A Model Driven Architecture Approach to Web Development

Alejandro Gómez Cuesta[1], Juan Carlos Granja[1], and Rory O'Connor[2]

[1] Software Engineering Department, University of Granada, Spain
elales@gmail.com, jcgranja@ugr.es
[2] School of Computing, Dublin City University, Ireland
roconnor@computing.dcu.ie

Abstract. The rise of the number and complexity of web applications is ever increasing. Web engineers need advanced development methods to build better systems and to maintain them in an easy way. Model-Driven Architecture (MDA) is an important trend in the software engineering field based on both models and its transformations to automatically generate code. This paper describes a a methodology for web application development, providing a process based on MDA which provides an effective engineering approach to reduce effort. It consists of defining models from metamodels at platform-independent and platform-specific levels, from which source code is automatically generated.

Keywords: Model Driven Development, Web Engineering, Software Engineering, Agile Development

1 Introduction

The requirements for web applications involve a great diversity of different services; multimedia, communication and automation, which reside in multiple heterogeneous platforms. The development of such systems is a difficult task due to the large number of complexities involved. Accordingly there is a need for solid engineering methods for developing robust web applications.

The development of most systems is based on models as abstractions of the real world. In software engineering the Unified Modelling Language (UML) is becoming the standard for Object-Oriented (OO) modelling. While OO models traditionally serve as blueprints for systems implementation, the Model-Driven Architecture (MDA) [11], [10], which is a software design approach, promotes the usage of models throughout the entire development process. Such an approach provides a set of guidelines for specifying, designing and implementing models.

Currently there are many methodologies to define web applications which are based on models such as: OO-H [6], UWE [9], ADM [5] or WebML [3]. These methodologies have different levels of abstraction, but all require spending much time on defining conceptual and usually do not take into account tangible elements such as pages. However, other approaches such as agile development are not so abstract.

MIDAS [12] is an approach to create web applications which merges models and agile development. The MDA approach defines three levels of model abstraction and a way to transform them:

- **CIM** (Computational Independent Model): A model of a system from a computation independent viewpoint where details of the structure of systems are not shown. It is used to represent the specific domain and the vocabulary for the specification of a system.
- **PIM** (Platform Independent Model): A model of a software or business system that is independent of the specific technological platform used to implement it.
- **PSM** (Platform Specific Model): A model of a software or business system that is linked to a specific technological platform (programming language, operating system or database).
- **QVT** (Query, View, Transformations): A way to transform models (e.g., PIM can be translated into PSM).

The application of both MDA and agile methods to web applications development can help to build better and fast systems in an easier way than applying traditional methods [2]. MDA supports agile development, providing an easy way to create prototypes through automatic code generation.

We have chosen the MIDAS [12], approach as our starting point, as it is both a development process based on an agile process and an architecture for web applications based on models. Using an agile approach prioritises the client-developer relationship and a using model-based approach improves communication between client and developer models, which allows the system to be seen from a higher level of abstraction.

The basic structure of web applications consists of three tiers: the Client, the Server and the Data store. In MIDAS, there are: graphical user interface (GUI), business logic and persistence, where each tier can be implemented with a different programming language, but utilising the same data structures. Thus MDA can be applied to this kind of software: joining all the data structures into a PIM, we can define the problem as only one and afterwards, splitting it into each tier. Thus we can define one problem and obtain code for different platforms / programming languages.

The work presented in this paper proposes a methodology for agile web applications development based on models. We provide a model-driven development method for the specification and implementation of web applications through our own metamodels. Our approach establishes a methodological framework by automating the construction of high-quality web applications (PIM) in a productive way, starting from a UML based representation for the, then an automated transformation method for the refinement of engineering models into technology models (PSM), and finally an automated method for the generation of platform specific code. Our approach has several advantages, e.g. the diagrams to use are very known and the metamodels are very simple so developers has not to learn new techniques. In addition, the automatic code generation provides boosts development time.

