Evaluation of UML Tools for Model-Driven Architecture

Janne Merilinna, Mari Matinlassi
VTT Technical Research Centre of Finland
P.O Box 1100, 90571-Oulu FIN

Abstract

Model-Driven Architecture (MDA) is about separating the specifications of the system from the platform the system is running on. This is done by presenting the system with one or more Unified Modeling Language (UML) models that may present different levels of abstraction. The change from one model to another is called transformation. In this paper, thirteen commercial and open-source UML tools were evaluated to find the most suitable one for realizing quality-driven architecture model transformation. The actual evaluation comprised two separate stages. First, the tools were studied from vendors’ website to find the most promising tools for further study. Second, three the most promising tools were selected for trial. In this stage, support for MDA and extensibility were observed. Ultimately Telelogic Tau/Developer was considered the most suitable tool mainly because it was the only one supporting platform independent development.

1. Introduction

Model-Driven Architecture (MDA) is about using modeling languages as programming languages rather than merely as design languages [1]. MDA uses multiple models at different levels of abstraction to isolate system specification from underlying platform that is for example operating system or middleware. In MDA, one model can be converted to another one of the same system by using transformation. Model transformation is described as a process of converting one model to another model of the same system [2]. Transformation can happen between abstract model and a less abstract model or vice versa. Transformation between models of the same abstraction level is also possible. In this paper, a few commercial and open-source Unified Modeling Language tools were evaluated to
find the most suitable one for realizing quality-driven architecture model transformation. That is, changing or varying quality requirements result in two or more alternative software architecture models. Quality-driven architecture model transformation occurs between alternative architectural models or sub models.

UML is specified graphical modeling language for specifying, visualizing and constructing software systems. By the time of writing this paper, the current adopted UML version is 1.5, but in the near future version 2.0 will be published. Perhaps the most significant addition in this new version from the perspective of describing software architecture [3] will be better support to express software decomposition e.g. class’s and components’ internal structure [4]. In addition, components are encapsulated from their environment and interaction with the environment takes the form of ports.

The new concepts for structural modeling are welcome in the community of software architects and therefore, tool vendors are keen on supporting these long-awaited features. Similarly, MDA or model-driven development “holds promises of being the first true generational leap in software development since the introduction of the compiler [5].” These promises even encourage integrating UML 2.0 and MDA in computer aided software engineering (CASE) tools. After all, tools significantly ease model-driven development. However, currently there exist only some previous study of different CASE tools capabilities, see e.g. [6]. As far as known no published research has been conducted from the perspective of supporting MDA and structural modeling with tools.

In this paper the following commercial and open-source tools were studied:

- ArcStyler 4.0
- ArgoUML (2/04)
- IUMLite 2.2
- Jvision 2.1
- Poseidon Pro 2.1.2
- Prosa UML 2004 Programmer edition
- ProxyDesigner 1.0
- Rhapsody Developer 5.0
- Rational Rose RealTime 6.5
- Rose XDE Developer Plus (2/04)
- Telelogic Tau/Developer 2.2.51
- Together ControlCenter (2/04)
- UMLet 3 beta
2. Model-Driven Architecture

The concepts of Model-Driven Architecture lies on three different types of models: Computation independent, platform independent and platform specific models.

Computation independent model (CIM) is a model, which shows the system in the environment where it will operate. Not too specific details of the system are presented, as typically this model is independent of how the system is implemented. For that reason CIM is sometimes called business or domain model [2].

Platform independent model (PIM) describes system completely, but it does not show any platform specific details. Platform independency is highly relative. For example, one considers for its platform operating system while for other it may be a “technology infrastructure represented by a well-defined programming model such as J2EE or .NET” [7].

In this paper a platform is generally considered to be the same as David Frankel is using in the book Model Driven Architecture [1]:

1. Information-formatting technologies, such as XML DTD and XML Schema
2. 3GLs and 4GLs, such as Java, C/C++ and Visual Basic
3. Distributed component middleware, such as J2EE, CORBA and .NET
4. Messaging middleware, such as WebSphere MQ Integrator and MSMQ

Platform specific model (PSM) can be described as a realization of PIM with all the details of the chosen platform. For example a CORBA specific PSM could be expressed in the UML profile for CORBA and a Web Services PSM could be expressed in WSDL [8].

