Model Engineering Support for Tool Interoperability

Jean Bézivin, Hugo Bruneliere, Frédéric Jouault, Ivan Kurtev

Model Engineering Support for Tool Interoperability

Jean Bézivin, Hugo Bruneliere, Frédéric Jouault, Ivan Kurtev

ATLAS group - INRIA - LINA
Université de Nantes - Faculté des Sciences
2, rue de la Houssinière, BP 92208
44322 Nantes cedex 3, France
Phone: +33 (0) 2 51 12 59 64
{bezivin, hugo.bruneliere, frederic.jouault, ivan.kurtev}@gmail.com

Abstract: In this paper we want to show how MDE (Model Driven Engineering) approaches may help solving some practical engineering problems. Our view of MDE is not based on the usage of huge and rather monolithic modeling languages like UML 2.0 but instead on small DSLs (Domain Specific Languages) defined by well focused metamodels. As a consequence we use a rather "agile" view of MDE where each tool is characterized by one of several metamodels and where the interoperability between tools is implemented by specific model transformation operations. We base our discussion on a practical illustrative problem of bug-tracking in a collaborative project involving different partners using tools at different maturity levels. We conclude by discussing the help that MDE approaches may bring to solving these kinds of situations.

1 Introduction

Today's software development practices rely on a dynamic set of languages, tools, and processes. Development teams are often geographically distributed and their work usually needs an integration step to form a complete product. However, before this integration is achieved the differences at the syntactical level of the created artifacts (e.g. different file formats, data representation schemes) and at the semantic level (differences in the underlying conceptual models, problems with language expressiveness, etc.) must be reconciled. In other words, the integration problem is preceded by a problem of interoperability at various levels.

One possible interoperability problem is the interoperability among tools. Currently the market offers a large set of software tools coming from various vendors for achieving a broadening spectrum of different purposes. The competition between vendors often leads to proprietary solutions or to solutions that slightly deviate from standards. Apart from this, a large set of partially supported or not supported legacy tools introduces an additional complexity to the problem.

In general, to enable interoperation between tools, the artifacts created with one tool should be somehow accessible from other tools. One possible solution is to apply transformations between various formats employed in different artifacts. The Model Driven Engineering (MDE) approach prescribes the utilization of modeling practices,

mainly models, metamodels and model transformations and should provide the required capabilities to perform such transformations.

In this paper we focus on the tool interoperability problem in the context of software development and more specifically on the interoperability of bug tracing systems. We use the AMMA (ATLAS Model Management Architecture) model engineering platform as a means for representing various tools and their underlying conceptual models and to specify and execute transformations among them. The more general goal we pursue is to generalize the experience gained in this domain and to try to attack interoperability problems in other domains too.

We have experimented with the model engineering approach to interoperability by using a case study in which three different tools were used for bug tracing. We developed bridges between these tools by using model transformations. The results showed that the approach is promising and many problems, in this particular area, can be efficiently solved on the basis of MDE principles.

All the artifacts described in this paper (mainly source of metamodels and transformations) are available as open source contributions in the Eclipse GMT project [1] together with the transformation engine and associated development tools (ATL).

This paper is organized as follows. In section 2 we present the motivating example treated in the paper. Section 3 describes our approach to interoperability based on model engineering. In this approach metamodels are built for every tool and transformations between the models created by the tools are defined. Section 4 presents the metamodels and section 5 presents the transformations. Section 6 presents a discussion and a short comparison with related work. Finally, section 7 concludes the paper.

2 Motivating Example

Problems of tools interoperability can be found in many and various domains. Software quality control is one of these domains because several tools are often used to ensure the quality of a software product. Since this is a quite wide domain, in this paper we will focus on a sub-domain in order to illustrate our approach.

We take an example of “bug-tracing” or “bug-tracking” in the case of a software product development. Assume that three teams are currently working on the same product at the same time but on different modules of this product. Teams may be geographically distributed, may have different levels of maturity of the used development process, may have different experience for the team members, and may consequently use different tools. The global situation may be described as follows:

- Team “A” is developing the first module by using an Excel workbook with a specific format to track or trace bugs;
- Team “B” is working on the second module and uses Bugzilla [2] which is a free bug-tracking system;
- Team “C” is developing the third module and uses Mantis Bug Tracker [3] which is another free bug tracking system;

A fourth team (that we name Team "D") must integrate the various modules developed by this three teams into a complete software solution. It also has to deal with all the bugs that have been detected but not yet resolved for each module.

