

Using Weaving Models to automate Model-Driven Web Engineering proposals

Juan M. Vara¹, M^a Valeria De Castro¹, Marcos Didonet Del Fabro²,
Esperanza Marcos¹

1: Kybele Research Group, Universidad Rey Juan Carlos (Madrid - Spain)

2: ATLAS Group (INRIA & LINA), University of Nantes (France)

{juanmanuel.vara, valeria.decastro, esperanza.marcos}@urjc.es,
marcos.didonet@univ-nantes.fr

Abstract. The impact of Model-Driven Software Development in Web Engineering has given rise to the advent of Model-Driven Web Engineering, a new approach for Web Information Systems development. Its basic assumption is the consideration of models as first class entities that drive the development process from analysis to final deployment. Basically, each step of the process consists of the generation of one or more output models from one or more input models. Thus, model transformations are the key to complete each step of the process. However, the special nature of the behavioral models implied at the early stages of a Model-Driven Web Engineering process complicates the specification of a model transformation that works for any input model. In such situations, it is not feasible to automate the whole development process (one of the premises of Model-Driven Development). Some design decisions has to be considered before executing each model transformation. This work shows how we solve this problem in SOD-M, a model-driven approach for the development of Service-Oriented Web applications. The technique proposed is based on the use of weaving models as annotation models and it can be easily generalized to other domains and contexts.

1 Introduction

According to the roadmap for research in Service-Oriented computing (SOC) outlined by Papazoglou et al. [18], one of the main challenges that SOC has to face is the provision of methodologies supporting the specification and design of service composition. They should provide software engineers with the tools to move from the earlier stages of business analysis to the final step of implementation. While the design and development of simpler services is a relatively simple task, the development of business process comprising several independent services is not so simple. The transformation from high-level business models, generally defined by business analysts, to an executable business process language, such as BPEL4WS, is far away from being a trivial issue [24]. MDA (Model Driven Architecture) [16] is an important tool for the alignment of high-level business processes and information technologies [11]. It provides with a conceptual structure to combine the models built by business managers and analysts with those developed by software developers.

Attending to these facts, in previous works we have defined a model-driven approach for the development of service-oriented web applications [5]. The Service-Oriented Development Method (SOD-M) is part of MIDAS, a Model Driven Architecture framework for development of Web Information Systems (WIS). Therefore, SOD-M provides with all the advantages derived from the Model Driven Engineering (MDE) approach [2, 20]. The method defines a model-driven process and proposes a set of models at the different MDA abstraction levels. It starts from a high level business model. After several model transformations a service composition model is obtained. The later simplifies the mapping to a specific Web service technology. This work focuses on some of those models. The ones defined for behavioral modeling of service-oriented applications; and the

set of mappings rules between them. Without automating the mappings between models, the effort needed to manually transform the models become prohibitive and organizations using MDA will not get a full return on MDA's promise of faster, less costly software development [9].

All the mappings between the models comprised in MIDAS framework follow a common approach [4, 21]: firstly, we specify the transformation rules with natural language to later formalize them using graph transformation rules [19]. Once formalized, those rules are implemented using the ATLAS Transformation Language (ATL) [12]. Thus, after having formalized the mappings between the PIM models of SOD-M in [5], we face the implementation of those rules. This task raises some problems due to nature of the models considered in this work.

On the first hand, PIM to PSM transformations are more prone to be automated than PIM to PIM transformations. While the former implies decreasing the abstraction level and consequently handling more specific artifacts easier to be modeled; the later requires decisions from the designer due to the higher abstraction level of the implied elements. As a matter of fact, the advent of generic model transformation approaches was mainly related with addressing this problem [23].

On the other hand, business process models, like the ones we handle here, present considerable differences compared to structural models that raises a number of issues concerning model transformation [15, 17]. One has to be familiar with the hidden concepts in the metamodels. Resulting ambiguities on the metamodel layer have to be solved either by reasoning algorithms or user input.

Consequently, defining a one-size-fits-all model transformation in such contexts is too ambitious. There is a need to define non-uniform mappings [10]. More information is needed in order to execute the model transformation. It is our belief that in a real MDE context, this additional input should take the shape of a model.

