionales siempre construyen modelos antes de construir sus obras y artefactos**
- **Los modelos sirven para:**
  - **Especificar el sistema**
    - Estructura, comportamiento,…
    - Comunicarse con los distintos *stakeholders*
  - **Comprender el sistema (si ya existe)**
  - **Razonar y validar el sistema**
    - Detectar errores y omisiones en el diseño
    - Prototipado (*ejecutar* el modelo)
    - Inferir y demostrar propiedades
  - **Guiar la implementación**



Características de los modelos

[Selic, 2003]

- **Abstractos**
  - Enfatizan ciertos aspectos, mientras que ocultan otros
- **Comprensibles**
  - Expresados en un lenguaje comprensible por por los usuarios y *stakeholders*
- **Precisos**
  - Fieles representaciones del objeto o sistema modelado
- **Predictivos**
  - Deben de poder ser usados para inferir conclusiones correctas
- **Baratos**
  - Mas fáciles y baratos de construir y estudiar que el propio sistema



Limitaciones actuales de los modelos (de software)

- **Sólo se usan como documentación**
  - Que además no se actualiza!
- **"Gap" entre el modelo y la implementación del sistema**
  - Grandes diferencias semánticas en los lenguajes respectivos
  - No hay herramientas de propagación automática de cambios
    - Cambios en el modelo no se reflejan en el código
    - Cambios en el código no se reflejan en el modelo (el modelo no vuelve a usarse jamás tras la primera implementación)
- **Los distintos modelos del sistema no se armonizan**
  - Suponen vistas de un mismo sistema, pero no hay forma de relacionarlas
  - No hay herramientas de integración de modelos
  - Cada lenguaje de vista tiene una semántica distinta del resto (*)
- **No hay ni lenguajes ni herramientas para manejar modelos**
  - Solo editores, pero no hay "compiladores", "optimizadores", "validadores", "transformadores de modelos", etc.
- **¿Estamos realmente hablando de Ingeniería (del software)??**

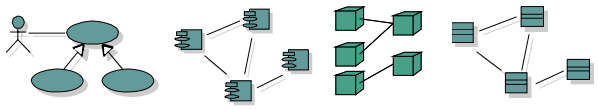
The Remarkable Thing about Software

> **Software has the rare property that it allows us to directly evolve models into full-fledged implementations without changing the engineering medium, tools, or methods**
>
> [John Hogg, 2003]

- Esto facilita enormemente garantizar la fiabilidad entre los modelos y los sistemas producidos, puesto que todos viven en el mismo mundo
- **Corolario: El modelo *es* la implementación.**
- **Salvedad:** Sólo si el modelo contiene toda la información necesaria para producir el sistema

¿Qué es un modelo (de software)?

- A *description* of (part of) a system written in a *well-defined* language. (Equivalent to *specification.*) [Kleppe, 2003]
- A *representation* of a part of the function, structure and/or behavior of a system [MDA, 2001]
- A *description* or *specification* of the system and its environment for some certain *purpose*. A model is often presented as a combination of drawings and text. [MDA Guide, 2003]
- A set of *statements* about the system. [Seidewitz, 2003]
  (Statement: expression about the system that can be considered true or false.)

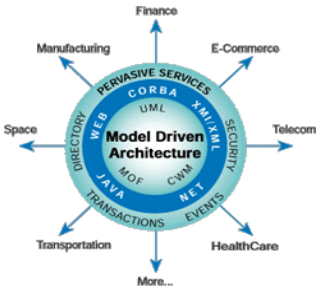


Model Driven Development (MDD)

- **Un enfoque de desarrollo de software en donde las entidades de primer nivel son los modelos y las transformaciones de modelos**
  - **frente a los programas y los compiladores, que constituyeron el paradigma análogo hace treinta años**
- **MDD implica la generación (casi) automática de implementaciones a partir de modelos**
- **En MDD son claves los lenguajes, tanto de modelado como de transformación de modelos. Los modelos son conformes a meta-modelos**
- **MDA es la propuesta para MDD que hace OMG, usando sus estándares:**
  - MOF, UML, OCL, XMI, QVT
  - MOF y UML permiten definir nuevas familias de lenguajes