The problem is that each team has used a different tool for keeping track of bugs. So in that case, how to succeed in centralizing bug-tracking, i.e. how to be able to interoperate from a tool to another without losing critical information about detected bugs? Furthermore the level of maturity of each team may dynamically evolve, each one learning from the global project. A given team may thus upgrade at some point in time its practices to those used by another one.

The AMMA model engineering platform (ATLAS Model Management Architecture) [4] offers a means of solving this type of problems by using metamodels and transformations between models conforming to these metamodels.

3 Proposed Approach

Our approach to interoperability between bug tracking tools is based on the principles of model engineering. In this approach we treat the information described by a bug tracking system as a model that conforms to a given metamodel (usually specific to the tool being employed). This makes it possible to use model transformations to convert the description of bugs from one tool to another. The AMMA platform provides the capabilities to define metamodels and to define model transformations. It is built on top of Eclipse Modeling Framework (EMF) [5]. The general overview of the approach is presented in Figure 1.

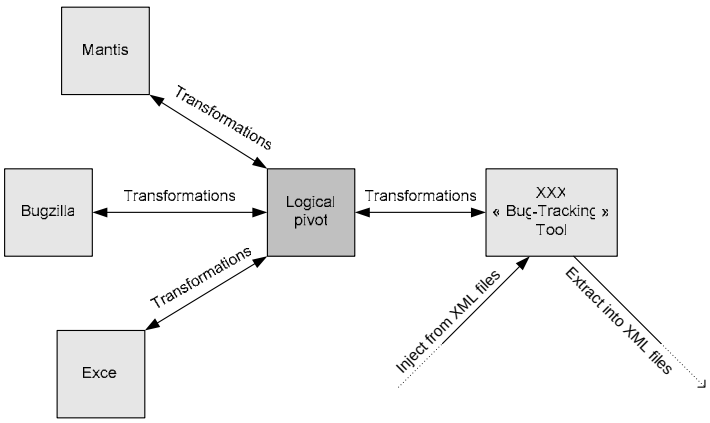

Figure 1 Overview of our model engineering approach to interoperability

In this approach, each different "bug-tracking" tool is described by a metamodel. Each tool's metamodel is linked to others by the "logical" pivot: transformations based on these metamodels to the pivot and from the pivot to the metamodels are implemented. This pivot is also a metamodel. It abstracts a certain number of general concepts about "bug-tracking". Moreover, because in this example each tool uses an

XML format to import/export data, we need to create an XML metamodel[1]. The aim of this metamodel is to make possible the injection of the content of an XML file into a model and the extraction of the content of a model into an XML file. Each model that we will have to handle will conform to one of these metamodels (tool's, pivot's or XML's one). Thus there is a transformational bridge between each of the tools shown in Figure 1 and, in this manner, each of these tools can interoperate with the others.

As a consequence, it is possible to easily add a new tool to the previously described system by:

- creating the associated metamodel;
- building the bridge (composed of two transformations) between this metamodel and the logical pivot metamodel;
- making the XML injector/extractor for this metamodel;

After we have set up the principles of our approach, we are able to present the metamodels and the transformations that have been implemented. Section 4 presents the metamodels and Section 5 explains the model transformations.

4 Metamodel Support

Metamodels (lying at the M2 level of the OMG metamodeling stack) are entities used to define domain specific languages (DSL) related to data formats and tools. In the context of the AMMA platform metamodels conform to KM3 (Kernel MetaMeta-Model), which is a metametamodel (M3 level) close to Ecore [6] and EMOF 2.0 [7]. KM3 provides a textual concrete syntax quite similar to the Java notation.

In our approach we may distinguish several kinds of metamodels:

- Input/Output metamodels that in most of the cases describe the format of input/output files such as XML files;
- Physical metamodels that describe the domain specific languages used by tools;
- Logical metamodels that are created to generalize a certain number of concepts common to several tools and thus to several DSLs;

This section provides the descriptions of the metamodels used in our example by respecting the classification given above.

