Inter-DSL Coordination Support by Combining Megamodeling and Model Weaving

Frédéric Jouault[1], Bert Vanhooff[2], Hugo Brunelière[1], Guillaume Doux[1], Yolande Berbers[2], Jean Bézivin[1]

[1] AtlanMod, INRIA RBA Center & EMN
4 rue Alfred Kastler
44307 Nantes, France
Firstname.lastname@inria.fr

[2] Department of Computer Science, K.U. Leuven
Dept. Computer Science Celestijnenlaan 200A
B-3001 Heverlee Belgium
Firstname.lastname@cs.kuleuven.be

ABSTRACT

Model-Driven Engineering (MDE) advocates the use of models at every step of the software development process. Within this context, a team of engineers collectively and collaboratively contribute to a large set of interrelated models. Even if the main focus can be on a single model (e.g. a class diagram model), related elements in other models (e.g. a requirement model) often have to be considered and/or accessed. Moreover, all the involved models do not necessarily conform to the same metamodel and thus may have been built using different independent Domain-Specific Languages (DSLs). Such a situation has already been frequently observed in many large-scale industrial deployments of MDE. Manually coordinating all the involved models, i.e. being able to both manage and use the links existing between them, can become a cumbersome and difficult task. As a proposal to solve this inter-DSL coordination issue, we introduce in this paper a generic and extensible inter-model traceability and navigation environment based on the complementary use of megamodeling and model weaving. We illustrate our solution with a concrete working example.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures – *Languages (description, interconnection, definition)*

D.2.12 [**Software Engineering**]: Interoperability – *Data mapping*

D.2.13 [**Software Engineering**]: Reusable Software – *Domain engineering, Reuse models*

D.3.2 [**Programming Languages**]: Language Classifications - *Design languages, Very high-level languages*

Keywords

Model-Driven Engineering, Domain-Specific Languages, Coordination, Megamodeling, Model Weaving.

SAC'10, March 22-26, 2010, Sierre, Switzerland.

1. INTRODUCTION

Model Driven Engineering (MDE) largely promotes the use of models at each step of the software development lifecycle. One possible approach, and the one mainly considered in this paper, is to model each aspect of the system using a different Domain Specific Language (DSL). A DSL is a modeling language that strongly focuses on a specific area of a system such as requirements, data structure or user interface navigation for instance.

All the different models representing a same system are often interrelated. However, we cannot assume that all the DSLs which have been used for building them systematically provide explicit inter-DSL coordination capabilities. On the contrary, a DSL is often deliberately defined in isolation, focusing on just one aspect of the system (while completely ignoring others) and being independent of other DSLs. This makes it more reusable but also means that useful interrelationships between models built with different DSLs cannot be expressed directly using them. Nevertheless, for a given set of DSLs within the context of a project or project's type, we can identify a cartography of the coordination by specifying the possible kinds of relationships that may occur. For instance, the link between a *Class* element in a *Java* model and a *Paragraph* element in a *Requirements* model, or an *Entity* element in a *Data* model, can be considered.

In an incremental development process, most of the models are constantly elaborated and updated. Modifying one model might have an impact on, and require corresponding modifications to, other related models. Therefore, a developer needs to be able to quickly and easily navigate from model to model and from model element to model element. As soon as an MDE process includes more than a few DSLs, manually maintaining links and navigating between models (and corresponding model elements) does not scale very well. The problem of *language coordination* – managing many interrelated DSLs – is thus emerging as a very important challenge in MDE software engineering.

In this paper, we propose a generic and extensible modeling solution which is based on the combined use of megamodeling [9] and model weaving [11]. Concretely, we offer a framework for representing both model-level and model element-level links and provide tool support for developers to be able to use these links and to navigate along them. This general modeling solution has been directly applied within the specific context of inter-DSL coordination.

Within the paper, we use the *Pet Store* reference application as an example. This application involves several DSLs and underlying models, each one describing a different aspect of this software system. While the techniques of inter-model coordination have a much broader scope than presented in this simple example, it allows us to practically demonstrate our solution. We consider a snapshot of a given project at a certain point in the development process. At this point, a developer wants to contribute to the project, for example by adding a new feature. In order to do so, considering a single model often does not suffice. Rather, a number of interrelated models need to be considered and modified accordingly. Our solution, together with all the used metamodels and *Pet Store* models, are publicly available from the Eclipse AtlanMod MegaModel Management (AM3) project [1].

This paper is organized as follows. Section 2 briefly presents the fundamental model engineering techniques, and more especially the AmmA MDE toolkit that is used in our approach. Section 3 explains in details the proposed generic and extensible modeling solution. Section 4 introduces the *Pet Store* example, and uses it in order to illustrate how concrete models can be coordinated so that related traceability links may be navigated. Similar works on inter-model coordination in general are listed in Section 5 whereas Section 6 discusses some related issues. Section 7 concludes the paper.