By means of a case study, this work shows how, a weaving model, can be used as a container for the extra data. We are able to parameterize a model transformation by defining a weaving model that serves to annotate the source model. Then, both the source and the weaving model are taken as input to generate the target model. The results lend strong support to the idea that current model-driven engineering tools, like model transformations and weaving models are powerful enough to fulfill the requirements of Web Engineering. Moreover, the approach presented here can be easily generalized to other domains and contexts, where simple model transformations are not enough and there is a need for some kind of design decisions making any time a transformation is executed.

This paper is structured as follows: both the proposal and SOD-M, the framework in which it is formulated, are introduced in Section 2. A case study showing how to apply the idea is explained in Section 3. Finally, Section 4 outlines the main findings of the paper and raises a number of questions for future research.

2 Using model annotations in SOD-M

SOD-M, the Service-Oriented Development Method, is a service-oriented approach for Web Information Systems (WIS) development. It defines a model-driven process to obtain a service composition design from high-level business models.

In this work we focus on the models proposed by SOD-M for the behaviour modelling of the system at a high abstraction level (Analysis models in traditional Software Engineering, Platform Independent Models in MDE jargon). Since it is a service-oriented approach, such models serve to identify the *business services* offered by the system as well as the functionalities and processes needed to carry them out. The modelling process of SOD-M at PIM level is summarized in Fig. 1.

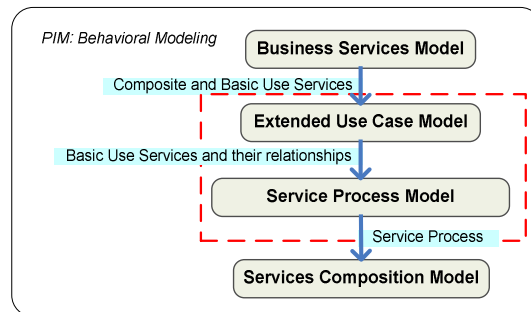


Fig. 1. Behavioural Modelling process in SOD-M

Here we address the mapping between two of these models (see [5] for more information on the others):

- The *Extended Use Case* model is represented with a Use Case model at a lower granularity than the previous one. Its main objective is to model the functionalities (services) required by the system to carry out each one of the business services identified in the Business Services model.
- The *Service Process* model is a kind of Activity Diagram used to represent the *service processes*. Thus, it shows the set of logically related activities that need to be performed to carry out a business service.

This work is based on an earlier study about the mappings between SOD-M PIM models. The aim of this work was initially to implement a set of mappings previously sketched [5] using the ATL language. However, once we started to code the ATL program, we realized that some information needed to generate the target model was not included in the source model. For each execution of the transformation some *extra* data was needed. In some sense, these extra data can be shown as a way of parameterize the transformation. In this context, the first option was to extend the source metamodel to support the modeling of these extra data. However this meant polluting the metamodel with concepts not relevant for the domain that it represents. We needed a different way to collect these extra data that was related with the source model but not included in it. That is, we just needed a way to annotate the input model [8]. Moreover, since this information, *parameters* or annotations had to be available from the ATL program that executes the transformation and considering that we were in a MDE context, the best option was to use another model (and thus to define a new metamodel): an annotation model. The idea behind the use of model annotations for model transformation is the following: suppose we have a source and a target metamodel, a terminal model conforming to the former and the corresponding model transformation. Then, for each annotation model used to execute the transformation, different target models are generated.

Finally, instead of defining a completely new metamodel for our annotation models, we use a weaving model to annotate the input model. A *Weaving Model* is a special kind of model used to establish and handle the links between models elements [1]. This model stores the links (i.e., the relationships) between the elements of the (from now on) *woven* models.

To create and handle the weaving models used in this work we have used the ATLAS Model Weaver (AMW). The model weaver workbench provides a set of standard facilities for the management of weaving models and metamodels [8]. Moreover, it supports an extension mechanism based on a Core Weaving Metamodel [7]. The Core Weaving metamodel contains a set of abstract classes to represent information about links between model elements. Typically, these classes are extended to define new weaving metamodels for specific contexts

So, we extended the aforementioned core weaving metamodel to obtain a new weaving metamodel for annotating Extended Use Case models. The weaving models conforming to the new metamodel serve as the annotation models for the execution of the model transformation. The process is summarized in the picture below.