4.1 The Input/Output Metamodel: XML Metamodel

Microsoft Excel, Bugzilla, and Mantis (and a lot of tools in general) do not natively use XMI (the OMG standard for serializing and exchanging models and metamodels). Instead they use a general XML format to import/export data. It is thus necessary to

[1] In the AMMA platform, XML is considered as a technical space with standard projectors (injectors/extractors) to/from the base MDA technical space. Other different technical spaces (e.g. EBNF) could have been similarly considered.

define an XML metamodel in order to be able to inject the content of XML files into models and to extract the content of these models into other XML files.

The simple XML metamodel we use is described in Figure 2.

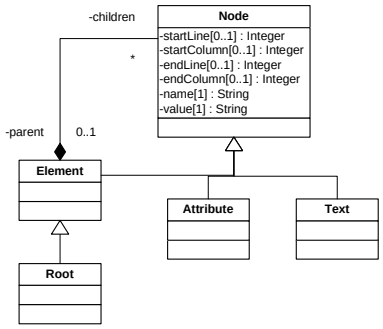

Figure 2 A simple XML metamodel

In this metamodel each element is a node which is identified by its name. The root element is the root of an XML document. It can contain several elements which can, in turn, contain other elements. Moreover, each element can have a list of attributes and text nodes.

4.2 The Physical Metamodels

In our example, we have three metamodels regarded as physical metamodels. Each different "bug-tracking" tool or general tool used for bug-tracking (Bugzilla, Mantis Bug Tracker, Excel) used by a team is associated to a metamodel of this kind. This subsection presents in more details these three physical metamodels.

4.2.1 SpreadsheetMLSimplified (Excel)

Microsoft Excel uses an XML language called *SpreadsheetML* to import/export Excel workbooks in the XML format. This is a very complex XML language defined by several XML schemas allowing to deal with most of the information saved in the Excel binary format '*.xls*'. Since handling of relatively small metamodels is more efficient, we only use a part of this language: a sufficient subset to preserve the global structure of an Excel workbook and to contain the raw data.

The simplified metamodel of Excel called *SpreadsheetMLSimplified* is presented in Figure 3.

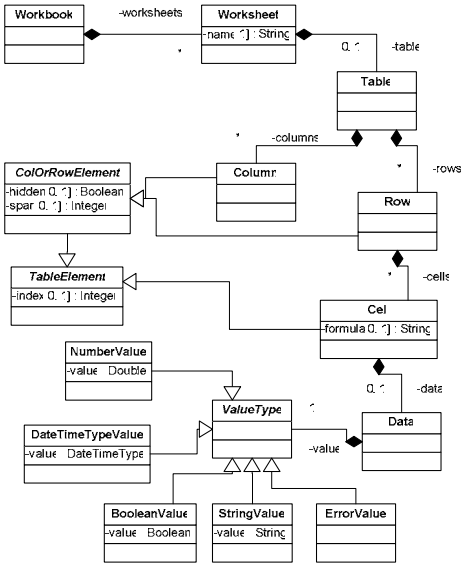

Figure 3 Simplified version of the Excel metamodel (SpreadsheetMLSimplified)

According to this metamodel an Excel file is a workbook. A workbook can have several worksheets. Each worksheet can have a table which is composed of table elements (columns, rows and cells). A table contains columns and rows. Each row consists of several cells. The cells contained in rows store the raw data of a given type.

4.2.2 Bugzilla

Bugzilla is a free “defect-tracking” or “bug-tracking” system originally developed by the Mozilla Foundation. A huge database allows it to store a large amount of information about a lot of bugs. These data are too complex to be easily handled by SQL requests. However, it is possible to use a Perl script to import/export bugs’ data in a simpler XML format. The data in XML files conform to a simple DTD.

The simple Bugzilla metamodel presented in Figure 4 is inspired by this DTD.