In this paper we use the case study of a web bookshop which allows the purchaser to search books by author, subject or title from a database. Once the user has a listing with the result, he can select several books which will be added to a cart. The user can manage this cart adding or deleting his selected books as times as necessary. Finally, the user can buy those books which are in the cart, and in this moment is when he has

to insert his personal details in the system. We construct a web system for this example using own described methodology.

2 MDA-Based Approach for web applications

The development of a web application system implies the use of many different technologies in order to satisfy all user requirements. Usually these technologies provide low abstraction level constructs to the developer. Therefore, applying a MDA approach to web applications involves bridging an abstraction gap that must deal with the technology heterogeneity. Thus, MDA approach raises the abstraction level of the target platform and, therefore, the amount of code is clearly reduced. We reach a balance between model based and agile development. Although we are not completely fulfilling the requirements of MDA (that of abstracting as much as possible) we are adding some level of work schema to an agile development, even though this kind of development is not based on restrictions when writing code.

Accordingly our proposed approach for web applications development is based on:

1. Web application PIM.
2. Web application PSMs.
3. Transformations from PIM into PSM.
4. Code generation from PSMs.
5. Validation of this approach.

We can see this methodology is purely based on MDA. As we will see the metamodels used are not very complex and therefore we are also taking into account a certain level of agile development.

2.1 The Web application PIM

Our PIM is intended to represent web application which consists of services. A service can be defined as functionalities offered by the system and whose results satisfy some specific needs from a user.

Our approach consists in defining at an early stage the services we want our application to fulfil. The second step is to refine those services into others more detailed. Finally, for each detailed service a set of messages is defined. These messages specify the relationship between the previous defined services and the final system. Moreover, we need to set up a class model collecting the elements that have to be modelled into the web application and can be used into the previous models.

So, our methodology define four stages - or from an MDA perspective - four different submodels, which all together shape our web application platform-independent model:

- **Class model**: a typical definition of the problem by means of classes.
- **Conceptual user service model**: defines the actors and services for a web application.
- **Extended service model**: details the functionality of a conceptual user service dividing it into extended services.

- **System sequence model**: establishes how each actor acts on the web application through messages or callings to other extended services within a extended service

The conceptual user and extended services are definitions which are taken from MIDAS, but which have simplified the conceptual user service model and added new features to the extended service model to support our system sequence model as well as our class model.

2.1.1 Class PIM

This is a typical class model, but using many simplifications. There is neither encapsulation for attributes (all are public), complex associations between classes nor interfaces. It is a simple metamodel of a class model which is useful for the agile development and keeping the MDA process.

2.1.2 Class PIM

This model is similar to a use case model where the system services as well as who uses them are represented. System services are called Conceptual User Services (CUS) and are depicted like a use case, where an actor is somebody or something who executes a CUS. Actors and services connect themselves the former to the latter.

Figure 1 shows an example CUS. The execution of the *BuyBooks* service consists of: searching some books, adding them into a cart. The execution of a CUS is related to the execution of a web by their actors. This concept is a new feature on conceptual user service that MIDAS does not include.

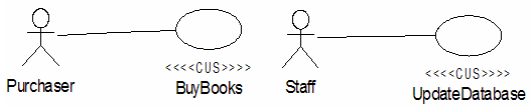

Fig. 1. Conceptual user service model.

In Figure 2 we can see the metamodel of this model. Actors are instances of the class called Actor. CUSs are instances of the *ConceptualUserService* class. As the *ConceptualUserService* class is a subclass of the abstract class *UseCase*, an unspecified number of actors can be associated to CUS's.

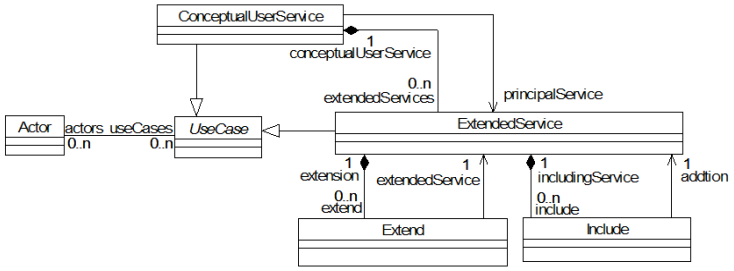

Fig. 2. Metamodel for CUS and ES models.