2. MODEL ENGINEERING WITH THE AmmA TOOLKIT

This section briefly describes the AmmA (AtlanMod Model Management Architecture) MDE toolkit that is used as a basis for this work. AmmA is composed of four functional blocks available from the Eclipse website:

- **Model Transformation** offers model-to-model transformation capabilities. Within our framework, it is provided by ATL (AtlanMod Transformation Language) [2];
- **Model Weaving** allows representing links between elements from different models. This functionality is provided by the AMW (AtlanMod Model Weaver) component [3];
- **Global Model Management or Megamodeling** assists in managing large collections of models and high-level links between them. The AM3 (AtlanMod MegaModel Management) component [1] provides an implementation for this;
- **Projection** of models to and from other technical spaces like XML, grammarware, etc. The block implementing the various tools is called ATP (AtlanMod Technical Projectors).

In order to allow developers to establish the cartography of the coordination between different related models, we need to be able to represent all the available links. In order to do so (section 3), we need the Model Weaving component to manage links at the model element-level (subsection 2.1) and the Global Model Management component to manage links at the model-level (subsection 2.2). This latest component will then be extended in order to effectively enable the required navigation. In the next subsections, both these components are briefly discussed.

2.1 Model Weaving with AMW

Model weaving operations are performed between models or metamodels (two or more). A *weaving model (*WM*)* specifies links between elements from different models (called *woven models*). A weaving model conforms to a *weaving metamodel (*WMM*)*, which provides the semantics of the links. AMW provides concrete solutions to the following model weaving related problems:

- **Automating link creation**. The generation of abstract correspondence links between elements from different models can be partially automated by means of heuristics (several ones are already available);
- **Storing links**. It is possible to record this set of model element-level links as a whole, in order to use it later in various contexts;
- **Using links in tools**. It is possible to use the links as the input of automatic or semi-automatic tools. The typical use is the generation of a concrete transformation from a weaving model with the help of a higher order transformation.

The output of a model weaving operation on a set of models is a WM that conforms to a WMM. Hence, this WM remains linked to the set of woven models. Each link (a model element of WM) is typed by a concept specified in the corresponding WMM.

The WMM defines the available link types. Since model-element level links can be used in many scenarios (e.g., traceability and model annotations) it is quite impossible to define a common WMM supporting all the situations. Therefore, AMW only defines a minimal WMM, which can be extended for specific purposes. The minimal metamodel only offers a single (but extensible) type of link. Depending on the purpose of the links, the user should specialize them to one or more project-specific link types by adding more data and constraints. Thus, a concrete WMM may be expressed as an extension of another WMM. This means that a hierarchy of metamodels is created, which allows AMW to generically deal with all weaving-related tasks.

2.2 Global Model Management with AM3

The AtlanMod MegaModel Management tool, AM3 [1], is a generic and extensible environment for dealing with large collections of heterogeneous models and metamodels. They are often accompanied by the tools and services which are usually combined in a given domain of application and/or specific process. The Global Model Management (GMM) principles [5] are implemented in this tool: for each part of the "real-world" intended to be modeled, which is often composed of several different *systems*, it is assumed that a *megamodel* can be specified and used in order to define the associated metadata.

A conceptual overview of the AM3 framework is shown in Figure 1. MDE introduces three different kinds of models [6]: terminal models (M1) conform to metamodels represent real-world systems, metamodels (M2) conform to a metametamodel and metametamodels (M3) conform to themselves. A megamodel is a specific kind of terminal model whose elements represent models themselves, as well as relationships between them and metadata on them. As it is a terminal model, a megamodel conforms to a specific metamodel. In the context of a given working zone, the

megamodel records all available resources or artifacts and their inter-relations.

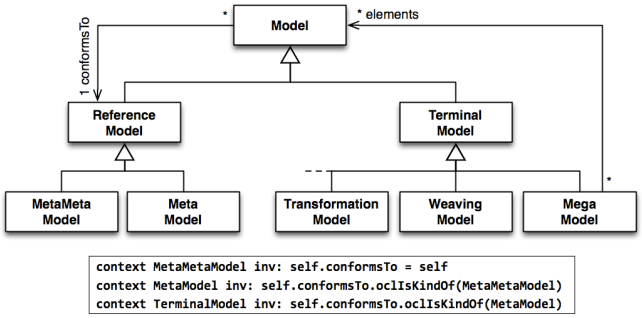

Figure 1. AM3 Conceptual Framework

A megamodel can thus be viewed as a metadata repository where representations of models and links between them are stored as a model. Other artifacts such as textual files and even tools or services can also be managed by a megamodel if the corresponding extension of the metamodel of megamodel is available.

The AM3 solution from the Eclipse foundation implements the previously described conceptual framework. It is an open source project which is part of the *GMT* subproject, which is itself part of the top-level Eclipse *Modeling* project. The generic and extensible AM3 megamodeling solution provides not only the capabilities to explicitly specify the metadata associated with a given system or process, but also a standard *Megamodel Navigator* as well as generic and extensible editors for instantiating and editing the megamodel in a more user-friendly way (Figure 6 and Figure 7). In addition, it offers several extension points allowing domain-specific extensions of the tool. Both the metamodel of megamodel and its related UI components can be extended.

