# Separación de conceptos y MDA: Arquitectura de un *framework*

A. M. Reina, J. Torres, M. Toro, and J. A. Álvarez

Dpto. de Lenguajes y Sistemas Informáticos,
Universidad de Sevilla,
Avda. Reina Mercedes, s/n
41012 Sevilla, España
{reinaqu, jtorres, mtoro, juan}@lsi.us.es

**Resumen** La industria del software se tiene que enfrentar tanto al rápido cambio tecnológico como al cambio en los requisitos y las nuevas necesidades de los clientes. Para dar solución al cambio tecnológico han aparecido propuestas centradas en modelos como MDA, mientras que para mejorar la evolución del software ha surgido con gran fuerza la separación avanzada de conceptos, que intenta localizar los cambios y minimizar las dependencias entre aspectos. Este trabajo presenta la arquitectura de un framework para generar aplicaciones basado en modelos, y con separación de conceptos, ya que se propone la definición de lenguajes específicos de modelado para cada uno de los conceptos del sistema.

## 1. Introducción

La industria del software tiene una característica especial que la hace diferente a cualquier otro tipo de industria: la rapidez con la que cambian y surgen nuevas tecnologías y/o nuevas plataformas, a las cuales se tiene que ir adaptando. Así, en muy poco tiempo han surgido vocablos como Java, Linux, XML, HTML, SOAP, UML, J2EE, .NET, JSP, ASP, Flash, servicios web, ... que muchas compañías han tenido que adoptar, bien por exigencias de los propios clientes, bien para poder utilizar herramientas que ya soportaban estas nuevas tecnologías. Con todo este entorno de trabajo, el poder afrontar tanto los cambios tecnológicos como los requisitos variables de los clientes a un coste mínimo, se ha convertido en un punto crucial para la supervivencia de cualquier empresa.

Por otra parte, el desarrollo de las comunicaciones, y sobre todo de internet, ha hecho que muchas empresas ofrezcan nuevos "e-productos" como un valor añadido a sus productos tradicionales, lo cual ha provocado un incremento del volumen de negocio en la red. Pero, sobre todo, ha causado que las aplicaciones web sean cada vez más complejas, pasando de simples repositorios de información a sistemas distribuidos en los que se requieren aspectos importantes tales como la seguridad o la persistencia, y otros fundamentales para los sistemas web como la navegación.

Siendo conscientes de estas necesidades, han surgido propuestas que atacan estos problemas desde distintas perspectivas. Por una parte, la filosofía MDA (Model Driven Architecture) [10] se ideó teniendo como principal objetivo la mejora del proceso de adaptación del software a diferentes entornos tecnológicos. Por otra parte, la separación avanzada de conceptos o programación orientada a aspectos [3] tiene como objetivo mejorar la evolución de las aplicaciones, para ello, define los conceptos por separado y minimiza las dependencias entre ellos, con lo cual los cambios quedan localizados en puntos concretos. Aunque la programación orientada a aspectos es un paradigma que surgió a nivel de programación, pronto se ha trasladado a otras etapas del ciclo de vida, así, se ha acuñado el término *aspectos primeros* ( o *early aspects*)[14] para referirse a los conceptos que se separan en las primeras etapas del ciclo de vida.

En este trabajo se propone una arquitectura para un *framework* generador de aplicaciones que sean capaces de enfrentarse, con un bajo coste, tanto al cambio tecnológico como a la propia evolución del software. Para conseguir estos dos objetivos se busca sacar partido tanto de las ventajas que proporciona la separación avanzada de conceptos como de las de MDA. El resultado de estas decisiones es un *framework* basado en modelos y que trabaja con diferentes lenguajes de modelado, uno para cada uno de los aspectos de alto nivel que componen el sistema.