2.1.3 Extended Service Model

Each CUS is related to an extended service model. This model breaks up each CUS into simpler elements called Extended Services (ES) which are usually either a data input or output. Figure 3 shows the ES model associated to the *Purchaser* actor depicted in Figure. We can see the functions such actor performs within the CUS called *BuyBook*; the ESs describe how the actor buys a book, managing them by means of a cart. The *AddBookToCart* ES is called by both *UpdateCart* and *OrderBooks* due to the <<*include*>> associations which go out from them, and *AddBookToCart* ES calls to *SearchBook*. On other hand, we have that *SearchBook* ES could be replaced with *SearchForAuthor*, *SearchForSubject* or *SearchForTitle* due to the <<*extends*>> associations.

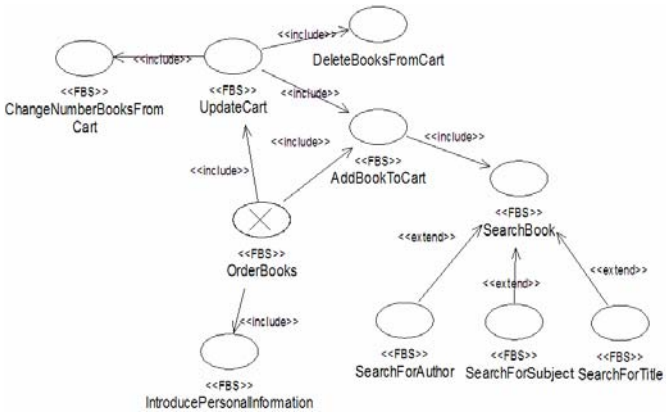

Fig. 3. ES model for the purchaser.

The ES model execution is similar to that of the CUS. It starts from the principal service and it continues through the <<*include*>> relations to other ESs depending on the associated system sequence. If the user executes an ES where <<*extend*>> associations are coming in, as *SearchBook* ES, means that any of those ESs can be executed instead; the final selection depends on the user. As ESs are similar to functions, when one is ended, the execution returns to the ES which made the call.

Using the example in Figure 3, a user executes the first extended service, *OrderBook*. Later he/she can select either *UpdateCart* or *AddBookToCart* extended services. From *UpdateCart* can change the number of books selected for a specific one and he/she can delete one, as well as to add other book selecting the appropriate extended service. From *AddBookToCart* the user can makes a search, and this search can be done for: author, subject o title. This model extends the requirements specified in the CUS model to have a second level of abstraction. Using this model, it is easy to check how many parts have a specific CUS. In Figure 2 we have the metamodel of the

ES model. We can see the relation between CUSs and ESs, where the latter owns several elements of the former, besides the relation *principalService*.

2.1.4 System Sequence Model

A System Sequence (SS) model describes the functions performed by an ES. This model is simpler than the legacy sequence models which exist in UML. In the diagrams of these models there are two vertical lines: first one for actors and the second one for objects, but in our case we use only one: the *System*. Figure 4 shows an example of a system sequence model, in this case for the *OrderBook* extended service Figure 3.

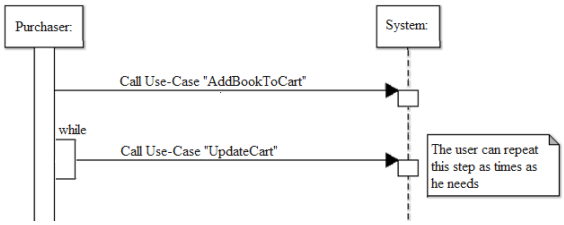

Fig. 4. SS Model for OrderBooks ES.