AM3 is composed of two distinct sets of Eclipse plug-ins:

- The *core* plug-ins providing the basic metamodel of megamodel, the core APIs , the core runtime environment and associated generic navigator and editors;
- The *extension* plug-ins providing extensions of the metamodel of megamodel and corresponding extensions of the UI (for instance specific editor pages, contextual actions, etc).

With AM3, users can build their customized megamodeling solution by extending either the core plug-ins or other already existing extension plug-ins. A set of generic MDE extensions have already been developed: *GMM* for Global Model Management, *GMM4ATL* for model transformation with ATL, *GMM4CT* for Composite Transformations.

In the next section, we describe a new extension that is specifically focused on the problems that were described earlier. By integrating the use of model weaving in the megamodeling environment and providing a number of UI extensions, we aim to provide some easy-to-use inter-DSL coordination capabilities.

3. COMBINING MODEL WEAVING AND MEGAMODELING

The problem of inter-DSL coordination, considering inter-model traceability and navigability, can be divided into two complementary problems corresponding to two different abstraction levels:

- *traceability/navigability between models* (higher level);
- *traceability/navigability between model elements* (lower level).

These two levels are related by a refinement relationship: navigability between model elements is a refinement of navigability between models. To support these two levels, both model-level links and model element-level links must be considered. Global Model Management proposes to use a megamodel (section 2.2) to represent all models involved in a given context as well as various relationships between them. Therefore, GMM is a solution applicable for model-level links. Model weaving (section 2.1) allows representing different kinds of fine-grained relationships between elements from different models. Model element-level links can thus be managed by using Model Weaving.

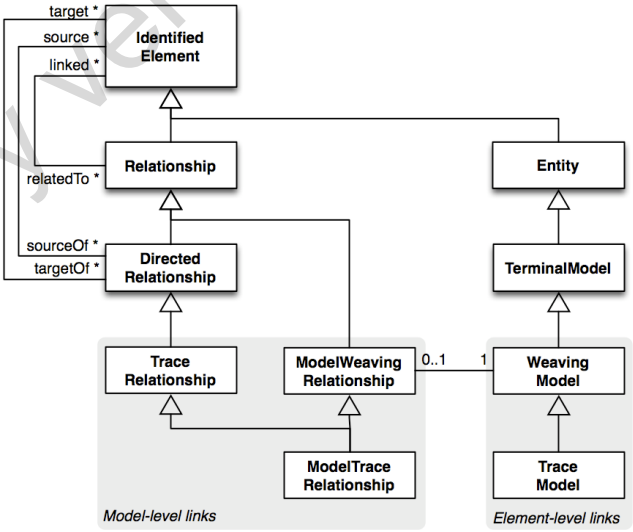

Figure 2. Support for navigation links in the GMM extension of the metamodel of megamodel

What was missing was a way to represent the *refinement* relation between the two levels of links. We have solved this problem by incorporating the notion of traceability links and models directly into the GMM extension of the metamodel of megamodel (Figure 2). The top part of the figure shows basic concepts, from the metamodel of megamodel, to represent models and relationships between them in a megamodel. On the bottom left, we have extended these to the notion of *ModelWeavingRelationship* and *ModelTraceRelationship*, the latter being a more specific case of the former. Since they both derive from the Relationship element, they have the ability to express high-level links between models. On the bottom right, we introduce the *WeavingModel* and *TraceModel* concepts as representations of weaving models expressed by AMW (low-level links). The refinement relationship between both levels can now be represented simply by adding an

association between *ModelWeavingRelationship* and *WeavingModel*.

This solution handles the navigability problem in a generic and extensible way. Navigability is provided by recording traceability links between both models and model elements. The concrete weaving or trace metamodel used can differ from project to project (extensibility) while navigation will be handled generically since each weaving metamodel will derive from a common base metamodel (AMW).

Figure 3 describes an example of a megamodel. The legend at the bottom may be informally interpreted as a summary of the metamodel of this megamodel, previously presented in Figure 2. Each of the models (M*x*) contains a number of model elements. These models conform to different metamodels (MM*x*) and are interrelated via model-level links (*MWRelation*). Each model-level link can be associated to a weaving model (*WeavingModelx-y*) which refines this link. All this information is stored within a single megamodel. Note that the weaving models are also registered into the megamodel, just like any other models.

Within a specific context, the semantics of the traceability links can be specialized: additional information about model-level links is specified by the metamodel of the megamodel and specific semantics for model element-level links can be provided by specializing the weaving metamodel(s).

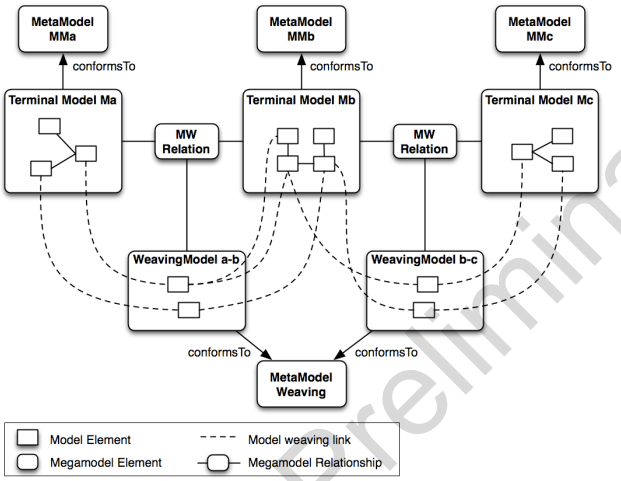

Figure 3. Overview of the proposed generic and extensible solution.