2.1 Model Transformations

Model transformation is described as a process of converting one model to another model of the same system [2]. In MDA, one of the key transformations is from PIM to PSM, but also several other transformations are defined. In this paper, transformations where abstraction level is changed are called vertical transformations to separate from horizontal transformations where abstraction level remains unchanged. Next transformations from CIM to PIM, PIM to PSM and PSM to the code are explained briefly.
1. **CIM to PIM.** This transformation might be somewhat abstract as computation independent, or business, models are typically not appropriate to express all the details needed for PIM. For this reason PIM may be drawn separately, but all the requirements and other aspects defined in CIM are taken into consideration.

2. **PIM to PSM.** This transformation is used when the PIM is sufficiently defined and its function is secured. In this transformation platform specific issues are attached to the PIM to form PSM, which should be then completely aware of its platform.

3. **PSM to code.** This transformation is used when all the platform specific details are defined and the model is ready for actual implementation.

Horizontal transformation, that is transformation between models at the same abstraction level, is also possible. For an example PIM to PIM transformation is used when models are enhanced, filtered and specialized during design process [9]. Horizontal quality-driven architecture model transformation lies on the fact that same software functionality may be implemented with various architectures. The driving force between alternative architectures are quality requirements e.g. portability, extensibility or reliability. Switching from model to model requires a model transformation.

Transformations are based on mappings, which are defined as specifications for transformation, between the models. To clarify these mappings, consider for example PSM to code transformation where target code would be Java. In PSM there would exist some kind of list structure, which should be then implemented with Java. There are at least two classes that implement list structure: one is ArrayList and the other is LinkedList. There exists four ways to deal with this variability problem [1]:

1. **Same Always.** Always choose LinkedList or ArrayList.
2. **Engineer overrides.** One of the two choices is default, but can be overridden via a tagged value.
3. **Tag the PIM.** Tag is associated with the PIM instead of PSM. In this way PIM is actually no longer PIM, but synchronization between PIM and PSM is simpler.
4. **Abstract it.** Determine if it is possible to express the distinction in the PIM in a platform-independent fashion.

Similar mappings are associated into every model to make transformations possible. Horizontal transformations are also quite similar but mapping problems might be a lot different than in previous example. For example, consider situation where transformation takes place when performance requirements are changed. Choosing the right components in PSM stage is essential, but modification may also be required at the PIM. PIM to PIM mappings in this situation may not be straightforward, as the whole architecture of the model may change.
3. Evaluation Frameworks

3.1 The First Evaluation Framework

Table 1 presents the evaluation framework of the first evaluation phase. In architectural modeling, expressing classes’ internal structure and defining components interfaces are considered essential. For that reason UML version of the tools were observed. As UML 2.0 is at the current time in finalization phase and some uncertainty exists in how tool vendors does implement it at a moment, structural modeling ability was observed separately as well.

As it is expectable that there does not exist direct support for quality-driven model transformation and it may had to be done by self, tools had to provide some kind of extension interface. What kind of interface and what languages could be used for extensions were not considered as essential aspects in the first framework and for that reason were not observed. Just whether the tools are extendable or not were monitored.

As it is defined in MDA, models are ultimately transformed into code, for that reason code generation for at least one language had to be provided. As code generation is not a straightforward task, there exist two kinds of generators: full source code and code template generating generators. Full source code generators should generate all or nearly all of the code, while code template producing generators generates only class and function templates. Code generation for C, C++ and Java were observed, as it is assumed that these are the most common languages to be used. Only full code generators were considered.

Some features that could lighten and quicken development, maintenance and testing were chosen into evaluation framework. Automatic document generator, which should produce readable documents from designed model, was observed. Support for some kind of testing and debugging environment was observed too.

Table 1. The first evaluation framework

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML</td>
<td>What is vendor’s announced UML version?</td>
</tr>
<tr>
<td>StructM</td>
<td>Does the tool support structural modeling?</td>
</tr>
<tr>
<td>Ex</td>
<td>Does the tool provide extensibility interface for user defined plug-ins?</td>
</tr>
<tr>
<td>C</td>
<td>Does the tool provide support code generation for C?</td>
</tr>
<tr>
<td>C++</td>
<td>Does the tool provide support code generation for C++?</td>
</tr>
<tr>
<td>Java</td>
<td>Does the tool provide support code generation for Java?</td>
</tr>
<tr>
<td>Doc</td>
<td>Does the tool provide any automatic document generator?</td>
</tr>
<tr>
<td>Sim</td>
<td>Does the tool support any testing and debugging environment?</td>
</tr>
</tbody>
</table>
3.2 The Second Evaluation Framework

As the first evaluation based only on vendors’ datasheets and sales talk, few the same features listed in the first evaluation framework were selected to the second evaluation framework to ensure information correctness. Again, the purpose is to describe significant characteristics of the tool from the perspective of MDA and extendibility, not to show every lesser feature. The second evaluation framework is presented in Table 2.