Actors from a specific ES can send two types of messages:

- Functions with return values and/or parameters, where both are objects which belong to classes defined in the class model. The functions are algorithms which are not modelled. They are just a reference for code generation, because they will transform into methods in Java code.
- Calls to other extended services.

Note in Figure 4, the second call to other extended service on the left has a little box with the text while above. It means that this call can be performed as times as the user wants. We have defined three kinds of way of execution for the messages on a box:

- Sequential: is the normal order.
- Repetition: messages are always repeated while the user wants.
- Choose one: from all messages the user selects one.

It is possible to have multiple levels of execution, so for example, we could have several messages upon a *choose one* execution and upon that one, other messages as well as other *repetition* executions. Therefore we define a specific order for the execution. MIDAS defines an activity diagram for this purpose, but we have the same idea using the previous execution concept. Using our approach we define a number of messages which are closer to implementation without getting into this low level favoring the agile development.

Figure 5 shows the system sequence metamodel. We can note that every ES has only a message sequence. An object of a *MessageSequence* class can have just one type of elements: Objects from Message class (functions) or *MessageSequence* objects (calls to other Ess). The attribute *MessageSequenceDefault* indicates what order of execution owns such sequence: sequential, repetition or choose one.

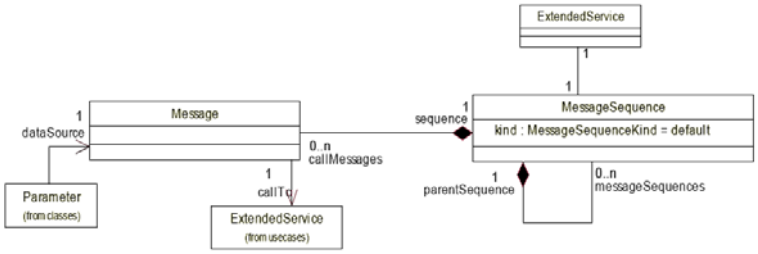

Fig. 5. SS meta-model.

2.2 PSM

MIDAS defines web applications with three-tiers: the GUI, the business logic and the content. In our case we have one PSM for each part, as each part is a platform-specific model. We have chosen SQL for the content, Java servlets for the business logic and a Java servlet application for the GUI.

2.2.2 The Logic Business and Content PSMs

The logic business PSM is a class model, but in this case is the same Class-PIM but adding interfaces. The content model is a database model which has been taken out from [8]. Summarizing, we have tables which consist of columns, where each column owns a SQL data type. Tables have both primary keys and foreign keys and both are related to a column.

2.2.1 The Web application PSM

A web is the place where users or actors connect to access the web services which are related to CUSs. A service consists of a number of web pages which are executed in a certain order. Each page is related to a ES. Every page has associations to others which are next (*outgoings*) and previous (*incomings*) ones; then, crawling among these pages a concrete service is executed. As each service has a first page we know from where that listing of services has to start. If we need to define a page which has the same execution than other one already defined, we can associate the first one to the second one, e.g., if we need to request certain information twice in different points, we create one page and the other just *calls to* this one.

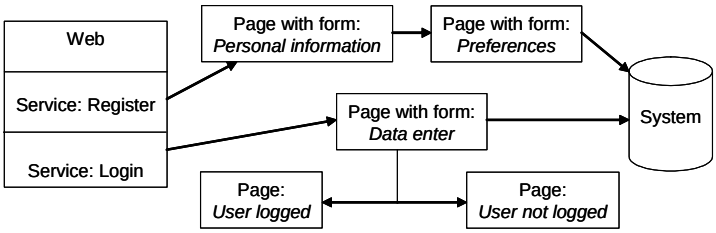

Fig. 6. Web application example.

