Model Driven Tool Interoperability in Practice

Jean Bézivin, Hugo Brunelière, Jordi Cabot, Guillaume Doux,
Frédéric Jouault, Jean-Sébastien Sottet

AtlanMod (INRIA & École des Mines de Nantes)
Nantes
France
+33 614 322 236

{Firstname.LastName}@inria.fr

Abstract.
Model Driven Engineering (MDE) advocates the use of models, metamodels and model transformations to revisit some of the classical operations in software engineering. MDE has been mostly used with success in forward and reverse engineering (for software development and better maintenance, respectively). Supporting system interoperability is a third important area of applicability for MDE. The particular case of tool interoperability is currently receiving a lot of interest. In this paper, we describe some experiments in this area that have been performed in the context of open source modeling efforts. Taking stock of these achievements, we propose a general framework where various tools are associated to implicit or explicit metamodels. One of the interesting properties of such an organization is that it allows designers starting some software engineering activity with an informal light-weight tool and carrying it out later on in a more complete or formal context. We analyze such situations and discuss the advantages of using MDE to build a general tool interoperability framework.

Keywords: MDE; Tool Interoperability; Metamodel; Model Transformation.

1 Introduction

The practices for producing and maintaining software systems are highly dependent on the toolset used. Software engineering is highly tool oriented. Tools are ubiquitous in this context. They implement methods. They can make a conceptual approach concrete. They are the support for roles. They can be called by human or automated actors, possibly from other tools. They can also be incorporated in other tools, allowing transitioning from simple tools to composite tools. They implement services. They define access API's and may encode part of the know-how and good practices. In the different perspectives that may be taken on the software and development landscape, the tool-centric one is certainly not the less important.

As an example, one old but still popular toolbox is the UNIX system. It is mainly composed of a set of elementary tools named commands. These tools can be

chained together and can be composed to build composite tools. Some of these tools have a certain degree of redundancy since the same goal may be achieved with the help of different tools. The UNIX toolbox contains elementary tools like *ls*, *awk* or *sort* as well as sophisticated ones like *sccs* utilities. Many users use their own know-how to develop new tools adapted to their concrete software practices.

Similarly, software engineering started by building software in development tools like assemblers, disassemblers, compilers, etc. Since then, the progresses of methodology and technology are paved with successive generations of tools, testifying the various steps of practice innovation. This paper discusses some new possibilities of model-based tools, i.e. tools at least partially based on Model Driven Engineering (MDE) [1], to improve current tool interoperability challenges. One of the characteristics of these tools is that they usually make use of explicit metamodels.

This paper is organized as follows. Section 2 presents the notions of metamodels and tools, and relates them. The issues of tool interoperability are presented in Section 3 while Section 4 discusses lessons learnt, related and forthcoming work. Finally, concluding remarks are given in Section 5.

2 METAMODELS and TOOLS

2.1 Metamodels

A metamodel is a simplified ontology. It is a graph of concepts and relations between these concepts. This is close to a grammar or to an XML schema or to a database schema. A metamodel may be used to discuss with precision a number of situations.

There are a number of languages to write metamodels like MOF, ECORE or KM3. All metamodels expressed in the same language may be easily compared, merged or jointly used in a number of abstract operations. They can also be transformed, including from one language representation to another one.

Each metamodel encodes a consensual agreement. There are standard metamodels like the ones defined by OMG (UML, KDM, SysML, BPMN, etc.). There is also a high number of non standard metamodels, sometimes in open source libraries like the AtlanMod zoos [5] containing more than 300 different metamodels in nearly 15 different notations. Some metamodels have a general purpose while others target a specific knowledge domain or software activity.

2.2 Tools

A tool is an entity that interfaces between two or more domains; facilitating more effective action of one domain upon the other. The screwdriver, invented in Germany in the late fifteenth century, typically interfaces between the operator's hand and the screw the operator wishes to tighten or untighten. Basic tools are simple machines. There is a very rich literature on tools and how the sophistication of these tools has accompanied and influenced the evolution of mankind. Usually, the use of a tool

supposes some know-how. For example, the operator needs to learn how to apply torque by rotating the handle of the screwdriver with correct positioning of the tip on the screw. Tools are constantly improved to make their usage easier and more efficient, for example electric motor screwdrivers.