Este artículo se estructura de la siguiente forma: En la sección 2, se explican los detalles de las propuestas que han servido de inspiración para nuestro *framework*, la separación avanzada de conceptos y MDA. En la sección 3, se da una relación de otros trabajos que de alguna manera tienen relación con nuestra propuesta, para continuar con la descripción de nuestra propuesta en el apartado 4. Por último, concluimos el artículo y apuntamos nuestras líneas futuras de trabajo.

## 2. Los fundamentos de la propuesta

En este trabajo se realiza una propuesta de arquitectura para un *framework* generador de aplicaciones conducido por modelos. Los dos objetivos principales a cubrir a la hora de diseñar la arquitectura de este *framework* son:

- Minimizar los costes de adaptación a nuevos entornos tecnológicos.
- Minimizar la influencia de un cambio de requisitos en las distintas partes de una aplicación.

Para alcanzar el primer objetivo se decidió aplicar la filosofía MDA (Model Driven Architecture), una propuesta de la OMG (Object Management Group) para el desarrollo de software basado en modelos; mientras que el segundo objetivo conduce directamente a la aplicación del principio de la separación de conceptos. Para ello se propone una arquitectura lógica de aplicaciones en capas, de forma que cada capa se encarga de tratar con un concepto distinto. Esto permite modelar de forma separada cada uno de estos niveles.

En los siguientes apartados de esta sección se describen con un poco de más detalle estas dos filosofías, sobre todo, para determinar las ventajas de cada una de ellas, y lo que pueden aportar al *framework*.

### 2.1. MDA(Model Driven Architecture)

MDA [10] es un *framework* para el desarrollo software definido por la OMG. La base de esta propuesta es la importancia de los modelos en el proceso de desarrollo software. Esta propuesta de la OMG separa por una parte, la lógica subyacente tras una especificación, y por otra, las propiedades particulares de los *middlewares* en los que la aplicación va a ser desplegada. Por lo tanto, se puede decir que MDA proporciona:

- Portabilidad.
- Interoperabilidad entre plataformas cruzadas.
- Independencia de la plataforma.
- Especificidad del dominio.
- Productividad.

Para alcanzar estos objetivos, MDA se basa en la transformación de modelos. Así, un posible ciclo de vida utilizando esta propuesta podría ser el representado en la Figura 1. Como se puede apreciar es un ciclo de vida muy similar al ciclo de vida tradicional, la diferencia radica en los artefactos que se crean durante el proceso de desarrollo. Estos artefactos son modelos y constituyen el núcleo de MDA. Así, en la figura se define un primer modelo con un elevado nivel de abstracción e independiente de cualquier tecnología de implementación, que se conoce como PIM (Platform Independent Model).

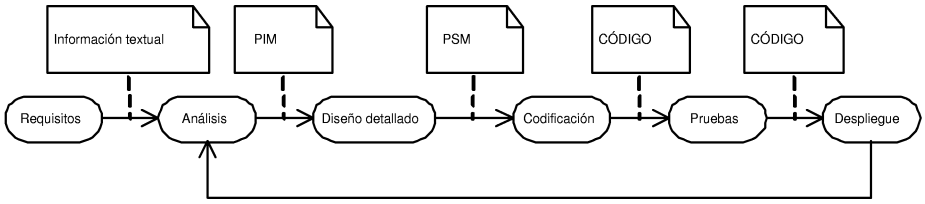


**Figura 1.** Ejemplo de ciclo de vida adaptado para MDA

El siguiente paso es trasformar el PIM en uno o varios PSMs (Platform Specific Models) o modelos específicos de la plataforma. Finalmente, habrá que realizar la transformación de cada PSM a código. Esta transformación no es demasiado compleja, ya que el PSM es un modelo que está bastante cercano a la máquina. Así, se puede resaltar que las nuevas actividades propuestas en este ciclo de vida adaptado a MDA:

- Especificar un sistema de forma independiente a la plataforma que lo soporta.
- Especificar plataformas.
- Elegir una plataforma particular para el sistema.
- Transformar las especificaciones del sistemas en unas especificaciones para una plataforma particular.