Figure 6 shows an example of web application. We want to create a web application with just two services: register and log in the web application. The first service, register, has two pages: the first one is where the user writes down his personal information and the second one does his preferences. Once he writes his personal information and tries to go to the second page, a web operation saves his personal information into a temporal container and, when he finishes inserting his preferences, other web operation sends both personal information and preferences to a data base. On the other hand, we have the second service, log in, where a form is showed to the user who can fill it with its information for accessing; other web operation is executed. If the information provided is not of any user, he is sent to the register page which belongs to the register service; in other case, he becomes registered. At the end, the web application will show to the user what service he wants to select: register or login. Depending on what he chooses, the specific service will be run. Figure 7 shows our web metamodel. *WebOperation* class is related to the messages defined in the previous system sequence model.

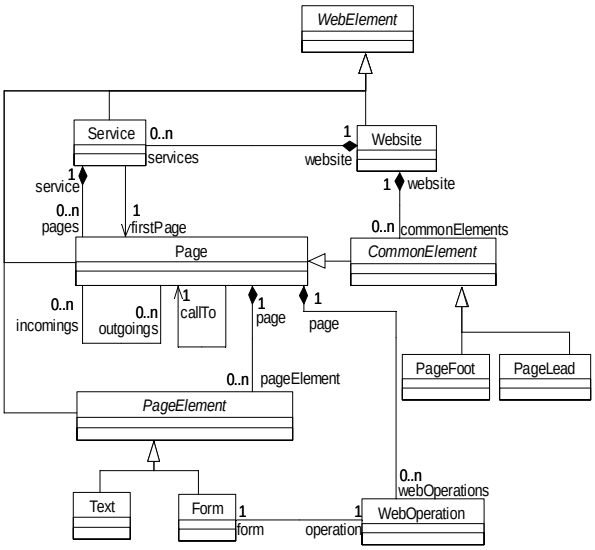

Fig. 7. Web application meta-model.

Figure 8 shows how the structure of a PIM which consists in actors associated to CUSs, CUSs associated to a set of ESs, ESs associated to a SS model which has messages, it is related to a structure in the web applications-PSM where webs have services, services have pages, and pages have web operations. At this level a relationship exists between both structures and that is the reason to create a transformation between each level.

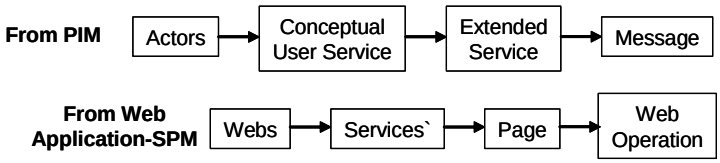

Fig. 8. Parallelism between the structure of the PIM and the web application-PSM.

2.3 Transformations

A transformation is defined as an operation that takes as input a model and produces another one [8]. This process is described by a transformation definition, which consists of a number of transformation rules. We have three transformations:

1. Class-PIM to content-PSM
2. Class-PIM to business logic-PSM
3. PIM to web application-PSM

As transformations for this methodology are many, due to space limitations we just introduce them a little bit. In a future work, we will develop them with more detail.

2.3.1 From class-PIM to Content-PSM

The first transformation we have to perform is to transform the class platform-independent model into the content model. There are many ways to approach this problem [7] from which we have chosen 'one class one table'. This transformation is made up of the rules we can see in Table 1.

Table 1. Transformation Rules

Class PIM	Content PSM
Class	A table with a column which is a identifier and a principal key
Extended class	A table with a column which is identifier. This identifier is a foreign key pointing to the identifier of the resulting transformed table of the parent class. See Figure 10

Multiple class association	A table with two columns being both identifiers and foreign keys and…
1:1	... each one is also unique and both are the primary key
1:n	.. the column which makes references to '1' in the multiplicity, is not a primary key, while the other one is the primary key
n:m	... both columns are the primary key
An attribute	A column which is inserted into the resulting table of the class which has the attribute

2.3.2 From Class-PIM to Business Logic-PSM

This is a complex transformation. We copy the same model from PIM to PSM but we change each class for an interface and we add a new class which implements such interface. [1] describe how this is done.