For each execution of the ATL program, i.e., for each source model, we define a weaving model that conforms to an annotation extension of the core weaving metamodel. The weaving model contains a set of annotations that represent the information needed to execute the transformation, that is, the *parameters* used by some of the rules of the ATL program that encodes the mapping. Both, the source (the woven model) and the weaving model are taking as inputs to generate the output model. This way, given an Extended Use Case model, we can generate different Service Process models depending on the weaving/annotation model attached.

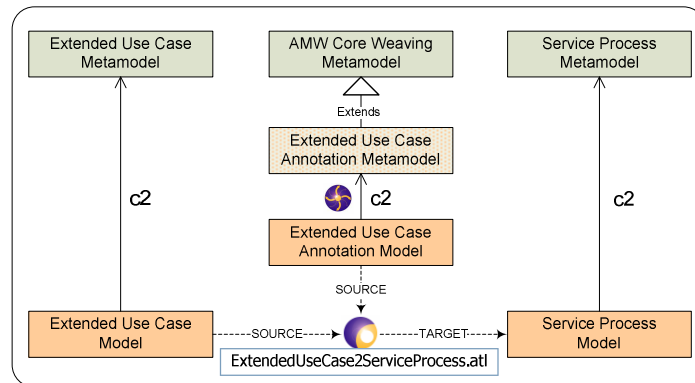


Fig. 2. Using weaving models to annotate Extended Use Case models

The following section sets out how this approach has been applied in a concrete scenario. The complete case study can be downloadable as an Eclipse project [22].

3 Case Study

To illustrate this work we use the models from a conference management system we have developed

3.1 The metamodels

As mentioned before here we address the mapping between the Extended Use Case metamodel (a simplified version of the UML 2.0 Use Case metamodel) and the Service Process metamodel (a simplified version of the UML activity package). To carry out this task we define a new extension of the core weaving metamodel for annotating Extended Use Case models: the Extended Use Case Annotation metamodel. Here we will introduce just the later since the formers can be found at [22].

Extended Use Case Annotation metamodel. If we want to use an Extended Use Case model as input in a model transformation, we need some extra information. For instance, if a use case includes two or more use cases, the order in which they should be executed in the including use case should be specified, since the mapping rule for `<<include>>` relationships [5] gives the two options showed in the picture below.

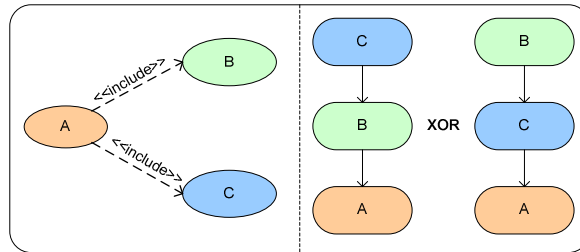


Fig. 3. Include Relationships mapping rule

As already mentioned, these data are not relevant to the model itself, so we use another model to collect them. More specifically, since these data represents relations between the elements of the Extended Use Case model, we use a weaving model. This process is known as annotation and the weaving model is known as the annotation model. Then, each link in the weaving model represents an annotation for the woven model. All this given, Fig. 4 shows the new weaving metamodel for annotating Extended Use Case models.

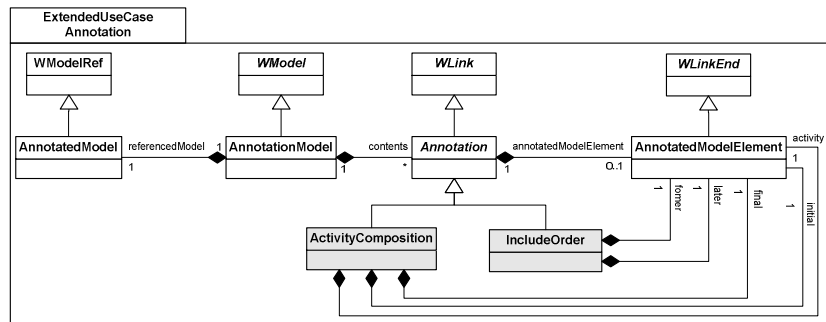


Fig. 4. Weaving metamodel for annotating Extended Use Case models

Each annotated use case in the Extended Use case model will be represented by an *AnnotatedModelElement* in the weaving model. We distinguish two types of annotations.

The *IncludeOrder* element helps in the mapping of several *include* relationships attached to the same use case. It relates the included use cases in groups of two, defining the order in which they should be executed (*former* and *later AnnotatedModelElement*). This information is used in the transformation rule that maps use cases to service activities to know which service activity has to precede the other.