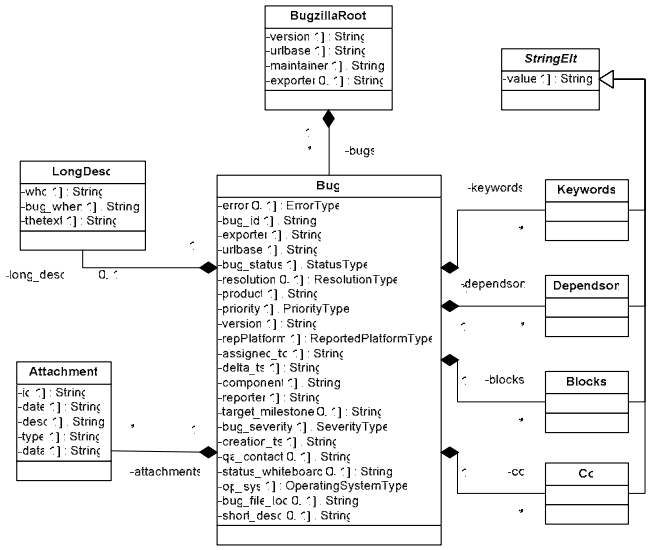

Figure 4 Simplified version of the Bugzilla metamodel

A Bugzilla model is a set of bugs. Each bug is identified by an “id” string and contains much information about the bug itself but also about the people who deals with it, the software product that is concerned with, etc.

4.2.3 Mantis

Mantis is a web-based bug-tracking system written in PHP that uses a MySQL database. Once again, such as for Bugzilla, the database is described by a complex relational schema. Mantis also allows importing/exporting bug data in XML files. The XML files conform to an XML schema. The Mantis metamodel inspired by this schema is presented in Figure 5.

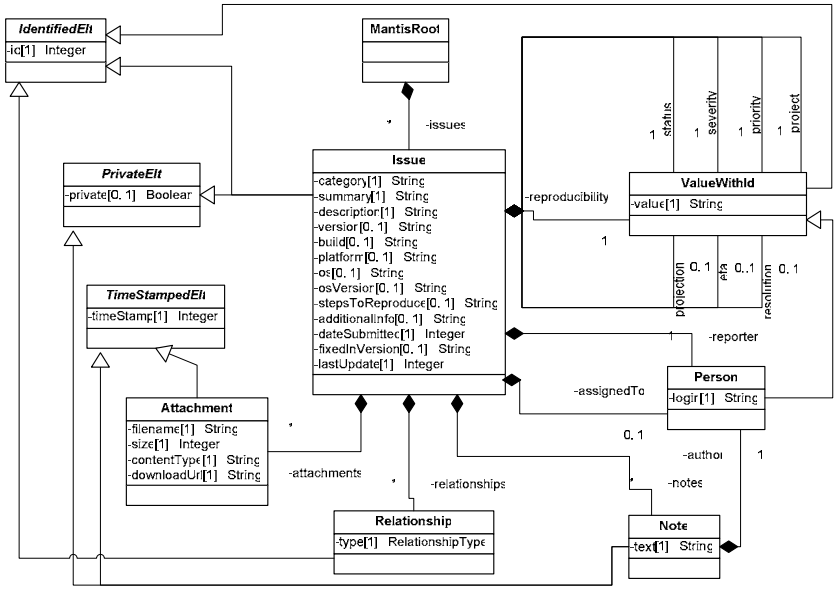

Figure 5 Simplified version of the Mantis metamodel

A bug in Mantis is named “issue”. Consequently, a Mantis model is a set of issues, each issue being identified by a unique number. Similarly to Bugzilla, an issue contains much information about itself, its software product, etc.

4.3 The “Pivot” Logical Metamodel: SoftwareQualityControl

By looking at the Bugzilla and Mantis metamodels, we find a lot of similarities. The information contained in a Bugzilla “bug” or in a mantis “issue” is often based on a similar structure. This information is also quite similar to the one contained in the specific format of Excel workbook in our example. That is the reason why it is necessary to create a logical metamodel which will be used as a “pivot” to facilitate the construction of bridges from a metamodel to another. By this skew, it will be simpler to improve interoperability between the various tools related to these metamodels.

Moreover, it would be interesting to allow the management of software quality control in general, not only its bug-tracking aspect. So the metamodel has to be general enough to easily allow adding of new types of software quality controls even if, for our example, we only need a “bug-tracking” logical metamodel.

The logical metamodel we created is depicted in Figure 6.

