

# Deploying MDD for Business Application Maintenance

Ahto Truu<sup>1</sup> and Mila Keren<sup>2</sup>

<sup>1</sup> WM-data, Saekoja 36a, Tartu, Estonia, [ahto.truu@wmdata.ee](mailto:ahto.truu@wmdata.ee)

<sup>2</sup> IBM Haifa Research Lab, Mount Carmel, Haifa, Israel, [keren@il.ibm.com](mailto:keren@il.ibm.com)

**Abstract.** It has been estimated by various studies [1,2] that over 50 percent of all software development effort is spent on maintenance projects. Therefore it is important to consider maintenance aspects when evaluating new development methodologies. Here we report on our experience of migrating an existing business application to the MDD (Model-Driven Development) methodology during a maintenance project.

## 1 Overview

The organization performing the experiment is a small ISV (Independent Software Vendor) that was acquired by a much larger organization during the time the experiment was in progress. However, the acquisition has not yet had any effect on the development processes within the organization and we believe the conclusions we draw should be applicable to many SMEs (Small and Medium Enterprises) within the industry.

The application used for the experiment is a fairly typical client-server desktop application, where a Java-based GUI (Graphical User Interface) and business logic layer is used to access and manage data stored in an SQL (Structured Query Language) based relational database.

The application has been in development since 2001. The initial version was produced with the effort of 30 person-months. Since then, several maintenance projects have been performed, with the total effort of another 30 person-months. The code base has reached the size of approximately 175,000 SLOC (Source Lines of Code).

## 2 Baseline Process and Tool-Chain

The development lifecycle is split between two companies: a foreign partner interfaces to the end users, defining business requirements and providing technical support, especially during deployment; the SME under investigation designs the software and implements it, essentially working as a subcontractor for the foreign partner.

The baseline process before the introduction of MDD did not use any modeling tools. The requirements were managed as text documents where color-coding

was used for attribution and history tracking. Architectural design was managed as a text document as well. User interface design was managed mostly as annotations on top of screen shots of the existing application. Database design was managed as a set of SQL DDL (Data Definition Language) scripts. The source code was hand-written and debugged using a generic Java IDE (Integrated Development Environment).

Test cases were managed using a spreadsheet where color-coding was used to track the status (passed/failed/skipped) of each test case in each testing round. External bug reporting (interface to the foreign partner) was managed using an NNTP (Network News Transport Protocol) server, while internally an off-the-shelf bug tracking application was used. All design documents and source files were versioned using a standard version control tool.

### 3 MDD Process and Tool-Chain

One of the constraints was that the interface to the foreign partner should not be disturbed by the deployment of the MDD process and tool-chain. Therefore, the lifecycle is still split the same way as before, and text documents and screen shots are still used to negotiate the requirements. However, once the requirements have been agreed upon, they are imported into a UML (Unified Modeling Language) model where they can be linked to the relevant use-cases for dependency tracking.

True to the MDD spirit, the tool-chain operates on two levels of UML models. The first model, the PIM (Platform Independent Model), contains the imported requirements, the use-cases with possible workflow transfers between them, and the business objects. For dependency tracking, the requirements are linked to the use-cases that must satisfy them and the use-cases are linked to the business objects that they manage or use. Because this is the information that is traditionally related to the work done by business analysts, this model is colloquially called the Analysis model within the project team.

The second model, the PSM (Platform Specific Model), is generated from the PIM using the ATL (Atlas Transformation Language) tool [5, 6]. The transformation creates several classes for each use-case and one class for each business data object in the PIM, to suit the existing framework and established design patterns. As this is the information that is traditionally handled by designers, this model is referred to as the Design model within the project team.

Both models have a custom profile applied to them. Stereotypes from the profile are used to express behavioural aspects of the use-cases and the business objects they are operating on. Attributes of the stereotypes are used to carry auxiliary information (for example, the labels and titles to be used in the GUI) from PIM through PSM to code. Both models and the profile are managed using the RSA (IBM Rational Software Architect) tool [4]. More details on the profile can be found in [3].

The Java and SQL code is then generated from the PSM using the MOFScript tool [7, 8]. Two related Java classes are generated for each class in the PSM: an abstract base class that contains all the generated attributes and methods, and

an empty child class as a placeholder for any hand-written code. All the classes follow a pre-defined naming convention which makes it easy to generate code that references them from other classes.

As the current tools do not support keeping manual changes during regeneration of the model, we have to completely avoid editing the PSM. While the tool support is no better for manual changes to the code, we can use the Java inheritance and overloading features to isolate the hand-written parts into separate files. Unfortunately, we're not aware of a comparable mechanism for the UML models.