The proposed solution allows users and tools to retrieve and navigate all kinds of traceability links recorded into the megamodel. We have also extended the AM3 megamodeling environment with two specific views to allow users to zoom in on a given model-level link. This way, it is possible to see which elements of the corresponding models are actually concerned and how they are precisely interrelated.

In the next section, we apply the proposed solution to a concrete example and demonstrate how the extended AM3 megamodeling environment can thus be used by developers as an inter-DSL coordination solution.

4. USE CASE: THE "PET STORE" APPLICATION

In 2001, Sun described an application named *Pet Store* with the objective of providing to developers a typical example of program built using J2EE best practices. Later, Microsoft reproduced this experiment using their *DotNet* framework. This *Pet Store* sample application has since been used in many areas of software development as a benchmark to evaluate alternative development methodologies. The Java Pet Store 2.0 is the reference application for building Ajax web applications on Java Enterprise Edition 5 platform [7]. It illustrates good practices for using AJAX with Java, building AJAX-enabled JSF component libraries, using Java Persistence APIs, applying MVC (Model/View/Controller) and other design patterns. In this particular AJAX Web application, several services such as Google Maps service for location, PayPal service for payment, etc. are used to implement parts of the functionalities.

We consider a snapshot taken during the software development cycle of a simplified Pet Store application: all artifacts developed at this time are models built using various DSLs. How the various models have been built and how the traceability relations between them have been established is out of the scope of this paper. They can be created manually, derived automatically or generated by model transformations. We currently focus solely on the representation of these traceability links and on their navigability.

A *Pet Store* case, based on seven different types of models (i.e. *Requirements*, *Use Cases*, *Deployment*, *Entity-Relationship*, *Class Diagram, Page Navigation* and *Java Project*), has been developed. These terminal models are provided in EMF-XMI. The seven corresponding metamodels are expressed in KM3 [6] but are also available in EMF-XMI (conforming to Ecore).

All these available models and metamodels are registered into a megamodel. In this megamodel, we also record the high-level relationships between the models. As discussed in the previous section, we also represent low-level relationships using weaving models. Figure 4 shows a snapshot of the *Pet Store* project showing a number of such relevant links. Solid arrows are model-level links whereas dotted arrows are model element-level links.

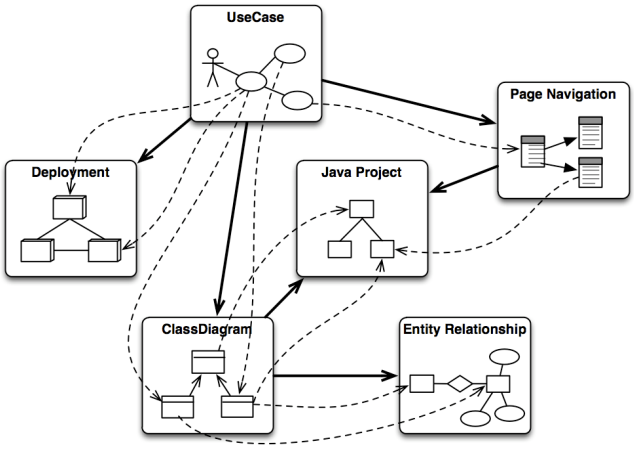

Figure 4. Traceability links between models and model elements

For example, there is a traceability link between the *UseCase* model and the *Deployment* model. By zooming on this link, more detailed traceability information can be retrieved: a given use case of the *UseCase* model is linked to two different components of the *Deployment* model. Using the same process, there is a traceability link between the *ClassDiagram* model and the *JavaProject* model: a design class of the *ClassDiagram* model is linked to two different Java implementation classes in the *JavaProject* model. A more detailed example of the links in Figure 4 can be found in Figure 5. In this specific example, there are traceability links respectively between the *UseCase* and the *ClassDiagram* model and between the *ClassDiagram* model and the *Entity-Relationship* model.

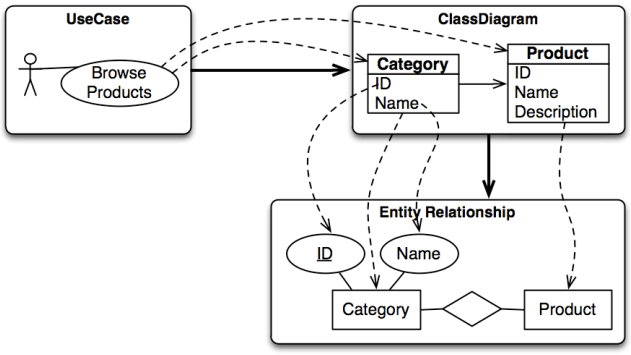

Figure 5. Detailed examples of traceability links

Zooming in on the model-level link between the *UseCase* and *ClassDiagram* models (Figure 5), we see that from the *Browse Product* use case we can navigate to design classes *Category* and *Product*. When we zoom in on the link between the *ClassDiagram* and *EntityRelationship* models, we see that the *Category* design class is related to the *Category* entity and that the *ID* class attribute is related to the *ID* field. Thus, it can be deduced that the *Category* entity is (indirectly) linked to the *Browse Products* use case.