### 2.2. Separación avanzada de conceptos

La separación de conceptos es uno de los pilares de la ingeniería del software. Esta consigna se basa en el principio conocido como "divide y vencerás". Así, apoyándose en este concepto ha surgido un nuevo paradigma para descomponer sistemas conocido como separación avanzada de conceptos.

Esta propuesta afirma que es más fácil desarrollar programas si especificamos los diferentes conceptos o áreas de interés que lo forman, y posteriormente, se componen las soluciones parciales para resolver todo el sistema. Así, se puede obtener una separación de conceptos "limpia" que mejorará la comprensión del software y reducirá su complejidad, ya que cada concepto se trata en el sitio justo. Así, en la bibliografía se han estudiado ampliamente conceptos como la sincronización y la distribución [7], pero también otros como la seguridad o el interfaz de usuario. De esta forma, las plataformas de componentes basadas en contenedores, tratan estos conceptos, que se conocen en este ámbito como *servicios de infraestructura*. En la Figura 2, se muestra un esquema en el que se muestra como la separación de conceptos parte de los requisitos, y se mantiene durante el ciclo de vida del proyecto. Este esquema también será importante a la hora de definir nuestra propuesta.

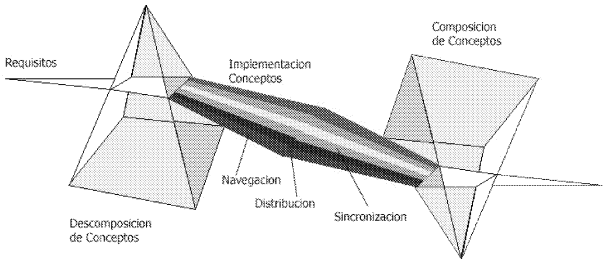


**Figura 2.** Desarrollo propuesto por la separación de conceptos

## 3. Trabajos relacionados

Si hacemos un análisis de las propuestas relacionadas con nuestro trabajo, podemos dilucidar, claramente dos vertientes. Por una parte, existen una serie

de trabajos que provienen de investigadores en el área de MDA, y en los que se vislumbran elementos de la orientación a aspectos. Pero, por otra parte, existen un conjunto de propuestas de investigadores con bagaje en ingeniería web y que se están dando cuenta de las ventajas de aplicar al desarrollo de aplicaciones web algunas de las propuestas de MDA.

Si nos fijamos en el primer grupo de propuestas, aquellas que provienen de MDA, podemos destacar el proyecto GMT (Generative Model Transformer) [12], que surgió a partir del *workshop* '*Generative Techniques in the context of MDA*' [11] celebrado junto con la conferencia OOSPLA 2002. GMT es una iniciativa de código abierto para construir una herramienta basada en MDA que permita trabajar con modelos independientes de la plataforma, modelos de descripción de plataformas, transformación de texturas, y refinamiento de transformaciones. En la Figura 3, se muestra un diagrama de actividad UML con las actividades propuestas en el desarrollo de una aplicación con la herramienta construida por GMT. Esta propuesta trabaja con los términos:

- Modelo Independiente de Plataforma.
- Modelo Específico de Plataforma.
- Modelo de Descripción de Plataforma.
- Textura. Una textura es una especificación de un patrón de forma no ambigua y en un formato basado en UML.
- Transformación de texturas.