Table 2. Transformation Rules

Class PIM	Business logic PSM
Package structure	The same package structure is copied.
Datatype structure	The same data type structure is copied.
Class c	An interface i and a class c'. Both are added to the same transformed package that the initial class c was. This new class c' implements the new interface i.
Attribute a (belonging to a class c)	An attribute a' whose visibility is private. As c is transformed into i and c', the attribute a' is added into i and c'. Besides, two new methods are added to interface i: some get- and set- public methods which let the access a'.
An operation o (belonging to a class c)	An operation o' added to the transformed interface i from the class c. The data types used in o' have to be the copied ones.
Association between classes	An association, with the same features, between the created interfaces from the initial classes.
Association end	An attribute. An association end with multiplicity equals to '1' is transformed into an attribute whose type is the class which points to. If the multiplicity is 'n', the association end is transformed into other attribute but in this case the type is *Vector<class_which_points_to>* (using Java 5 notation)

2.3.3 From PIM to Web Application-PSM

Even though it is quite easy to see this transformation, it has many rules. A summary are the following rules which are explained in Table 3.

Table 3. Transformation Rules

Class PIM	Graphical User Interface PSM
An actor	A web
Conceptual user service associated to actor	A service associated to the transformed web for such actor
An extended service	A page. The same structure defined for a extended service model is built using pages from the web application-PSM
A message	A web operation

2.4 Code generation

The last stage of our methodology is code generation. Once the PSMs are automatically obtained from the created PIM, they are used to generate source code. There are many ways to generate source code from models. Most of them are based on templates considered as other kind of model transformation: model-to-text transformation [4]. The code generation helps the developer to not start from scratch his development. We have defined these transformations:

1. From the content PSM, SQL code.
2. From the business logic PSM, Java code.
3. A general web application is created for the web application PSM. This application just needs the model to run.

In summary, code generation covers business logic which is the data structures, and data base. Finally instead of creating code for the web application, its model is directly executed.

2.4.1 From Business Logic-PSM to Java

From a given class-PSM for each class or interface we need to create a new file where we have to:

- Write the name of the package and define the class or interface.
- Write every attribute: visibility, type & name.
- Write every method with all its parameters. It is possible that some methods have some associated source code as get and set methods.

[4] provide the following example to illustrate how a template-based model-to-text transformation is.

```
<<DEFINE Root FOR Class>>
```

```
public class <<name>>{
  <<FOREACH attrs AS a>>
  private <<a.type.name>> <<a.name>>;
  <<ENDFOREACH>>
  <<EXPAND AccessorMethods FOREACH
    attribute>>  }
<<ENDDEFINE>>
<<DEFINE AccessorMethods FOR
  Attribute>>
public <<type.name>>
  get<<name.toFirstUpper>>() {
    return this.<<name>>;   }
public void
  set<<name.toFirstUpper>>(
  <<type.name>> <<name>> )
{  this.<<name>> = <<name>>; }
<<ENDDEFINE>>
```

2.4.2 From Content PIM to SQL

From a given relational model and for each table create a new file where we have to:

- Write code which defines a table.
- Look up all its columns and write its name along with its type.
- Usually every table has a primary key which has to be written in the code.
- Finally, if the class has some foreign key.

2.4.3 Web Application-Model to Servlets

Creating the web application has been done by means of other different kind of transformation. Instead of creating directly code for the web application using JSP or

similar, we directly execute the created model. The final web application is a web previously constructed which only needs a parameter which is a model, in our case, the web model automatically generated, to work properly. This is a complex transformation. We copy the same model from PIM to PSM but we change each class for an interface and we add a new class which implements such interface. [1] describe how this is done.

2.5 Validation

A plugin for Eclipse has been developed to validate this process. Such a tool comes with a GUI which allows to directly draw the diagrams we have seen before for the PIM. Once this model has been created the next step is to transform it. At this moment the tool does not allow to modify PSM models, but the transformation can be performed. When it is done, a new Eclipse project is created and you can see both the PSM models created and the generated code. Models created with this tool use XMI format, therefore can be exported to other tools. The proposed example was created using this approach. Although this application is not very complicated, we have seen that from CUSs we are defining at the same time requirements and a certain degree of the navigation of the web application which is being created. Such a navegation is completely described by means of ESs. Finally, when defining SSs, we are getting some functions which are very close to code but they are enough abstract to be a part of the PIM. For the future, the tool should add the possibility of modifying the PSM models and to define a customed presentation of the web pages. We are now working on these aspects.