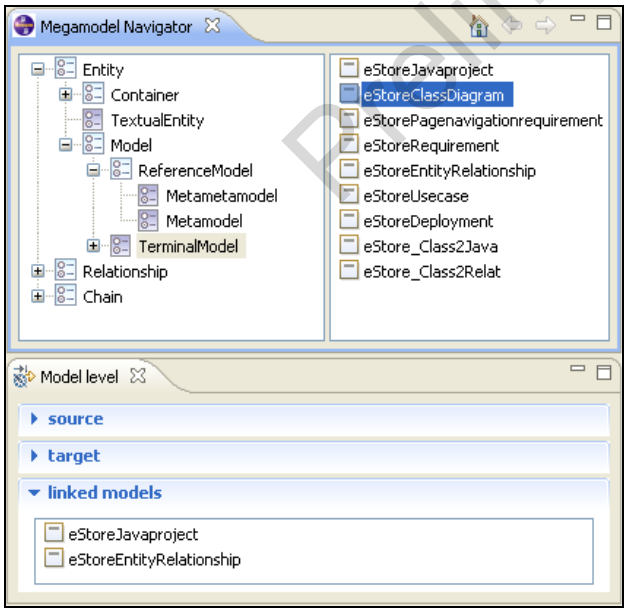

Figure 6. Model-level navigability

All kinds of links that have been presented above need to be persisted (i.e., they need to be stored and then potentially retrieved). We consider that representing these links directly inside the *Pet Store* models is not a suitable approach because the models are then polluted by additional traceability information. This is why, following our generic and extensible approach, the model level links are stored directly in the megamodel like any other relations. The model-element level links are represented as links in weaving models attached to the model element links.

Figure 6 and Figure 7 show how the AM3 megamodeling environment provides all these navigation links to the developer. When we select the *eStoreClassDiagram* model on the right part of the Megamodel Navigator, related models immediately show up the *Model Level* view. We can then easily navigate to the related *eStoreEntityRelationship* and *eStoreJavaProject* models or open up any of them using their corresponding editor. In Figure 5, the *ClassDiagram* model has traceability links to the *Entity-Relationship* and *JavaProject* models (*linked models* part of the *Model Level* view). Zooming on the *ClassDiagram* to *JavaProject* traceability link available from the screenshot of Figure 6 there is a model element-level traceability link between the *Category* design class and the *Category* type from the Java project model (Figure 7).

All the work presented in this section, including the AM3-based megamodel, editors and the sample models, is available from the Eclipse-GMT AM3 project webpage [1].

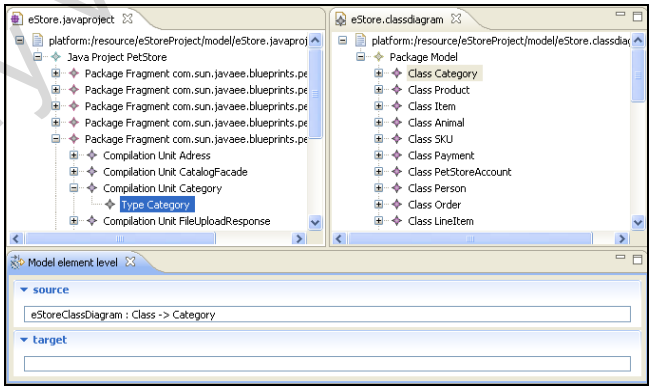

Figure 7. Model element-level navigability

We insist in this example on the way one may navigate all the models developed in a project (with different DSLs) at a given time. Thus, the available links may be used in order to ensure part of the inherent coordination existing between the different languages. However, the main interest of this work has been to demonstrate that this set of artifacts together with their mutual relationships and metadata, all providing the cartography of the coordination, may be represented as a megamodel (i.e. a model) conforming to a given (extensible) metamodel. This result is quite strong and new applications of it are still being found. For example, it is possible to check this megamodel for global consistency and this can be easily expressed with a model transformation. Similarly, it is possible to project this megamodel on a given display surface or to extract any kind of simplified view from it, again using a single model transformation.

5. RELATED WORK

The problem of the coordination between multiple related artifacts is not a new one. In code-centric environments, we also have to deal with multiple types of mostly textually-oriented artifacts such as source code, SQL code, Web service descriptions (WSDL), language dictionaries or any DSL. All these artifacts, their integration and their interrelations can be managed in different ways. Classically, the management of these relations has been performed using an Integrated Development Environment (IDE). Usually, a single data (meta)model is defined for all artifacts and only predetermined relationships are available. Considering some other approaches, the coordination is realized thanks to a dedicated DSL such as in [25]. Our approach focuses not on textual artifacts but more generally on models and is based on an extensible metamodel that allows to potentially link any types of model (i.e. models which conform to any metamodel) without modifying the core of the tools. Typical services offered by classical IDEs are 'find usages' and back/forward navigation throughout code. Our initial implementation is a first step in the direction of offering similar kinds of services for model-centric approaches.