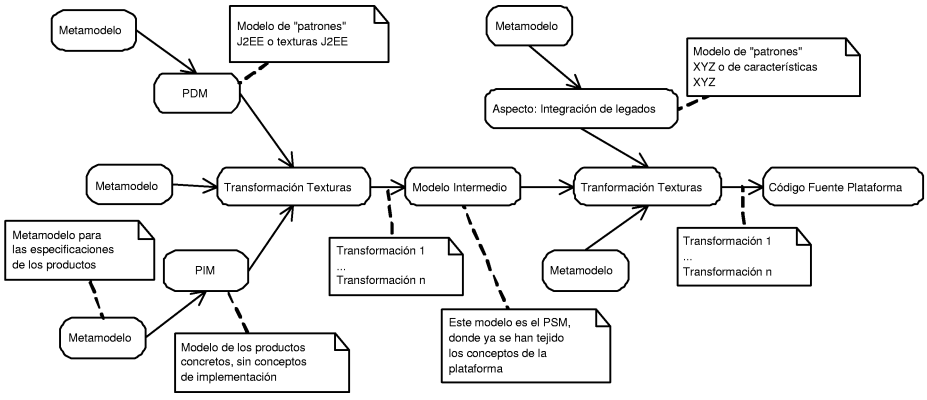


**Figura 3.** Proceso basado en MDA para utilizar GMT

En la Figura 3, se pueden ver, de izquierda a derecha, las actividades necesarias para desarrollar una aplicación utilizando GMT. Como se puede comprobar en la figura, en esta propuesta, la separación de conceptos se hace a nivel de PSM, es decir, de modelo específico de plataforma.

En cuanto a las propuestas relacionadas con investigadores provenientes del campo de la ingeniería web tenemos que muchas de ellas están abogando por metamodelos que sean capaces de adaptarse a los propios cambios en sus lenguajes de modelado. Así los autores de W2000 han sido uno los primeros en tomar algunas ideas de MDA y en [1] proponen la creación de una herramienta meta-CASE (Jweb3), de tal forma, que esta propia herramienta sea capaz de adaptarse a los cambios que van surgiendo en su propia metodología y en su propia notación. De la misma forma, y haciéndose eco de esta primera propuesta, el grupo de trabajo de UWE proponen la construcción de un metamodelo común a cualquier metodología de desarrollo de aplicaciones de sistemas web [6].

Mientras los autores de OO-H proponen una extensión de la arquitectura de MDA llamada WebSA [8] para complementar la especificación de aplicaciones web.

## 4. Nuestro framework

En este trabajo se realiza una propuesta de arquitectura para un framework generador de aplicaciones conducido por modelos, con separación de conceptos. Los principales objetivos a cubrir al definir la arquitectura del *framework* fueron:

- Que no fuera demasiado costoso el hecho de cambiar de tecnología durante el desarrollo de la aplicación, ya que el mantenerse en el mercado muchas veces depende de la capacidad de adaptación a los cambios.
- Poder enfrentarse con poco coste a los cambios de requisitos por parte de los clientes.

Para abordar el primer objetivo se determinó la adopción de una arquitectura basada en la la filosofía MDA, ya que esta filosofía facilita el cambio de plataforma al trabajar con distintos niveles de modelos (PIMs y PSMs). Mientras que para cubrir el segundo, se propone aplicar el nuevo paradigma conocido como separación avanzada de conceptos o programación orientada a aspectos, ya que los diferentes conceptos que componen el sistema, estarán bien localizados, se minimizarán su interdependencias, y por tanto, será más fácil realizar cualquier tipo de modificación.

Para poder trabajar con los conceptos de forma separada, se propone la definición de lenguajes de modelado específico de dominio, uno por cada uno de los aspectos que se vayan a separar. Así, las aplicaciones generadas por el *framework* tendrán una estructura similar a la mostrada en la figura 4. En ella se representa la estructura de una aplicación web distribuida genérica, en la que se tratan por separado los aspectos de persistencia, distribución, seguridad, navegación e interfaz de usuario. Como se puede apreciar en la figura, a diferencia de GMT que empieza a separar aspectos a nivel de PSM, en este trabajo se propone la separación de conceptos a nivel de PIM, elevando así el nivel de abstracción gracias a la definición de lenguajes de modelado de dominio específico para cada aspecto.

Para definir estos lenguajes específicos de modelado se pueden utilizar cualquiera de los dos mecanismos de extensión de UML, bien perfiles, bien metamodelos. Dependerá de la propia naturaleza del aspecto el elegir uno u otro. En [15], se propone una guía para elegir la técnica de extensión más adecuada.