On the other hand, the *ActivityComposition* elements serve to identify which use cases correspond to complex services and thus have to be mapped to activities in the Service Process models (*activity AnnotatedModelElement*) and which use cases have to be mapped to the initial and the last service activity for each activity (*initial* and *final AnnotatedModelElement*).

3.2 The models

The source model. Here we will refer just to the Extended Use Case model. As a matter of fact there will be two source models, this one (the woven model) and the annotation model, that can be thought of as an auxiliary source model. The model is showed in the left-hand side of the figure below.

The Web system for conference management offers three different services: *Submit an Article*, *View Submitted Articles* and *Edit Author Data*. In order to provide with this complex services,

some basic services are needed, as the *Log-In* or the *Register* ones. To model the relation between the different Use Cases two types of associations are used: `<<include>>` and `<<extend>>`. The former implies that the behavior of the included Use Case is contained in the including Use Case, while the later specifies how and when the behavior defined in the extending use case can be inserted into the behavior defined in the extended use case.

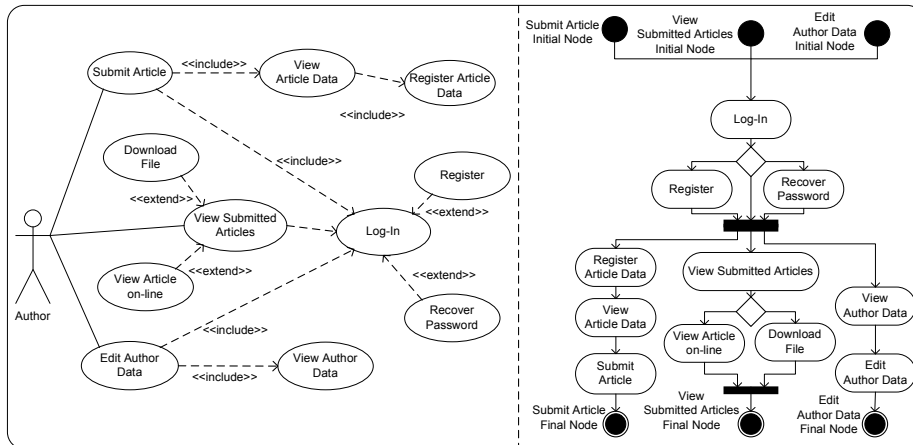


Fig. 5. Extended Use Case model and Service Process models for the conference management Web system

The target model. The Service Process model for our case study is shown in Fig. 5 (right-hand side). Each complex service identified in the previous model is mapped to an activity, while the basic services that it uses are represented as service activities. This way, we have three different activities that use a set of service activities. For instance the Log-In service activity is used by the three activities, Submit Article, View Submitted Articles and Edit Author Data.

Notice that the previous model (Extended Use Case model) did not show which the complex services were, those that had to be mapped to activities, or the order in which included use cases had to be executed. This kind of information is not conceptually relevant to be part of the corresponding metamodel. So it will be collected in the annotation model.

The annotation model. We have to annotate the *main* source model (i.e. the Extended Use Case model) to provide with the additional information needed to execute the transformation. To that end, we use the AMW tool to create a new weaving model conforming to the Extended Use Case annotation metamodel.

As mentioned before, a weaving model conforming to this metamodel includes two different types of *WLinks* (in fact, of annotations): *ActivityComposition* and *IncludeOrder*. The former serves to identify which of the use cases from the Extended Use Case model correspond to a Service provided by the system, as well as the entry and exit points to carry out the service. The later helps to solve the problem about the mapping of several include relationships attached to the same use case. The annotation model for the case study is showed in Fig. 6 (the whole process for obtaining the model as well as the metamodel is explained in detail in [22]).