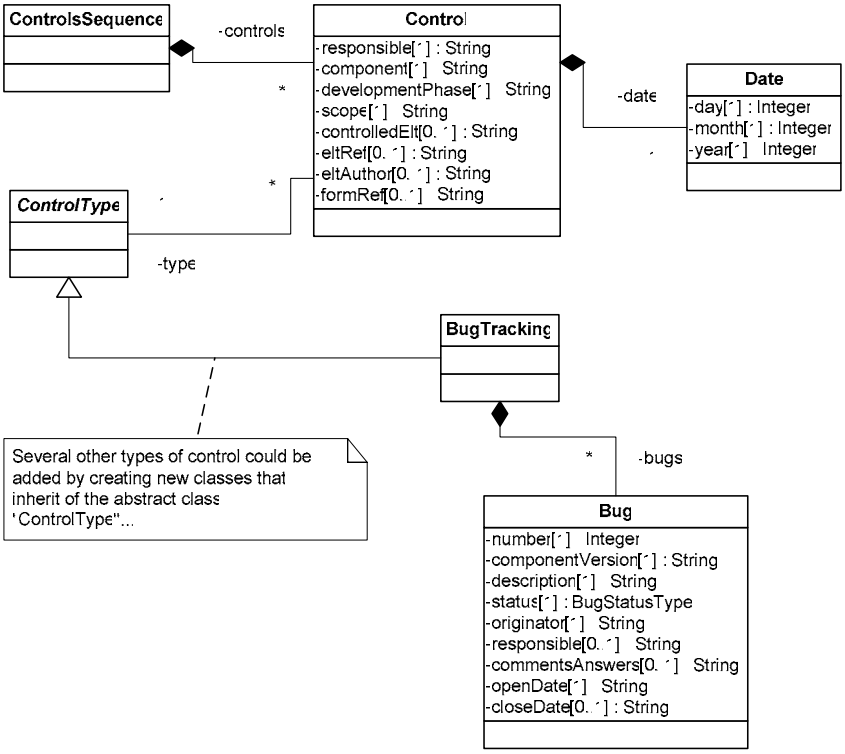

Figure 6 Software quality control metamodel

A software quality control model is composed of several controls. Each control is defined by specific information about the component and the element which are concerned, about the person who is responsible for the control, the date, etc. The main information is the type of the control. It determines what kind of actions has been performed and consequently what kind of data have been saved. In the case of our example, we only create *BugTracking* type but it could have a lot of other control types. In this type, the control consists of a set of bugs in which each bug is identified by a unique number. The information associated to a bug is identical to the one we can find in Bugzilla and Mantis.

5 Model Transformations

Model transformations are the second type of components in our proposed approach to solving problems of interoperability between several bug-tracing systems. In the context of AMMA platform transformations are written in ATL (ATLAS Transformation Language) [8]. ATL is a QVT-like language [9] for transformation of models (M1 level), which conform to metamodels (M2 level) previously written in KM3 (see section 4).

5.1 Overview

In the previous sections we described a library of small metamodels related to bug-tracking. We can use ATL to implement bridges between the different tools of our example which are Microsoft Excel, Bugzilla and Mantis. Several bridges are possible and interesting to implement. We have experimented with “Excel-to-Bugzilla” and “Excel-to-Mantis” bridges for this paper. These bridges are implemented as chains of model transformations. An overview of the bridges is given in Figure 7.