El utilizar estos lenguajes de modelado específicos del dominio del aspecto permite razonar mejor sobre los distintos aspectos del sistema, ya que se pueden tener expertos centrados solamente en modelar la persistencia, la distribución o la propia navegación. Es más, un cambio en el modelo de navegación no tiene por que afectar a los demás modelos.

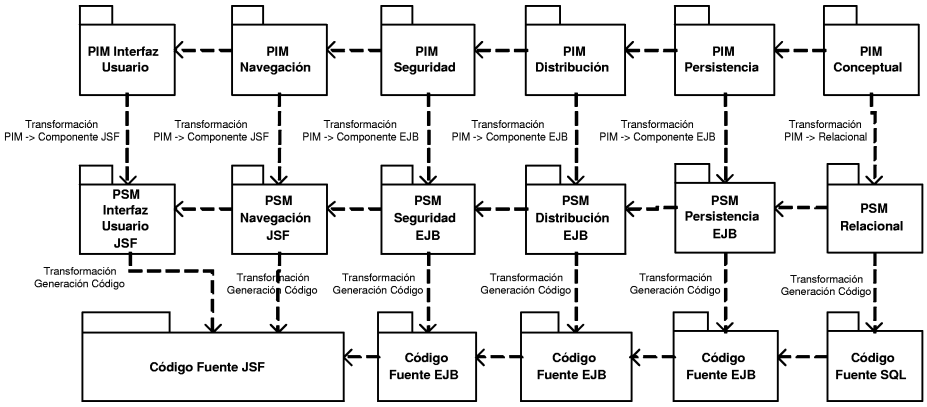


**Figura 4.** Tres niveles de modelos para una aplicación generada por el framework

Una vez que se tienen los PIM para cada una de las capas, habrá que aplicar transformaciones para obtener modelos específicos de la plataforma. Estos PSM's ya serán dependientes de la implementación que se vaya a hacer del sistema. Por último, habrá que aplicar transformaciones para generar el código fuente correspondiente a la plataforma concreta de implementación.

En la figura 4, se representan tres plataformas distintas a la hora de implementar la aplicación: JSF (Java Server Faces) [2], para las capas de interfaz y navegación, EJB (Enterprise Java Beans) [9],para el resto de las capas y una base de datos relacional para la capa de acceso a los datos. Esto no quiere decir que el *framework* esté obligado a trabajar con estas plataformas concretas.

Java Server Faces proporciona un conjunto de interfaces para representar componentes de interfaz de usuario, gestionar su estado, gestionar eventos, realizar la validación de las entradas de usuario y gestionar la navegación de páginas y la internacionalización. Es una propuesta que desacopla los componentes de su presentación de tal forma que los componentes se pueden dibujar de diferentes formas y en dispositivos diferentes. Para cambiar la implementación del interfaz de usuario y la navegación a otra plataforma, por ejemplo, Cocoon [13] o páginas JSP (Java Server Pages), solamente tendríamos que cambiar las trans-

formaciones de PIM a PSM, los lenguajes de modelado específicos de la plataforma y las transformaciones de PSM a código fuente de las capas de navegación e interfaz de usuario, el modelo independiente de la plataforma permanecería exactamente igual.

De la misma forma, si quisiéramos cambiar de plataforma de componentes, y pasar de EJB a COM+ [4], solamente habría que cambiar las transformaciones de los PIM de los niveles inferiores (seguridad, distribución, persistencia y funcionalidad básica) para generar PSMs para esta otra plataforma de componentes. A partir de estos PSMs sería más o menos inmediato generar el código fuente.

### 4.1. Componentes necesarios para el framework

Para acometer esta propuesta, y siguiendo el esquema proporcionado por MDA [5] necesitaremos los componentes representados en la Figura 5: un editor de modelos, un editor de código, un repositorio de modelos, un validador de modelos, un editor de definición de transformaciones, y un repositorio de transformaciones.