Model Driven Architecture

- **MDA es una iniciativa de la OMG**
  - Anunciada en el 2000
  - 10 años de plazo para madurar
  - Debe durar al menos 20 años
- **Extiende OMA**
  - Las plataformas middleware pasan a un segundo plano
  - La clave son los modelos



- **MDA aboga por la separación de la especificación de la funcionalidad de un sistema, independiente de su implementación en cualquier plataforma tecnológica concreta**
- **http://www.omg.org/mda**

Ventajas (esperadas) de MDA

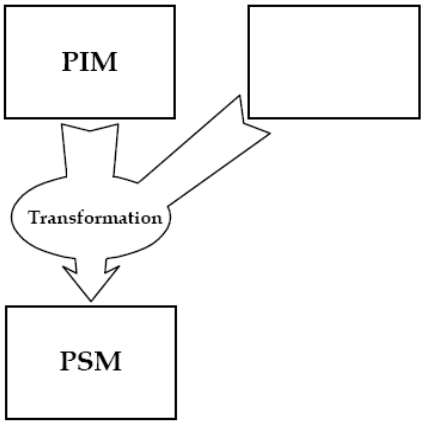
- **Protege la inversión ante los continuos cambios en las tecnológias**
  - Conserva los PIM de una empresa (su modelo de negocio) cuando aparece nuevo middleware
- **Permite abordar mejor sistemas más complejos**
  - Mediante la separación de diferentes aspectos en diferentes modelos
- **Permite la simulación y la implementación automática de los modelos**
- **Permite la integración de sistemas existentes (COTS, legacy systems)**
  - ADM: Architecture Driven Modernization
- **Permite la especificación de los requisitos del sistema independientemente de las plataformas de implementación**
  - MBA: Model-Based Adquisition

Ejemplos de modelos MDA

- **CIM**
  - Modelos de casos de uso que capturan los requisitos del sistema
- **PIM**
  - La descripción de la arquitectura software del sistema, que especifica cómo se descompone la funcionalidad básica del sistema en términos de componentes (arquitectónicos) y conectores
- **PSM**
  - Un modelo de la implementación J2EE del sistema, expresado usando el perfil UML para EJB para representar cómo los componentes (arquitectónicos) del sistema se han de implementar como EJBs
- **Código**
  - Los propios componentes EJBs, sus ficheros de configuración, y toda la información necesaria para realizar el deployment en las máquinas concretas

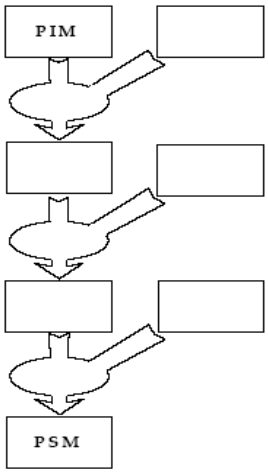
Transformaciones de Modelos

- **Una transformación de modelos especifica el proceso de conversión de un modelo a otro (del mismo sistema)**
- **El patron MDA incluye (al menos):**
  - un PIM,
  - un modelo de Platforma,
  - una transformación, y
  - un PSM.



Aplicaciones sucesivas

- **El patron MDA es normalmente utilizado sucesivas veces para producir una sucesión de cambios:**
  - El PSM resultante de una transformación se convierte en el PIM de la siguiente
  - De esta forma, cada “plataforma” se concentra en un aspecto diferente del sistema, y se aplica ordenadamente
  - Este proceso es modular y ordenado

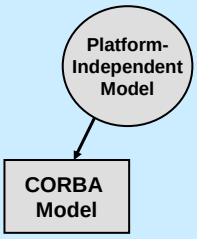


Cómo se construye una aplicación usando MDA

Un modelo detallado, que especificaría la estructura del sistema, las pre- y post-condiciones en OCL, y el comportamiento en Action Semantics Language (por ejemplo)

Se comienza con el *Platform-Independent Model* (PIM) que representa la lógica del negocio y su funcionalidad, independiente de los detalles de la implementación

Se genera el PSM

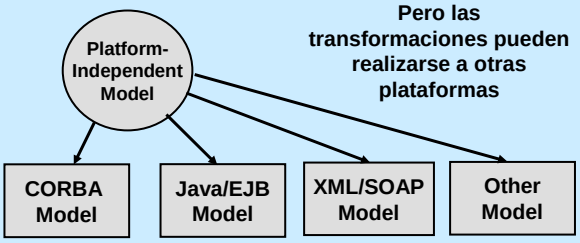


Se escoge una plataforma concreta, y el PIM se transforma al modelo PSM correspondiente a esa plataforma

Las transformaciones pueden ser definidas con QVT, entre los metamodelos origen y destino.