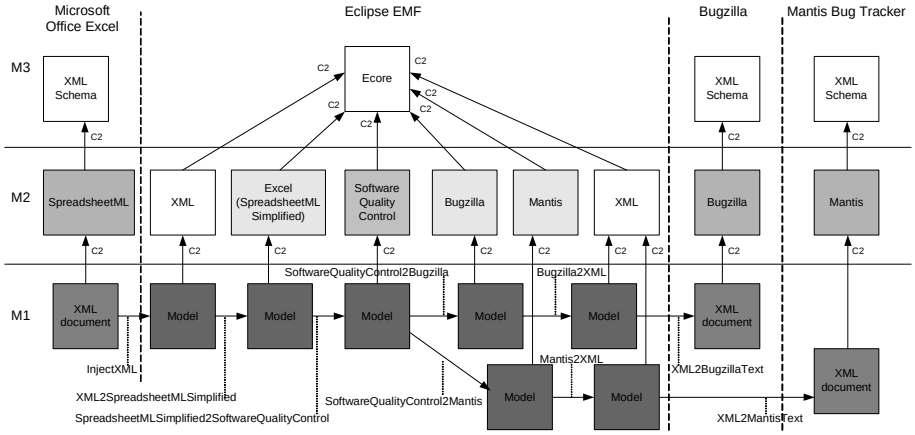

Figure 7 Excel-to-Bugzilla and Excel-to-Mantis bridges

In fact we do not implement direct bridges between these tools. As was mentioned in section 3 we use the logical metamodel SoftwareQualityControl as a pivot and therefore every model used in one of the tools is transformed to an intermediary model conforming to the logical pivot.

We can see that an Excel file in XML is the entry point of the two bridges. This file has to be injected into an XML model before being transformed into an Excel (SpreadsheetML) model and a SoftwareQualityControl model. Then, this Software-QualityControl model can be transformed into Bugzilla and Mantis models. Finally, these two models have to be transformed into XML models in order to be extracted into well-formed XML documents. These files are the exit points of the two bridges. Next sections explain each transformation in details.

5.2 Excel to Software Quality Control

This transformation is common to the two bridges. Here is the point of using the logical metamodel (*SoftwareQualityControl*) as a pivot. In this manner, we only need one transformation instead of two. Moreover, the creation of a new bridge to another tool will be facilitated because a part of this bridge is already made.

To make this transformation we proceed in three steps. There are detailed below.

5.2.1 First Step: Inject XML

Microsoft Excel is able to save its files as XML documents. So XML is the input metamodel of our two bridges. That is the reason why we first need to inject the content of the XML file in an XML model (conforming to the simple XML metamodel described in Figure 2). The XML injector is integrated into an Eclipse ATL plug-in from the standard ATL perspective.

5.2.2 Second Step: XML2SpreadsheetMLSimplified

This transformation is a simple mapping from an XML source model representing the content of an Excel file in the SpreadsheetML dialect to a SpreadsheetMLSimplified target model. As the target model has to conform to the simplified SpreadsheetML metamodel (Figure 3), the main work of this transformation is preserving the global structure of the Excel workbook in order to store only the raw data. All the information about styles, formatting and printing options is removed in the target model.

5.2.3 Third Step: SpreadsheetMLSimplified2SoftwareQualityControl

Now that we have a SpreadsheetML source model, we are ready to transform the data contained in the Excel workbook into a SoftwareQualityControl target model (conforming to the metamodel described in Figure 6) which will be then used as a "pivot". As the format of the Excel table contained in the source model is well-known, we can extract the required information quite easily from it and construct the target model.

At the end of this step, all the necessary information stored in the Excel workbook has been recovered and correctly formatted into the target model.

5.3 Software Quality Control to Bugzilla

This transformation is specific to Bugzilla bug-tracking system. It follows the Excel-to-SoftwareQualityControl transformation structure. Once again, it is composed of three steps which are detailed below.

5.3.1 First Step: SoftwareQualityControl2Bugzilla

The source model is the SoftwareQualityControl "pivot" model. Most elements of it, such as the bug number, and the bug status can easily find their equivalent in the Bugzilla target model. But for some other elements it is more difficult to find an equivalent. This can cause a loss of information in certain cases. Moreover, some required attributes in Bugzilla have no equivalent in the source model so they have to be initialized with a default value.

The aim of this transformation is to construct a valid Bugzilla model from the "pivot" model. So, in all cases, the generated Bugzilla model conforms to the Bugzilla metamodel (Figure 4).

5.3.2 Second Step: Bugzilla2XML

Files used to import/export Bugzilla's data are XML documents: XML is the output metamodel of the "Excel-to-Bugzilla" bridge. As a consequence we need to transform the source Bugzilla model to an XML target model that conforms to the simple XML

metamodel provided in Figure 2. The mapping between Bugzilla and XML is really obvious since the Bugzilla metamodel is based on a DTD. The content of the produced XML model conforms to the Bugzilla DTD.