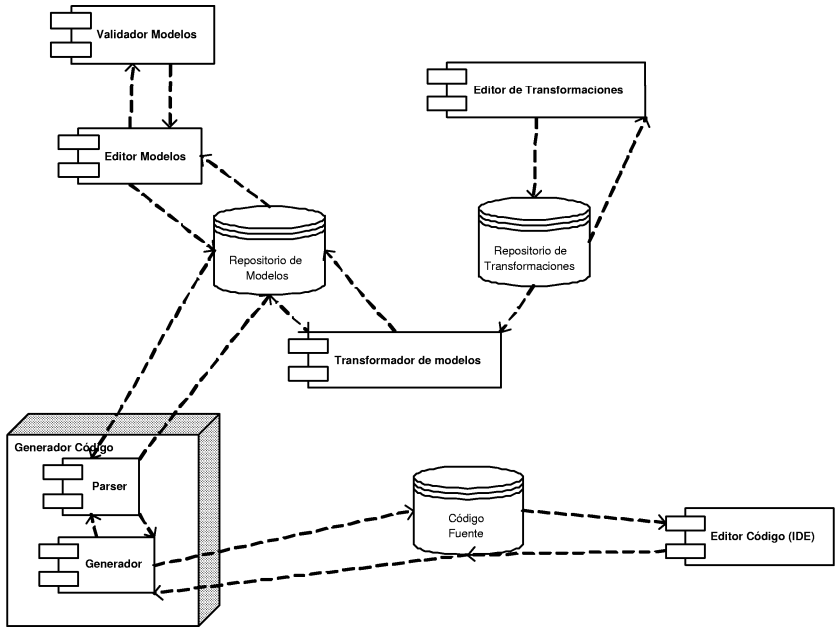


**Figura 5.** Componentes necesarios para el framework generador de aplicaciones

- El *editor de modelos* será una herramienta en la que se podrán construir modelos y modificar los ya existentes, con una funcionalidad bastante parecida a las actuales herramientas CASE. Esta herramienta deberá permitir

almacenar los modelos en un formato estándar. En este caso, el elegido es XMI (XML Metadata Interchange), uno de los estándares en los que se basa MDA. XMI se utiliza como formato de intercambio estándar de modelos UML.

- El *validador de modelos* se encargará de comprobar mediante una reglas (bien predefinidas, bien definidas por el usuario) de asegurar que el modelo es adecuado para ser usado con transformaciones. Este paso es necesario porque los modelos utilizados para generar otros modelos deben estar extremadamente bien definidos.
- El *repositorio de modelos* que será una base de datos en la que se almacenen los modelos en un formato estándar (en este caso, con XMI). Además de los modelos, también contendrá el repositorio de metamodelos y perfiles que definen cada uno de los aspectos que se va a definir en el sistema.
- El *editor de transformaciones* será una herramienta para construir transformaciones y modificar las ya existentes. Estas transformaciones deberían poder almacenarse en un formato estándar. En la actualidad, no hay ningún estándar para definirlas, pero la OMG está trabajando actualmente en un lenguaje para escribir definiciones de transformaciones denominado QVT (Query, Views, and Transformations). El documento RFP (Request for Proposals) [16] para dicha propuesta ya se ha publicado, aunque aún es un trabajo en progreso y no hay una versión estable o cuasi-definitiva de QVT.
- El *repositorio de transformaciones* será una base de datos donde se almacenen las definiciones de las transformaciones en un lenguaje estándar.
- El *transformador de modelos* tomará como entrada uno o varios modelos independientes de la plataforma (PIM), y una o varias definiciones de transformación, y dará como resultado uno o varios modelos específicos de la plataforma (PSM) a partir de los cuales se podrá generar el código fuente de la aplicación deseada. En este caso, el transformador de modelos hará las funciones de *weaver*, ya que tendrá que componer los modelos.
- El *generador de código* se encargará de generar el código de la aplicación a partir del o los PIMs y estará formado por dos componentes, un analizador o *parser* y un generador.
- Por último, el *editor de código* (IDE), tendrá la misma funcionalidad que cualquier IDE que conocemos actualmente, es decir, compilar, depurar y editar código, con la peculiaridad que de debe ser bi-direccional, es decir, si al modificar un modelo se modifica el código asociado a él, al modificar el código, puede que el modelo correspondiente se vea afectado, y se tenga que modificar en consecuencia.