In order to implement our solution for coordination, model traceability (which is an application of model weaving) and global model management (based on megamodeling in our case) have been combined. In the next two subsections we discuss related works in these two different but somehow interrelated areas.

5.1 Traceability

In order to enable navigation across different DSLs so that they can be coordinated, we use traceability links, stored as weaving models, between models and model elements. Many traceability approaches focus on requirements tracing and many different metamodels have been proposed [23], some of them trying to distill a standard metamodel [24]. Our model weaving approach offers a basic traceability metamodel that can be extended to fit almost any traceability scenario. Since the navigation operations are only dependent of the basic metamodel, it works transparently to the specific traceability needs of a project, meaning that any traceability links can be used for navigation.

In [22], a number of problems related to the management of many artifacts that have different purposes and different notations (describing parts of the same system, similarly to our problem statement) are discussed. They have developed an approach for traceability and inconsistency management between descriptions of software requirements, UML-style use case models and black-box test plans. Traceability links are often used to notify developers if changes might have to be propagated throughout the chain of artifacts, i.e. if the different artifacts have to be coordinated. Our approach can be used to accomplish the same goals if each artifact is described by a DSL. Developers can thus easily navigate between artifacts to manually analyze possible required changes. Automatic analysis of change impact could be added as an extension to our tools.

5.2 Global Model Management

The concept of a megamodel was proposed in [9] and in [10]. The Eclipse AM3 project [5] has served as a cooperative framework to experiment with concrete implementations of various forms of megamodels. The global view presented by a megamodel also uses the concept of a weaving model proposed in [11]. The idea of a joint use of megamodeling and model weaving was first presented in [12]. However, this work was really exploratory and at the time mostly focused on traceability in model transformation. This initial idea has now been generalized to any kinds of possible relationships between models, so that many different underlying coordination problems can be addressed. There have been several mentions of using megamodeling techniques in recent works when dealing with complex situations. In [14], an example of a megamodel describing model transformations is given, as well as a megamodel more focused on the model-driven evolution of software architectures. In [15], a megamodel is used in order to define families of reusable components. It refers to several metamodels and model transformations organized in an architectural framework. This megamodel allows formalizing the relations between different languages and thus provides some kind of support for coordination. In [16], an important network of transformations is described within the context of a rule-based management system. The paper illustrates that the development of a set of bridges between different DSLs implies the production of a high number of different artifacts (metamodels, projectors, transformations, etc) whose coordination is difficult to ensure and maintain without any automated support. Applying MDE techniques to the management of a high number of MDE artifacts naturally suggested using megamodeling techniques in this case.

As far as we know, there are not so many existing global model management approaches. However, some available ones have similarities with the megamodeling approach. For instance "macromodeling", as described in [17], proposes to define and use hierarchical models whose elements denote models and model relationships: as a consequence, a macromodel can be seen as some kind of megamodel. Moreover, this approach suggests considering macromodels for checking integrity constraints on the represented models and relationships. This is also the type of verification that can be performed from a megamodel, for example by applying a validation (model-to-model) transformation on it and generating a diagnostic model as a result.

The work described in [18] is also somehow related to global model management, even if it does not make use of explicit megamodels. The precise analysis of a significant application (the *Apache Open for Business OFBiz* framework) leads to identify a set of seventeen different DSLs used in this context. These languages belong to different technical spaces (XML, grammarware, modelware), but they are not disjoint. The work presented there is a strong motivation for studying the problem of coordination between the programs written in these different languages. An experimental tool for navigating them is described (SmartEMF) which is quite similar to the browser presented in our work. The main difference with our work is that, in their case, there is no global representation of the situation by a (mega)model, which leads to the direct handling of the connections between artifacts by the various technologies. Moreover, as the underlying metadata is not explicitly expressed as a model like in our approach, this information cannot be directly reused for other purposes (validation, analysis, code generation, documentation generation, statistics and reports generation, etc).

6. DISCUSSION

The work presented in this paper is a step in the direction of providing a generic and extensible inter-DSL coordination solution. A prototype based on the AM3 megamodeling tool has been developed and some experiments have been performed considering the *Pet Store* application as an example. The first results are very promising: the cost to develop the proposed prototype is relatively low since our solution is based on a generic and extensible megamodeling environment. Moreover, we could easily add many new DSLs to our example without requiring a lot of additional effort.

Apart from the evolution of the proposed example, the prototype in itself may be improved in different ways. From the navigability side, some user facilities such as navigation history support (Web browser-like) or additional navigation views may be developed. From a security side, user rights and restrictions of access on certain models and links may also be managed (this problem is somehow related to the notion of views and points of view on a system).