With all the requirements, use-cases, and business objects collected into the PIM, and a highly customizable model to text transformation tool at our disposal, it is only natural that big parts of the project documentation are generated from the model. So far we do not see much benefit in generating documentation from the PSM, as it is mechanically generated from the PIM, and thus does not contain any additional information.

For the time being, most of the testing and bug tracking activities are performed as before. However, we do use a test generation tool to help the tester to design the test scenarios.

## 4 Lessons Learnt

### 4.1 Requirements Management

The weakest point in our limited implementation of the requirements management and business modeling phase is the handling of requirement changes. When a requirement changes, we only have two possibilities to proceed with the current tools: either to re-import the changed requirement and manually re-connect it to the dependant model elements, or to manually update the already-imported instance in the UML model.

We have found that in practice, it is most efficient to use the existing requirements stored in the UML model as a reference while negotiating with the foreign partner, and to aim to import the new requirements only when the chance of any further changes is minimal. Basically the same approach also goes for the case when a new requirement is a request to change an existing one: during the negotiation process, we keep the text document and the UML model open side by side and only update the model when a consensus has been achieved.

The benefit over the previous approach is that there is a central collection of all existing requirements. Admittedly, this is not really an MDD result, as requirements tools exist also outside the MDD world. However, we believe that for future maintenances, it is also important to have the trace links from the requirements to the use-cases and the business objects, and this would be difficult to achieve using an external non-model-aware requirements tool.

### 4.2 Model to Model Transformations

Similar to the requirements management, also for model to model transformations the hardest part is change management. As mentioned before, there is

currently no tool support to keep the manual changes to the PSM when regenerating it from the changed PIM. As a consequence, we have adopted the policy of not changing the PSM at all and editing the code only. However, in several cases changing the model would have been much less work and perhaps even resulted in better code in the end.

We expect this to be a significant area of future research and development for the tool vendors, as the need for the “diff” and “merge” for models is sure to be quite universal. Unfortunately, the problem is not trivial at all, and it remains to be seen how soon any significant progress can be made.

A specific issue to be aware of in the context of the ATL tool and UML models is the profile support. In general, ATL transformations are considered to be functional mappings, and the order in which the individual elements are transformed should not matter. However, applying a profile to a UML model effectively changes the meta-model at runtime and thus makes the evaluation order of transformation rules important.

The current workaround for this issue is to provide in the ATL language a way to specify that all the profile and stereotype applications are to be performed in a specific order after all other transformation work has been done, but this imposes a few additional constraints on the transformation. A preferred solution would be a specific UML driver for the ATL language that would handle this problem transparently.

### 4.3 Model to Text Transformations

As mentioned above, the issue of keeping manual changes to the generated code can quite easily be solved with the help of the inheritance and overloading features of the Java language. The documentation we generate from the models is XHTML, thus merging the manual changes from one version to the next could be handled using standard text-manipulation tools or the version control system, whichever is more convenient for the development team.

A point worth mentioning is that, unlike the ATL engine, the MOFScript does not have an interactive debugger. The only debugging support in the current version is printing trace messages out to the console. This is not a real problem for experienced users of the language, but could intimidate beginners. On the other hand, the MOFScript plug-in for Eclipse has an auto-completion feature in the code editor, which ATL lacks, and which makes it much easier for a beginner to explore the meta-model.

### 4.4 Testing

We looked into the possibility to have at least some of the test cases generated from the model, but this proved to be a difficult proposal. Most of the behavioural information is contained in the transformations (especially the model to code transformation) and merged with the source model during the transformation, based on the stereotypes found on the model elements.

For generating the test cases, this behavioural information would have to be somehow made available to the test generation tools. Also, the tool-chain is designed to allow the behaviour specified in the model to be overridden by manually added code, and this cannot possibly be covered by the test cases generated from the model.

So, we settled for generating general use-case level test scenarios to help the tester to achieve the desired coverage of the use-case transition graph. At the moment we do not yet have enough experience with this approach to tell how big the benefits are (or even if there indeed are any). Of course, model-based test generation is still a rather young area, so we may see significant improvements in methods and tools in quite short term.

## 5 Conclusions

The most important result so far is that, unlike any other modeling tools we are aware of, it is possible to have the generated code follow the patterns and conventions of the existing code base closely enough that we can move the project from code-centric to model-centric approach gradually.

This means we could get by with only modeling the use-cases that we needed to change within the maintenance project and leaving the rest of the code alone. The plan is to convert the remaining code during following maintenance projects, if and when they need to be worked on.