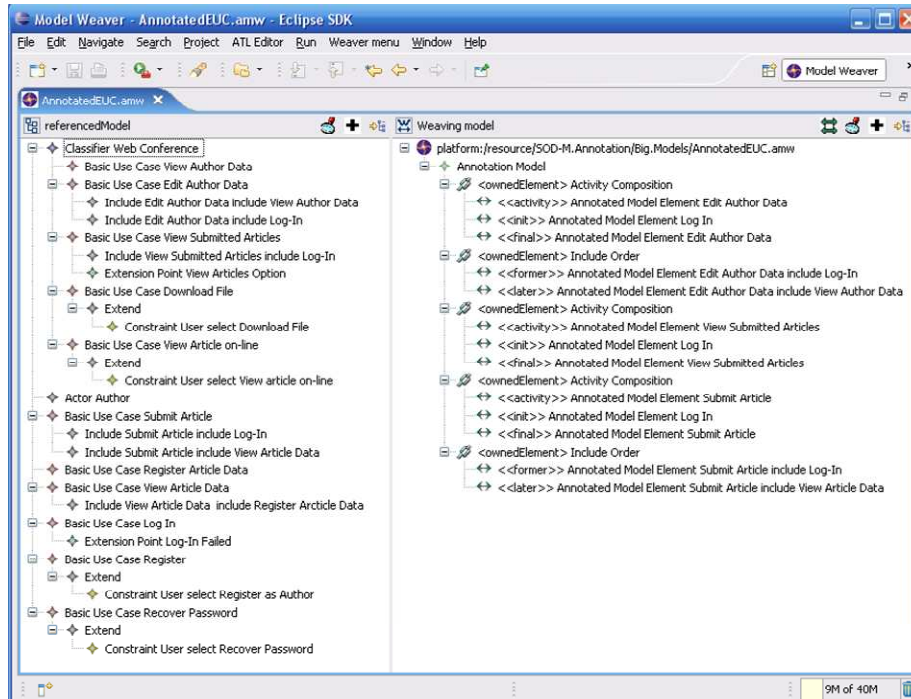


Fig. 6. Extended Use Case model and Extended Use Case Annotation model for the conference management Web system

We add an *Activity Composition* annotation for each service provided by the system. It serves to identify the complex services provided by the system, as well as its entry and exit points. For instance, the first one identifies the Edit Author Data as a complex service. It means that the corresponding Service Process model has to include an Edit Author Data activity. Moreover, the `<<init>>` and `<<final>>` annotations indicates that the Edit Author Data service starts by Log-In in the system and finishes by Editing the Author Data.

In the same way, we add an *Include Order* annotation for each pair of include relationships attached to a same use case. The annotation defines the order in which the included use cases should be executed to carry out the including use case. For example, according to the Extended Use Case model, both the Log-In and the View Article Data use cases are included in the Edit Author Data use case. However, there is no way to identify which one has to be completed first. To that end, the annotation (or weaving) model contains an *Include Order* annotation setting that the Log-In basic use case must precede the View Article Data use case. Therefore, as Fig. 6 shows, the Edit Author Data activity includes the Log-In and View Author Data service activities in the right order.

3.3 Using the annotations to customize the transformation

Once we have defined the *main* input model (the Extended Use Case model) and the *auxiliary* input model (the annotation model), it is time to code the model transformation program. Here we will focus just in showing an example of how to use the information provided by the annotation model. In this point some skills in the coding of model transformations are supposed to the reader.

The annotation of the input model allows us to add the missing data we need to execute the transformation. In order to use this information, we just have to include some helpers (auxiliary

functions) in the ATL program. For instance, to map an *include* object we have to know if it is related with other *include* objects (remember the ambiguity about the mapping of include relationships captured in Fig. 3). Thus, we define the following helper. It navigates the weaving model (the annotation model) looking for *IncludeOrder* annotations, whose *former* element is the same that was being mapped when the helper was invoked. To identify the *include* object, the *_xmiID_* property is used. It serves as the identifier of a woven element.

```
-- Returns the link in the Weaving model referring to the 'inc' ServiceProcess!Include
helper context ExtendedUseCase!Include def: getIncludeOrderLink(): AMW!IncludeOrder =
    AMW!IncludeOrder.allInstances()->asSequence()->
        select(link | link.former.element.ref = self._xmiID_)->first();
```