From a more theoretical point of view, the main lessons learnt in this work have been about the effective need of megamodeling techniques. Indeed, this exploratory work showed us the importance of megamodeling for inter-DSL coordination, but more generally for the management of any MDE process. The approach has a particularly high potentiality when used in combination with other basic MDE techniques such as model weaving or model transformation. MDE is based on the concepts of metamodel, metametamodel and model transformation. Considering our important experience with the model transformation applications (thanks to the ATL community), we came to the conclusion that the notions of model weaving and megamodeling should also be considered as essential concepts of MDE.

The three-level metamodeling stack [19] is now consensually accepted. However, there are still many degrees of liberty in deploying MDE in practical contexts. One choice still on the floor is about considering either a minimal or universal metametamodel (this corresponds to the level M3 of the stack). In our work, we have clearly opted for this alternative by using the KM3 proposal [6]. A second choice is about using either a universal or several specific metamodels (this corresponds to the level M2 of the stack). UML may be seen as a universal modeling language which obliges to handle a restriction device (the so-called *profile* mechanism) when the scope of the language needs to be more narrowly defined. We have clearly opted for the second approach where a metamodel is considered for defining the abstract syntax of a DSL [20]. These choices have a cost: nowadays we are facing the multiplication of small DSLs. As a consequence the penalty to pay is fragmentation, and megamodeling comes here as an obvious generic and extensible solution for coordinating a high number of related modeling artifacts.

One of the other interesting properties of our proposal is its low conceptual cost. Since weaving models (i.e. trace models) and megamodels are models, a lot of new functionalities can be achieved and plugged to the existing solution at a very low cost, thanks to the model unification principle and to all available MDE techniques. There are plenty of nice properties that come naturally out from the fact that a megamodel is a standard terminal model. Thus, it is possible to transform a megamodel into another model providing a different view on it. For example, some experiments we have already done allow to generate a graphical view, automatically obtained from the *PetStore* megamodel (using ATL), showing (using DOT) the *conformance* dependences between all the registered models.

We also discovered that megamodeling is necessary but not sufficient when used alone, as is model weaving. The joint efficient use of both techniques for coordination purposes is the main original contribution of the present paper. It has already been proved to work on a relatively small but significant application. It has now to be deployed on more important real life applications to be completely assessed. Following this idea, we already have observed several interesting properties of our proposal. In particular, it should scale up very nicely due to the recursive property of megamodeling: since a megamodel may represent any kind of models, it can represent megamodels also. Thus, we have the intuition that scalability issues in presence of a very high number of different artifacts may be addressed using the recursive property of megamodeling.

However, some other topics, which were not the subject of this paper but are related to the present work, still remain to be explored. One obvious topic is the production of the coordination information, e.g. of the traceability links both at model and model element levels. In this paper, this information was considered as already available in the required format but this is not always the case. More often, these data are created manually afterwards or automatically generated during the model creation process (for instance when a model transformation is running). Sometimes, they are also inferred from provided metadata using some heuristics, or imported from various inputs in different formats (for example from available documentation). Within our context, the deep study of all these possible cases still represents a lot of work. Another interesting topic is the checking and validation of the coordination information. For instance, by providing a more precise semantics to the traceability links (maybe by attaching explicit constraints to them), it could be possible to perform their verification and validation. Thus, the correctness and the relevance of the provided coordination information could be ensured before its actual use.

7. CONCLUSIONS

The naïve view where an application was entirely developed in one unique general purpose language (like Java) and only maintained in this language is now considered as completely unrealistic. Present day's applications make use of a high number of different DSLs (for example XML-based DSLs) and code written with these different DSLs does not disappear when the application is put in use. We first observed in this paper that this fragmentation problem is serious but should be rapidly addressed by defining scalable coordination schemes.

We presented our generic and extensible solution for dealing with the inter-DSL coordination problem. Our approach is based on the combined use of two different but complementary MDE techniques: megamodeling (or global model management) and model weaving. We have shown that our solution can be practically applied in order to manage the various artifacts related to a given context in a software project. As an example, we have used a simple situation based on the development of a sample software system and we have shown that many applications of

MDE generate or use a high number of artifacts. From there, we have reported significant progresses on handling the general situation. The example presented in this work shows how to use the Eclipse-GMT AM3 project as a coordination tool fitted with traceability and navigation capabilities.

It is also very important to notice that megamodeling has a much broader application scope than the one presented here. The inter-DSL coordination problem is just an example, among many others, of a possible use of a megamodel. The megamodeling approach, being by nature generic and extensible, can potentially have many different applications in all domains involving the management (and so the coordination) of a lot of modeling artifacts and their associated metadata. Dealing with the design, building, and execution of complex chains of model transformations is one example directly coming from the MDE field. Being able to efficiently manage hundreds (or even thousands) of heterogeneous models, retrieved from source code and documentation during the reverse-engineering process of a complex legacy software system, is another interesting problem we are currently exploring. Another application we are now experimenting on, which is also directly related to the reverse-engineering domain, is the use of a megamodel in order to represent the actual high-level cartography of different software platforms (in terms of tools, services, plugins, etc). This should allow obtaining all the different available dependences, in order to check the validity and the appropriateness of the overall platform for instance.