5.3.3 Third Step: XML2BugzillaText

The produced XML model must be extracted to an XML file in order to be used by Bugzilla. The extractor is based on an already existing "XML2Text" transformation that generates a valid and well-formed XML document.

The end of this step is the exit point of the "Excel-to-Bugzilla" bridge.

5.4 Software Quality Control to Mantis

This transformation is specific to Mantis Bug Tracker. It follows the "SoftwareQualityControl-to-Bugzilla" transformation structure. To make this transformation we also proceed in three steps which are described below. Because this transformation presents many similarities with the "SoftwareQualityControl-to-Bugzilla" one, we will not describe it in detail.

5.4.1 First Step: SoftwareQualityControl2Mantis

The observations we made for the "SoftwareQualityControl2Bugzilla" transformation are also valid for this one. However, it is important to note that Bugzilla and Mantis metamodels (Figure 4 and Figure 5) have some differences: some elements in the Mantis metamodel, for example "reproducibility", do not have any equivalent in the Bugzilla metamodel.

In all cases, the produced Mantis model conforms to the Mantis metamodel.

5.4.2 Second Step: Mantis2XML

For the same reasons as for the Bugzilla case, an XML target model is generated from the Mantis source model. The content of this target model conforms to the Mantis XML schema.

5.4.3 Third Step: XML2MantisText

As the "XML2BugzillaText" transformation, this one is based on the "XML2Text" transformation. The produced XML file is reusable by Mantis.

The end of this step is the exit point of the "Excel-to-Mantis" bridge.

6 Discussion & Related Work

The problem of interoperability has been addressed by different research communities. Different levels of interoperability may be identified: syntactic interoperability and semantic interoperability. Here we briefly outline some of the existing work in this area without claiming exhaustiveness.

In Business-to-Business electronic commerce different companies exchange electronic documents that may be written in different XML languages thus causing an interoperability problem. This problem may be solved by XSLT transformations or by splitting the transformation task at two levels as described in [11]. This splitting separates the syntactic and semantic mapping in two subtasks. By using only XSLT transformations these tasks are performed in a single transformation program.

A similar problem is observed in the area of knowledge representation where a domain may be described by different ontologies that carry the same meaning but are expressed in different ways. This problem has been addressed by applying transformations between ontologies [10].

Finally, in the area of integration of heterogeneous XML data sources a pivot conceptual model is sometimes built to abstract from the syntactical differences and to provide a common and abstract view over different sources. The approach described in [12] relies on ontology language for specifiying the pivot and on XPath as a base for transformations from the logical pivot and the underlying XML data.

The main difference of these approaches compared to ours is that they are usually performed in the context of only one technical space [13] (e.g. XML in [11] and ontologies in [10]), or in a fixed number of spaces. In our approach it is possible to deal with an arbitrary number of technical spaces and their associated tools provided that the metamodels employed by the tools are built and injectors of their content are defined. From this point of view an MDE-based approach allows a uniform treatment of various technologies on the basis of MDE principles.

Our approach allows a relatively easy addition of new tools that have to operate with the existing ones. The only thing that is required is to build new physical and input/output metamodels. After this, bridges from these models to the ‘pivot’ model have to be defined. This evolution scenario may be complicated if there are significant differences between the pivot and the newly added physical metamodel. In that case the pivot may need to be updated.

7 Conclusion

Many lessons learnt in this project may be applicable more generally to various contexts and goals. This section will try to outline some of them.

In our work we have a specific bias to using "agile metamodeling" with small metamodels. This contradicts many mainstream proposals that use large and predefined "one size fits all" metamodels like UML 2.0. We need more experiments to assess the advantages and drawbacks of each of these approaches named below "agile modeling" and "monolithic modeling".

Agile modeling presented here is based on the idea that a new DSL (defined by a metamodel) should be created (or reused) for each specific task or situation. On the contrary, monolithic modeling (mainly illustrated by typical UML 2.0 approaches) is based on the idea that a unique (or unified, or united) modeling language may be used for most activities. No new metamodel has thus to be defined. Instead we may possibly specify that we use only some part of the language (informal restriction) or that

we use some controlled extension to the language, by the way of UML profiles for example.