## 5. Conclusiones y trabajos futuros

En este artículo se ha presentado una propuesta de arquitectura para implementar un *framework* para generar aplicaciones basado en MDA y con separación de conceptos. A diferencia de algunas de las propuestas vistas en el apartado de trabajos relacionados, en la que los aspectos se incorporan a nivel de PSMs, nuestra propuesta eleva el nivel de abstracción de los aspectos, tratándolos a nivel

de PIM. También, se han incorporado conceptos, que ampliamente se tratan en el desarrollo de sistemas web.

Una de las ventajas principales de tener los modelos separados es el hecho de tener localizados los cambios. Para esta propuesta, nos hemos basado en los conceptos se han tratado más ampliamente en las plataformas de componentes, pero en un futuro queremos estudiar la incorporación de nuevos aspectos.

Además de la definición del *framework*, se busca ahondar, sobre todo en cuestiones relativas a la separación de conceptos, tales como si el orden de los aspectos es importante a la hora de aplicar las transformaciones, las incompatibilidades entre los mismos.

## Referencias


1. L. Baresi, F. Garzotto, L. Mainetti, and P. Paolini. Meta-modeling techniques meet web application design tools. In R. D. Kutsche and H. Weber, editors, *Proceedings of the FASE 2002, LNCS 2306*, pages 294–307. Springer-Verlag, 2002.
2. P. S. Bhogill. An introduction to java server faces. *Java News Brief*, Aug 2003.
3. *Workshop on Aspect Oriented Programming (ECOOP 1997)*, June 1997.
4. G. Eddon and H. Eddon. *Inside COM+*. Microsoft Press, 1999.
5. A. Kleppe, J. Warner, and W. Best. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison Wesley, 2003.
6. N. Koch and A. Kraus. Towards a common metamodel for the development of web applications. In J. M. Cueva Lovelle, editor, *Proceedings of the ICWE 2003, LNCS 2722*, pages 497–506. Springer-Verlag, 2003.
7. C. V. Lopes. *D: A Language Framework for Distributed Programming.* PhD thesis, College of Computer Science, Northeastern University, 1997.
8. S. Melia, C. Cachero, and J. Gomez. Using MDA in web software architecture. In *2nd OOSPLA Worshop on Generative Techniques in the Context of MDA*, 2003.
9. Sun Microsystems. Enterprise javabeans specification, version 2.0, 2001.
10. OMG. Mda guide version 1.0. Technical Report omg/2003-05-01, OMG, May 2003.
11. *Workshop on Generative Techniques in the Context of MDA held in Conjunction with OOSPLA 2002*, 2002.
12. Generative Model Transformer Project. http://www.eclipse.org/gmt/.
13. The Apache Project. The cocoon home page. http://xml.apache.org/cocoon/, 2002.
14. Awais Rashid, Ana Moreira, and Joäo Araújo. Modularisation and composition of aspectual requirements. In M. Akşit, editor, *Proc. 2nd Int' Conf. on Aspect-Oriented Software Development (AOSD-2003)*, pages 11–20. ACM Press, March 2003.
15. A.M. Reina, J. Torres, and M. Toro. Towards developing generic solutions with aspects. In *Proceeding of the Workshop in Aspect Oriented Modelling held in conjunction with the UML 2004 Conference*, oct 2004.
16. Tata Consultancy Services. Revised submission for mof 2.0 query / views / transformations rfp. version 1.1. Technical report, Tata Consultancy Services, aug 2003.