One issue certainly worth discussing is resource consumption. We have already spent almost 30 person-months to set up the MDD tool-chain and to create the partial model of the application. While this may seem really high compared to the effort spent over the past 6 years to develop the application in the traditional code-based way, two points should be kept in mind.

First, a significant portion of that effort went to testing and debugging of early prototype versions of the tools we used. Now that the tools have somewhat matured and end-user documentation has been improved, we expect any future attempts at such a conversion to be much more efficient.

Second, the team that embarked on the project did not have any previous MDD experience. Of course we had done a few UML diagrams before, but most of us had not seen or used, much less developed, an UML profile before. Again, with much more training materials available, for example on the ModelWare project website [9], and new ones popping up on a daily basis, we expect the learning effort also to be significantly lower now than it was two years ago.

Another issue that can't be overlooked in the context of industrial applications is that of tool and support availability and cost. At the time of writing, several components of the tool-chain — most notably the test generation tool and some glue code to improve the interoperability of the modeling tool and the model to model transformation engine — are not publicly available at all. However, the parts that should be most useful to small ISVs — the model and code generation engines — are available immediately, and even free of charge.

For our particular tool-chain, we expect the most likely next improvements to be upgrading to a proper requirements tool and more automation of the whole generation sequence. We are also interested in installing quality controls between the different transformation stages, most likely using the OSLO OCL (Object Constraint Language) tool [10].

## Acknowledgements

This work is partly supported by the European Commission in context of the ModelWare project (IST-2004-511731).

## References

1. B. Boehm. Software Engineering Economics. Prentice Hall, 1981.
2. T. C. Jones. Software Cost Estimation in 2002. CrossTalk, June 2002.
3. M. Keren, et al. MDA Approach for Maintenance of Business Applications. Proc. ECMDA-FA 2006, Bilbao, Spain (to appear).
4. RSA Home Page. <http://www.ibm.com/developerworks/rational/products/rsa/>
5. F. Jouault, I. Kurtev. Transforming Models with ATL. Proc. MoDELS 2005, Montego Bay, Jamaica.
6. ATL Home Page. <http://www.eclipse.org/gmt/atl/>
7. J. Oldevik, et al. Toward Standardized Model to Text Transformations. Proc. ECMDA-FA 2005, Nuremberg, Germany.
8. MOFScript Home Page. <http://www.eclipse.org/gmt/mofscript/>
9. ModelWare Home page. <http://www.modelware-ist.org/>
10. OSLO Project Home Page. <http://oslo-project.berlios.de/>

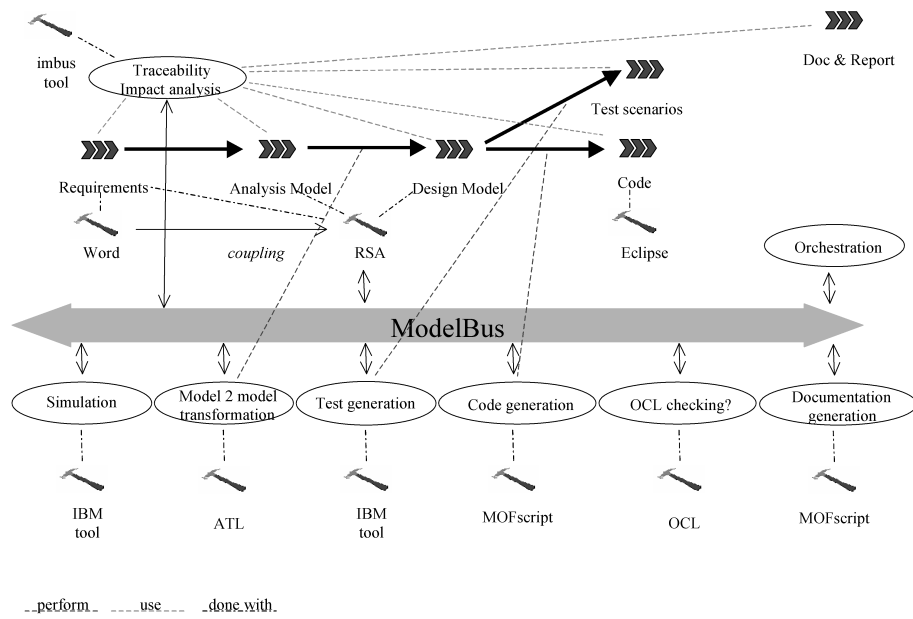


Fig. 1. MDD toolchain