The monolithic modeling approach has some apparent advantages:

- No metamodel to create;
- No large library of metamodels to manage and maintain;
- One consistent language framework where all the interactions between components (e.g. between classes, use cases, statecharts, activities, etc.) have already been accurately specified;
- Only one language to learn;

Each of these advantages is a drawback for the agile approach which seems much more demanding:

- It needs metamodel designers, i.e. highly skilled personnel in the project;
- It has to manage the fragmentation of all these created or reused metamodels;
- The transformations are exogenous transformations (different source and target metamodels) whereas in the monolithic approach we have mainly endogenous transformations (i.e. UML-to-UML);

Our definitive choice of the agile modeling approach is based on the experience accumulated in the AMMA and previous model engineering projects:

- A transformation is more precise if the source and target metamodels closely fit the transformation itself. This is similar to typing issues in programming languages, metamodels playing the role of types and models the role of variables or parameters;
- Experimental evidence has shown that the work involved in defining a transformation is rather concentrated on the definition of the source and target metamodels. This is also related to the previously mentioned issue;

We are well aware of the remaining difficulties of agile modeling, mainly due to the fragmentation issue already mentioned. However we do believe that, in the end, this option will prevail because of several advantages:

- It is easier to work by modular extension, starting from small metamodels, than to work by restriction, masking the facilities that one does not need for a particular task;
- It is easier to define with accuracy a small metamodel than a huge metamodel;
- The work of defining a small metamodel (like Excel, Bugzilla or Mantis) consists mainly of explicating the understanding of various tools or situations. This work should have to be done anyway. Using DSLs and specific metamodels only offers a precise framework and notation to do it;
- When we will have the correct infrastructure to handle vast libraries of metamodels, the expertise gained in projects like the one described in this paper will be efficiently reusable and adaptable to similar contexts;

In the present work we have mainly considered tool interoperability based on data stream exchange, the format and content of these streams being captured by metamodels. A possible extension would be to deal with the dynamic behavior of these

tools, captured by other forms of state-based or event-based metamodels. Even if this is more complex to handle, the general approach would still be very similar.

8 Acknowledgements

This work has been partially supported by a grant from Microsoft Research (Cambridge) and by the ModelWare IST European Project #511731. We thank also all the colleagues of the ATLAS group that have participated in this work and specially, Freddy Allilaire, Marcos Didonet del Fabro, David Touzet and Patrick Valduriez.

References

1. Eclipse GMT Project (General Model Transformer), http://www.eclipse.org/gmt/
2. Bugzilla official site, http://www.bugzilla.org
3. Mantis Bug Tracker official site, http://www.mantisbt.org
4. The AMMA Platform, http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT/
5. Eclipse Modeling Framework (EMF), http://www.eclipse.org/emf/
6. Ecore, http://download.eclipse.org/tools/emf/2.1.0/javadoc/org/eclipse/emf/ecore/package-summary.html#details
7. Essential MOF 2.0 (Meta-Object Facility), http://www.omg.org/docs/ad/03-04-07.pdf Part III – Chapter 7. The Essential MOF (EMOF) Model
8. ATL, ATLAS Transformation Language Reference site, http://www.sciences.univ-nantes.fr/lina/atl/
9. OMG MOF 2.0 Query/Views/Transformations RFP, http://www.omg.org/docs/ad/02-04-10.pdf
10. Maedche, A., Motik, B., Silva, N., and Volz, R. MAFRA - A MApping FRAmework for Distributed Ontologies. EKAW 2002: pp.235-250
11. Omelayenko, B. and Fensel, D. A Two-Layered Integration Approach for Product Information in B2B E-commerce, In: K. Bauknecht, S. -K. Madria, G. Pernul (eds.), Electronic Commerce and Web Technologies, proceedings of the Second International Conference on Electronic Commerce and Web Technologies (EC WEB-2001), LNCS 2115, Springer Verlag, Munich, Germany, September 4-6, 2001, pp. 226-239
12. Amann, B., Beeri, C., Fundulaki, I., Scholl, M. Querying XML Sources Using an Ontology-Based Mediator. In proceedings of CoopIS/DOA/ODBASE, 2002
13. Kurtev, I., Bézivin, J., Aksit, M. Technical Spaces: An Initial Appraisal. CoopIS, DOA'2002 Federated Conferences, Industrial track, Irvine, 2002