Then, we define two different rules for mapping *include* objects: one for those not related with other include objects and one for those related. We include a guard in those rules to distinguish which rule should be used in each specific case. The guard invokes the helper we have just showed to make the decision.

```
-- Normal mapping of Include relationships
rule IncludeSimple2ServiceProcess {
    from
        inc : ExtendedUseCase!Include(inc.getIncludeOrderLink().oclIsUndefined())
    to
        edge : ServiceProcess!ControlFlow (
            name <- ([T + inc.addition.getFinalActionName() + 'to '+ inc.includingCase.name + T]),
            source <- inc.addition.getFinalAction(),
            target <- inc.includingCase
        )
}

-- Mapping of Include relationships involved in a multi-option selection
rule Include2ServiceProcess {
    from
        inc : ExtendedUseCase!Include(not inc.getIncludeOrderLink().oclIsUndefined())
    to
        edge : ServiceProcess!ControlFlow (
            name <- ([T + inc.getSourceNodeName() + 'to ' +
                inc.getIncludeOrderLink().getNextInclude().addition.getPrevUseCase().name + T]),
            source <- inc.addition.getFinalAction(),
            target <- inc.getIncludeOrderLink().getNextInclude().addition.getPrevUseCase()
        )
}
```

As the rules show, in both cases the *include* object is mapped to an *Edge* object. The difference comes in which is the target of the edge. If the include object is not related to other include objects, the target of the corresponding edge will be the service activity that maps the including use case (identified by *inc.includingCase* property). Otherwise, the edge target will be the Service Activity that maps the use case linked to the include object that was set to be subsequent to the one that is being mapped. To identify that service activity we code a new helper that given an *IncludeOrder* link, returns the reference to the *later* include.

```
-- Given the 'self' IncludeOrderLink, it returns the 'include' referred as next by the link
helper context AMW!IncludeOrder def: getNextInclude(): ExtendedUseCase!Include =
    ExtendedUseCase!Include.allInstances()->asSequence()->select(inc | inc._xmiID_ = self.later.element.ref)->first();
```

Concluding this section, we can say that we are able to customize the execution of the model transformation and consequently obtain different outputs, by modifying the annotation or weaving model used to run the transformation.

4 Conclusion

This work has presented a technique to address the development of model transformations in the context of Model-Driven Web Engineering (MDWE). The special nature of the models handled at the initial stages of a MDWE process, used to model the business logic, complicates the coding of valid transformations. On the one hand, the gap between models uses to be bigger than usual. On

the other hand, model transformations are generic by nature (they have to work for any terminal model conforming to a given metamodel). To fill the afore-mentioned gap we propose to introduce some design decisions to drive the transformation. This way, we keep generic nature of the model transformation, since we are just *parameterizing* it. Thus, we produce a different target model depending on the design decisions taken.

In this study we have showed how weaving models can be used as annotation models to introduce design decisions by the time of executing a model transformation. We have applied this technique to develop the mapping from the Extended Use Case model to the Service Process model of SOD-M, a service-oriented development method for Web Information Systems. To that end, we have defined a new weaving metamodel for annotating Extended Use Case models. Next, we have coded a model transformation that takes as input, not only the desired Extended Use Case model, but also a weaving model that annotates the former. Then, depending on which the annotation model is we obtain a different Service Process model. As a result, we are able to customize the mapping process and keep the generic nature of the model transformation without polluting the source model.

This technique contributes to improve the accuracy of the models used at the initial stages early stages of WIS development and can help to increase the quality of the models built as well as the subsequent code generated from them. These activities are especially important in proposals aligned with MDA (like SOD-M) because it proposes the models to be used as a mechanism to carry out the whole software development process.

The approach presented here can be easily generalized to other domains and contexts. The proof of concept has been achieved and the results are available as open source. We may qualify this technique of simple or complex according to the point of view. In comparison with the benefits it brings, the solution may be considered as quite simple as has been discussed in the paper. This is mainly due to the simplicity, the genericity and power of the AMW tool and its good coupling with the ATL model transformation solution.

At the present time we are working in the integration of this model transformation process in M2DAT (MIDAS MDA Tool), a case tool which integrates all the techniques proposed in MIDAS for the semiautomatic generation of Service-Oriented applications. The tool is now under development in our research group and its early functionalities have been already presented in previous works [22]. Besides, the open issue of automating the rest of mappings in the MIDAS methodology using the approach presented here is being tackled.

Acknowledgments. This research is partially granted by the GOLD project financed by the Ministry of Science and Technology of Spain (Ref. TIN2005-00010) and the M-DOS project (URJC-CM-2007-CET-1607) cofinanced by the University Rey Juan Carlos and the Regional Government of Madrid.