UML version and especially structural modeling ability is taken into framework to make certain that the tools do support structural modeling like it is informed. This is because vendors often promises more than they deliver.

Extensibility is divided into two separate categories: extensibility interfaces and UML profile extensions. Extensibility interfaces are for accessing the tool through provided application programming interface (API) with some programming language. UML profile extension is for modifying and extending UML itself.

Platform independent and platform specific modeling is observed when tool’s support for Model-Driven Architecture is considered. At this time platform is considered to be any 3GL or 4GL, so being platform independent, the tool has to provide its own action language for describing model’s behaviour. In this way model can be compiled to any supported target languages. If no action language is defined, it is considered that the tool allows only platform specific modeling.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML</td>
<td>What is vendor’s announced UML version?</td>
</tr>
<tr>
<td>StructM</td>
<td>Does the tool support structural modeling?</td>
</tr>
<tr>
<td>ExtL</td>
<td>What languages can be used for tool extension?</td>
</tr>
<tr>
<td>Profiles</td>
<td>Does the tool provide support for defining new UML profiles?</td>
</tr>
<tr>
<td>MDA</td>
<td>In what extend the tool supports MDA?</td>
</tr>
</tbody>
</table>

4. Tool Evaluation

4.1 The First Evaluation – Literature Study

From the perspective of the first evaluation framework (see Table 1), every evaluated tool is reported in Table 3. Mark “X” means “yes” answer to the question presented in the first evaluation framework.
Table 3. Summary of the results of the first evaluation

<table>
<thead>
<tr>
<th>Tool</th>
<th>UML</th>
<th>Ex</th>
<th>StructM</th>
<th>C</th>
<th>C++</th>
<th>Java</th>
<th>Doc</th>
<th>Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArcStyler 4.0</td>
<td>1.4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ArgoUML (2/04)</td>
<td>1.3</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iUMLite 2.2</td>
<td>1.4</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JVision 2.1</td>
<td>1.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Poseidon Pro 2.1.2</td>
<td>2.0</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prosa UML 2004 Prog.</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ProxyDesigner 1.0</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rhapsody Developer 5.0</td>
<td>2.0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Rational Rose RealTime 6.5</td>
<td>1.4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Rose XDE Developer Plus (2/04)</td>
<td>1.4</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Telelogic Tau/Developer 2.2.51</td>
<td>2.0</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Together ControlCenter (2/04)</td>
<td>1.4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMLet 3 beta</td>
<td>N/A</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

UML is at this moment on the threshold of a new era, as version 2.0 is being published at April 2004. For that reason it would not be reasonable to consider further any tool that does not support UML 2.0. On the other hand one of the tools do support ports, structure modeling and other typical features of UML 2.0 even with version 1.4 and at the same time one vendor claims tool’s supported UML version to be 2.0 even when the tool does not support structural modeling. Due to this it must be considered that supported UML version does not reveal the real structural modeling capabilities and for that reason supported UML version cannot be used for evaluation.

The most effective way to drop out unsuitable tools was by checking do they support structural modeling. This criterion filtered ten tools out from further consideration and left Rhapsody Developer, Rational Rose RealTime and Telelogic Tau/Developer for further consideration.

All the remaining tools do provide some kind of extension interface and support for code generation for at least one language; therefore no tool is dropped out at this stage. Neither are any of the tools filtered out when support for document generation and for testing and debugging environment were observed as these were considered as minor aspects. Due to this all the remaining three tools were selected for more detailed evaluation.

4.2 The Second Evaluation – Empirical Study

Three tools, Rhapsody Developer, Rational Rose RealTime and Telelogic Tau/Developer, were selected for closer evaluation. Telelogic Tau/Developer and Rhapsody Developer were evaluation versions, downloadable completely free
from the vendors’ web site, whereas Rational Rose RealTime was a commercial version.

Evaluation was performed as follows. First every tool was installed and after that tools were evaluated one at the time. Tool evaluation consisted of two tasks: First, all included modeling tutorials were performed. Next, tool extension tutorials and also some own modifications were done to get more familiar with the extension interfaces.

From the perspective of the second evaluation framework, every evaluated tool is reported in Table 4. ExtL column contains languages what can be used for tool extensions. All the tools do support two ways of creating own plug-ins; either scripting within tool’s internal editor with Visual Basic for Applications (VBA), Rational Rose RealTime Scripts (RRRTS) or by Tool Command Language (TCL), or by accessing provided API with any OLE or COM enabled languages.