3 Conclusions

In this paper we have shown a methodological approach to web applications development. We have kept to MDA framework, where the development is performed by two different abstraction levels: independent and specific from the platform. The chosen models allow the user who uses this methodology to not lose in any moment the sense of what he is doing: he will generate code. This method help us construct better web applications in a time not very high, because it joins on the one hand advantages of having a set of steps very marked from a model-based methodology and on the other hand the agile development allows to construct prototypes of the final system from very early development stages. The automatic code generation makes MDA promote agile development, since the models from the code which are generated are kept, and they are the documentation of the final web application.

Using this process, we extend the work made by MIDAS adding new models to take into account, as well as a new approach closer to agile development such as our system sequence models. It should be feasible to make a fusion between MIDAS and our approach to build one more complete one.

Using this method, the web engineering industry has a new way to build simple web application in a faster manner. Our contribution is to provide metamodels which

are applied. It is not easy to find metamodels which are applied to a specific field. Usually, other proposed model-based methodologies only offer diagrams and one cannot see how the process is really working. Besides, they used to be models for UML class or Java diagrams. We have proposed a set of metamodels for web engineering as well as concrete syntax for those ones, explaining what each one does and how the models are transformed.

There is much possible future work to be done. It would be useful to include new models from UML but considering our goal of simplifies them to keep to agile development. Also, we could leave the agile development to centre our effort in constructing a comprehensive methodology for web engineering using complex models and complex transformations. Independently, our transformations have to be improved

References

1. Bézivin, J., Hammoudi, S., Lopes, D., Jouault F. 2004. Applying MDA Approach forWeb Service Platform. Enterprise Distributed Object Computing Conference.
2. Cáceres, P., Marcos, E. 2001. Procesos ágiles para el desarrollo de aplicaciones Web. Taller de Web Engineering de las Jornadas de Ingeniería del Software y Bases de Datos de 2001 (JISBD2001).
3. Ceri, S., Fraternali, P., Bongio, A. 2000. Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks 3 (1-6): 137-157.
4. Czarnecki, K., Helsen, S. 2006. Feature-based survey of model transfomation approaches. IBM Systems Journal, Vol 45, No 3.
5. Díaz, P., Aedo, I. Montero, S. 2001. Ariadne, a development method for hypermedia. Dexa 2001, Munich. LNCS 2113, 764-774.
6. Gómez, J., Cachero, C. 2002. OO-H Method: Extending UML to Model Web Interfaces. Idea Group Publishing.
7. Keller, W. 2004. Mapping Objects to Tables. http://www.objectarchitects.de/ObjectArchitects/papers/Published/ZippedPapers/mappings04.pdf, 2004
8. Kleppe, A., Warmer, J., Bast, W. 2003. MDA Explained - The Model-Driven Architecture: Practice and Promise. Addison-Wesley.
9. Koch, N., Kraus, A. The Expressive Power of UML-based Web Engineering. 2002. Second International Workshop on Web-oriented Software Technology (IWWOST02).
10. Mellor, S., Scott, K., Uhl, A., Weise, D. 2004. MDA Distilled, Principles of Model Driven Architecture. Addison-Wesley.
11. Millar, J., Mukerji, J. 2003. MDA Guide Version 1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01.
12. Vela, B., Cáceres, P., de Castro, V., Marcos, E. 2005. MIDAS: una aproximación dirigida por modelos para el desarrollo ágil de sistemas de información web, Chapter 4 from the book "Ingeniería de la web y patrones de diseño", Coordinadores: Mª Paloma Díaz, Susana Montero e Ignacio Aedo. Pearson - Prentice Hall.