5 References

1. Bernstein, P A. Applying Model Management to Classical Meta Data Problems. First Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA. 2003.
2. Bézivin, J.: In search of a Basic Principle for Model Driven Engineering. *Novatica/Upgrade*. V, 21-24 (2004).
3. Bézivin, J.: Some Lessons Learnt in the Building of a Model Engineering Platform. 4th Workshop in Software Model Engineering (WISME), Montego Bay, Jamaica (2005).
4. Caceres, P., De Castro, V., Vara, J.M., Marcos, E.: Model transformations for hypertext modeling on web information systems. *ACM Symposium on Applied computing (SAC)*. ACM Press, Dijon, France (2006) 1232-1239.

5. De Castro, V., Vara, J.M., Marcos, E. Model Transformation for Service-Oriented Web Applications Development. 7th International Conference on Web Engineering. July 2007.
6. De Castro, V., Marcos, E., López-Sanz M.: A Model Driven Method for Service Composition Modeling: A Case Study. *Int. Journal of Web Engineering and Technology*. 2006 - Vol. 2, No.4, pp. 335 - 353.
7. Didonet Del Fabro, M.: Metadata management using model weaving and model transformation. Ph.D. thesis. University of Nantes (2007).
8. Didonet Del Fabro, M., Bézivin, J. and Valduriez P. (2006). Weaving Models with the Eclipse AMW plugin. Eclipse Modeling Symposium, Eclipse Summit Europe, Esslingen, Germany. 2006.
9. Flore, F.: MDA: The Proof is in Automating Transformations between Models. In *OptimalJ White Paper*, pages 1-4, 2003.
10. Gerber, A., Lawley, M., Raymond, K., Steel, J., Wood, A.: Transformation: The Missing Link of MDA. *International conference on Graph Transformation (2002)* 90-105.
11. Harmon, P.: The OMG's Model Driven Architecture and BPM. *Newsletter of Business Process Trends (May 2004)*. <http://www.bptrends.com/publications.cfm>
12. Jouault, F., Kurtev, I.: Transforming Models with ATL. *Satellite Events at the MoDELS 2005 Conference (2006)* 128-138.
13. Mellor, S., Scott, K., Uhl, A., Weise, D.: Model-Driven Architecture. In: *Advances in Object-Oriented Information Systems*. pp. 233-239 (2002).
14. Moreno, N., Romero, J., Vallecillo, A.: An Overview of Model-Driven Web Engineering and the MDA. In: *Web Engineering: Modelling and Implementing Web Applications*. pp. 353-382 (2008).
15. Murzek, M., Kramler, G.: Business Process Model Transformation Issues - The Top 7 Adversaries Encountered at Defining Model Transformations. In: *International Conference on Enterprise Information Systems (2007)* 144-151.
16. OMG. MDA Guide V1.0.1. Miller, J., Mukerji, J. (eds.) Document N° omg/2003-06-01 (2001). Accessible in: <http://www.omg.org/cgi-bin/doc?omg/03-06-01>.
17. Strommer, M., Murzek, M., Wimmer, M.: Applying Model Transformation By-Example on Business Process Modeling Languages. *Advances in Conceptual Modeling -Foundations and Applications (2007)* 116-125.
18. Papazoglou, M. et al.: Service-Oriented Computing: A Research Roadmap. In: *Dagstuhl Seminar Proceedings (2006)*.
19. Rozenberg, G.: *Handbook of graph grammars and computing by graph transformation: Volume I. Foundations*. World Scientific Publishing Co., Inc., River Edge, NJ, USA (1997).
20. Selic, B.: The Pragmatics of Model-Driven Development. *IEEE Software*. 20, 19-25 (2003).
21. Vara, J.M., Vela, B., Cavero, J.M., Marcos, E.: Model transformation for object-relational database development. *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing (2007)* 1012-1019.
22. Vara, J.M., Didonet Del Fabro, M.: Web Applications Modelling. ATL Use Cases. <http://www.eclipse.org/m2m/atl/usecases/webapp.modeling/>.
23. Varró, D., Pataricza, A.: Generic and Meta-Transformations for Model Transformation Engineering. *UML 2004 - The Unified Modeling Language. Model Languages and Applications. 7th International Conference, Vol. 3273*. Springer (2004) 290-304.
24. Verner, L.: BPM: The Promise and the Challenge. *Queue of ACM Vol. 2, N° 4 (2004)* 82-91.