8. ACKNOWLEDGMENTS

This work is being supported by the IST-FP6 MODELPLEX European project [21] and the French IdM++ project.

9. REFERENCES

[1] The Eclipse-GMT AM3 (AtlanMod MegaModel Management) project's website: http://www.eclipse.org/gmt/am3/

[2] The Eclipse-M2M ATL (AtlanMod Transformation Language) project's website: http://www.eclipse.org/m2m/atl/

[3] The Eclipse-GMT AMW (AtlanMod Model Weaving) project's website: http://www.eclipse.org/gmt/amw/

[4] Jouault, F, and Kurtev, I: *Transforming Models with ATL*. In: Satellite Events at the MoDELS 2005 Conference, Montego Bay, Jamaica, October 2-7, 2005, pages 128—138. 2006.

[5] Allilaire, F, Bézivin, J, Brunelière, H, and Jouault, F, *Global Model Management In Eclipse GMT/AM3*. In: Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference, Nantes, France: http://www.sciences.univ-nantes.fr/lina/atl/www/papers/GlobalModelManagementInEclipseGMTAM3_Revised.pdf

[6] Jouault, F., Bézivin, J.: *KM3: A DSL for Metamodel Specification*. In: the 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, LNCS, Volume #4037, ISBN 978-3-540-34893-1, pp. 171-185. Bologna (2006).

[7] The Sun JAVA Pet Store demo: https://blueprints.dev.java.net/petstore/

[8] The AtlanMod Zoo http://www.emn.fr/x-info/atlanmod/index.php/Zoos

[9] Bezivin, J, Jouault, F and Valduriez, P: *On the Need for Megamodels*. In: Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM OOPSLA. 2004.

[10] Favre, J-M and Nguyen, T: *Towards a megamodel to model software evolution through transformations* (2004), SETRA Workshop, Elsevier ENCTS, 2004, pp. 59-74

[11] Didonet Del-Fabro, M, Bézivin, J, and Valduriez, P : *Weaving Models with the Eclipse AMW plugin*. In: Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany. 2006.

[12] Barbero, M., Didonet Del-Fabro, M and Bézivin, J. *Traceability and Provenance Issues in Global Model Management,* In: Proceedings of the Third ECMDA Traceability Workshop, Haifa, Israel 2007

[13] Bézivin, J, Jouault, F, Rosenthal, P and Valduriez, P. *Modeling in the large and modeling in the small*. In Model Driven Architecture: European MDA Workshops, volume 3599 of "Lecture Notes in Computer Science", pages 33–46. Springer-Verlag, 2005.

[14] Bas Graaf, *Model-Driven Evolution of Software Architectures*, PhD Thesis, 27 november 2007, Technische Universiteit Delft: http://www.st.ewi.tudelft.nl/~basgraaf/publications/PHDTHESIS2007.pdf

[15] Favre L., Martinez, L. *Formalizing MDA components*. LNCS Volume 4039/2006, in Reuse of off the shelf components, July 2006

[16] Didonet Del-Fabro, M, Albert, P, Bézivin, J and Jouault, F : *Industrial-strength Rule Interoperability using Model Driven Engineering* - INRIA Technical Report 00344013: http://hal.inria.fr/inria-00344013/fr/

[17] Salay, R, Mylopoulos, J and Easterbrook, S : *Using Macromodels to Manage Collections of Related Models*, 21st International Conference - CAiSE 2009, Amsterdam, The Netherlands, June 8-12, 2009

[18] Hesselund, A., Czarnecki, K., Wasowski, A. *Guided development with Multiple Domain-Specific languages*, MODELS, 2007, pp. 46-60, Nashville, USA.

[19] Bézivin, J and Gerbé, O : *Towards a precise definition of the OMG/MDA framework*. ASE'01, Automated Software Engineering, San Diego, USA, November 26-29, 2001.

[20] Kurtev, I, Bézivin, J, Jouault, F, and Valduriez, P : *Model-based DSL Frameworks*. In: Companion to the 21st Annual ACM SIGPLAN OOPSLA 2006, October 22-26, 2006, Portland, OR, USA. ACM, pages 602—616. 2006.

[21] MODELPLEX IST-FP6 European Project: https://www.modelplex-ist.org/

[22] Olsson, T., Grundy, J.: *Supporting traceability and inconsistency management between software artefacts.* In: Proceedings of the 2002 IASTED International Conference

on Software Engineering andApplications, Boston, MA, USA, November 2002.

[23] Gills, M.: *Survey of traceability models in IT projects.* In: ECMDA-TW, Proceedings of the 1st Traceability Workshop, November 2005.

[24] Limon, A., Garbajosa, J., *The Need for a Unifying Traceability Scheme*, Proceedings of the 1st Traceability Workshop (ECDMA 2005), November 2005.

[25] Groenewegen, D and Visser, E : *Declarative Access Control for WebDSL: Combining Language Integration and Separation of Concerns*, Eighth International Conference on Web Engineering, ICWE 2008, 14-18 July 2008, Yorktown Heights, New York, US