Las transformaciones pueden ser parcial o completamente automatizadas

Generación a múltiples tecnologías



Pero las transformaciones pueden realizarse a otras plataformas

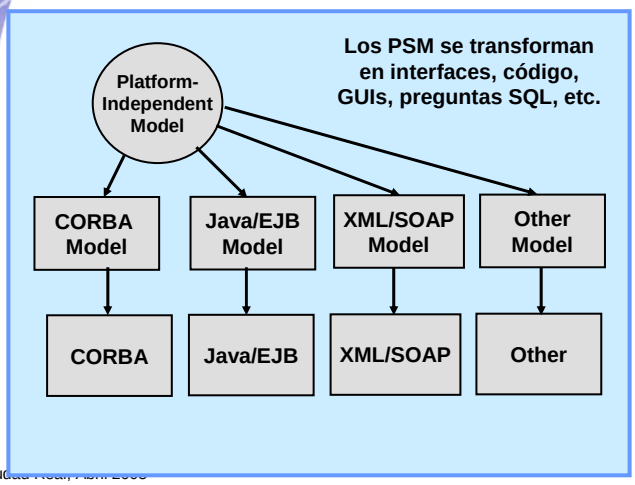
Las transformaciones pueden ser definidas con QVT, entre los metamodelos origen y destino.

Las transformaciones pueden ser parcial o completamente automatizadas

Generación de implementaciones

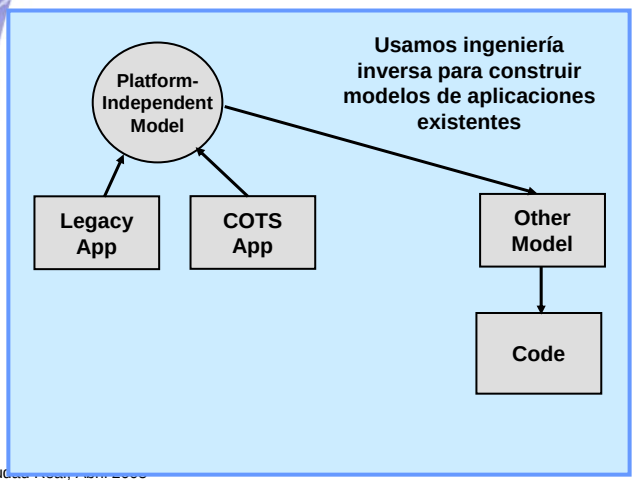
~~Write Once, Run Everywhere~~

Model Once, Generate Everywhere!



Es fácil contar con implementadores automáticos a partir de modelos específicos, pues son de muy bajo nivel

ADM e integración de sistemas



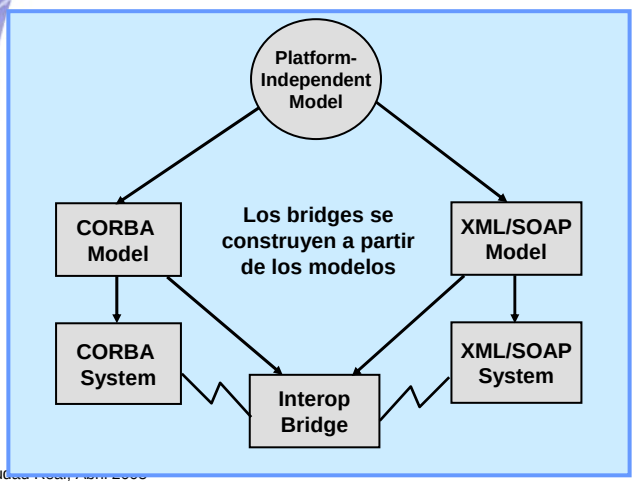
Muy útil para:

(1) **Integración** en nuestra aplicación de COTS, sistemas de terceras casas, y sistemas heredados

(2) **Architecture Driven Modernization**: modernización de sistemas actuales

NASA, DoD, EDF, Banca

Generación de bridges



Los bridges (puentes) pueden generarse de forma automática en la mayoría de los casos, tanto dentro de la propia empresa, como para lograr interoperabilidad entre sistemas de diferentes compañías

Ventajas

- **Cada modelo es independiente del resto**
  - Se definen de forma separada
  - Cada modelo define sus propias "entidades", reside en un nivel de abstracción adecuado, y se expresa en un lenguaje apropiado para el tipo de *stakeholders* interesados en ese tipo de modelo
- **El proceso de desarrollo software se convierte en transformación de modelos**
  - Cada paso selecciona una "plataforma" y transforma uno o mas PIM del sistema en uno (o más) PSM del mismo
  - ...hasta que se llegue a la implementación final del sistema
  - Las transformaciones pueden automatizarse
- **Ganamos modularidad, flexibilidad y facilidad de evolución**
- **Los modelos de la aplicación que capturan la lógica del negocio y la propiedad intelectual se convierten en los principales activos de la empresa, y son independientes de la(s) tecnología(s) en las que serán implementados**

Pero...

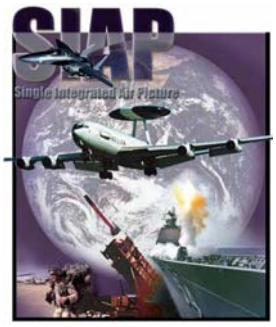
¿De verdad crees que funciona esto del MDD?

- ¿Sinceramente, tu crees en eso de pintar dos cajas y tres líneas y obtener todo el código de tu aplicación?
- ¿Seremos capaces de generar código a partir de modelos, incluso cuando todavía no estamos de acuerdo ni en cómo representar el comportamiento?

- **La mejor respuesta a esta pregunta la dan las experiencias que actualmente han demostrado que (en ciertos dominios de aplicación) MDA no sólo es factible, sino que consigue mejoras espectaculares en la productividad y la calidad de los productos desarrollados**

Éxitos (hasta la fecha)

- **Lockheed Martin (F16 mission computer)**
- **Nortel (Passport)**
- **TRW Automotive**
- **BAE Systems (Stingray torpedo)**
- **US DoD SIAP (Single Integrated Air Picture)**
- **US Goverment Model-based Adquisition**
- **US ERA (Electronic Record Archives)**



- **Usando herramientas y sistemas de:**
  - Kennedy Carter iUML
    - MDA y ADM usando UML ejecutable
  - IO-Software ArcStyler, Compuware Optimal/J, Borland Together, etc.
    - Herramientas MDA de carácter general, muy útiles para aplicaciones distribuidas (J2EE o CORBA), aplicaciones Web, o aplicaciones orientadas a servicios.
  - Kabira's Adaptive Realtime Infrastructure (ARI)
    - Aplicaciones para sistemas distribuidos transaccionales
  - Secant technologies
    - Aplicaciones de extracción de conocimiento

¿Conclusiones?

- **MDD eleva el nivel de abstracción de programas a modelos**
  - Igual que los lenguajes de programación estructurada elevaron el nivel de abstracción del ensamblador
- **MDA es la propuesta de OMG para hacer MDD, usando sus estándares: UML, MOF, XMI, OCL, QVT**
- **Las ideas son buenas, los primeros resultados están aquí (¡y son buenos!) y la dirección parece la adecuada**
- **Algunos peros:**
  - las herramientas y la tecnología que soporta MDA no están del todo maduras
  - La compatibilidad y portabilidad entre modelos no funciona del todo
  - Hay pocas experiencias todavía
- **De todas formas, ¿se plantearía a día de hoy si usar o no lenguajes de programación, frente a ensambladores, para construir sus aplicaciones?**

¡Gracias!

av@lcc.uma.es
http://www.lcc.uma.es/~av