<table>
<thead>
<tr>
<th>Tool</th>
<th>UML</th>
<th>StructM</th>
<th>ExtL</th>
<th>Profiles</th>
<th>MDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rhapsody Developer 5.0</td>
<td>2.0</td>
<td>1</td>
<td>COM, VBA</td>
<td>X</td>
<td>PSM</td>
</tr>
<tr>
<td>Rose Technical Developer</td>
<td>1.4</td>
<td>X</td>
<td>OLE, RRRTS</td>
<td></td>
<td>PSM</td>
</tr>
<tr>
<td>Tau Generation 2</td>
<td>2.0</td>
<td>X</td>
<td>COM, TCL</td>
<td>X</td>
<td>PIM, PSM</td>
</tr>
</tbody>
</table>

Vendor’s announced UML version did not seem to be important aspect when ranking, as every tool does support ports and structural modeling whether the version is 1.4 or 2.0. Currently Rhapsody Developer supports structural modeling only when working with C++ language.

All the tools do support at least two extension mechanisms for own plug-ins, so the count cannot be used for ranking. Nor can the number of extension languages be used, as there are plenty of where to choose from in all cases. Actual API cannot be assessed either, as only trivial extensions were made during tool evaluation. Because of that, there were no experience implementing full-fledged plug-ins and for that reason there remained some uncertainty in every provided API. According to documents, every tool is relatively freely extendable, so the tools had to be considered as equals at this stage. Instead, support for creating new UML profiles could be considered.

Telelogic Tau/Developer is the only tool, which supports platform independent designing, as no target code has to be written in anywhere. Whereas in Rhapsody Developer and Rational Rose RealTime behavior of state machines and classes’ operations has to be implemented with target language. In Tau Technical/Developer platform specific issues has to be taken care of just when the model is integrated with actual environment.
When MDA is considered, none of these tools do support it in all its forms. Although Telelogic Tau/Developer allows platform independent developing, it does not support platform specific modeling as it is defined in MDA. There is no transformation from PIM to PSM defined in any way. PIM can be made into a PSM by writing inline target code into the model, but there is no transformation taken place. Actually, the model is more like a blend of PIM and PSM than just plain PSM, as platform independent action language still exists in the model. On the other hand if source code is considered as a platform specific model, then there exists clear transformation from PIM to PSM. Direct transformation from PIM to code is also defined in MDA, so no standard is violated. But the whole developing cycle from CIM to PIM and from there to PSM and finally to the code does not exist.

The two remaining tools do not support MDA actually in any of its forms, as at least one PIM has to exist when model-driven architecture is considered. On the other hand, if platform is defined as an operating system, then these two tools do support MDA. In summary, support for MDA is just a matter of definitions.

In overall, Telelogic Tau/Developer can be considered as the most suitable one, as it was the only one, which allows platform independent developing. Rhapsody Developer and Rational Rose RealTime are quite similar tools, as neither of them can be used for platform independent developing. Just some differences occur in structural modeling, as Rational Rose RealTime has its capsules where in Rhapsody Developer structure is designed straight into classes. There is actually no specific reason why one should be preferred above the other, but it has to be leaned towards Rhapsody Developer, as it supports creating new UML profiles. For that reason Rhapsody Developer is ranked the second and Rational Rose RealTime the third.

5. Conclusion

In this paper thirteen different CASE tools were studied to find out the most suitable for realizing quality-driven architecture model transformation. The tool had to support UML 2.0 version or at least structure modeling. Also extensibility interface was required. These two criteria filtered ten unsuitable tools out and left three for further evaluation. Telelogic Tau/Developer, Rhapsody Developer and Rose RealTime were evaluated one at the time and later on compared against each other.

Telelogic Tau/Developer allows platform independent developing by including its own action language to describe model’s behavior completely, but also target code can be written in any place if desired. Tau provides two extension interfaces for own plug-ins and allows defining new UML profiles, so extension possibilities are relatively unlimited.
Rhapsody Developer and Rational Rose RealTime are in the most part quite similar. Platform independent developing is not possible as no strong action language is included, so target code has to be written in to describe behavior. There are also restrictions in Rhapsody Developer’s modeling capabilities, as it does support class structure modeling only when working with C++. Both of the tools do support two extension interfaces for plug-ins, but only Rhapsody Developer allows defining new UML profiles.

Telelogic Tau/Developer seemed to be the most suitable tool; Rhapsody Developer was considered as the second and Rational Rose RealTime the last. None of these tools are incompetent, but the main reason why Telelogic Tau/Developer achieved the first place is that it makes possible platform independent developing, whereas it is not possible with the two other tools.

References