A tool may share key functional attributes with one or more other tools. In this case, some tools can substitute for other tools. The Swiss knife is often taken as the symbol of a hypothetical universal tool. Sentences like "One tool does it all" or "All tools can be used as hammers." emphasizes the issues of exchangeability or specialization of tools.

2.3 Software Tools

A *software tool* is a software device run by a computer system and intended to be used by an actor to achieve a given goal in a given context. Any domain of human activity may use software tools, for example house building, airline design, payroll management, differential equation solving, etc. The actor may be a human or an automated agent. Examples of software tools are Catia, Microsoft Excel or Word, Simulink, Modelica, etc.

We may consider most applications found in what is commonly called AppStores to be a reservoir of hundreds of thousands of software tools for different purposes. Smartphones are often considered as containers for a large library of such software tools for agenda scheduling, road finding, music composition, and much more.

2.4 Software Development Tools

One particular case of a software tool is a programming tool or a *software development tool*. Related to that, one may also mention the Software Development Toolkit (SDK) that may be defined as a set of software development tools. A typical software development tool is a compiler, but a general software tool like a text editor may be used non-exclusively for software development.

2.5 Modeling and Model-based Tools

Some software development tools are *modeling tools* **[**1**]**, i.e., tools specifically intended to produce or manage models. The ATL model transformation language support system or the Rational Rose tool fit in this category. However, we are interested in a different category called *model-based tools*, i.e., tools that are implemented using MDE principles. There is obviously the possibility of having a model-based modeling tool, and the ATL tool is an interesting example. On the contrary, many modeling tools are completely implemented in Java, without any use of MDE principles, and this does not mean that they are not useful. Of course, an important category of tools are model-based tools that are not modeling tools and sometimes not even software development tools.

2.6 Crowded World of Tools

Let us restrict ourselves essentially to software development tools. An interesting exercise consists in making a raw list of tools used in a given context by a development team, for example: "*Papyrus; ATL; Polychrony; Protégé; Scicos; Subversion; Mantis; Ant; Autosar; Lustre; Excel; Word; OpenOffice; RequisitePro; Reqtify; Doors; Matlab; Simulink; ArtisanStudio; GoogleMail; Spyware; SimExplorer; SwingExplorer; TextUML; Celerity; Fossil; Tynamo; Panopticode; BOUML; Bugzilla; Trac; SharpForge; Make; Ant; Rake; Flowtracer; JTest; Krugle; CodeWarrior, Doxygen; Xcode; Javadoc; Swig; Insure++; Lex; Yacc; Bitkeeper; Clearcase; Git; SVN; Perl, Ruby, Awk; Python, REXX; Ruby; Shell; Tcl/Tk; Grep; Find; Emacs; Vi; XCode; IBMRAD; WinDev; Gdb; Valgrind, BinUtils; Javacc; CodeGear; ARIS; TIBCO; Mega; …*". From this plain raw list to a map of all the relations between these tools and of the various roles and actors that use them there is a significant distance. However this could show many of the characteristics of the process followed by the team. But the interest of such a brainstorming exercise is to show the diversity and the difficulty to organize these tools in well defined categories. More than that, a list of real world tools like above illustrates the need to cope with a high number of different scenarios, many of them presenting challenging interoperability problems.

2.7 An Ideal Model-based Tool

The tool interoperability goal becomes clearer when considering a hypothetical ideal tool and using it as a comparative basis. Even if such a tool does not exist, or even if it is impossible to build, this may be a very useful comparative benchmark. This will allow in particular separating the discussion in two threads: (1) how do ideal tools interoperate and (2) how to interoperate an ideal tool with a common tool. We thus consider an ideal metamodel-based software tool having the following properties:

- It is strongly associated to one or more metamodels (e.g., data, event, state)
- It has been built with the direct use of these metamodels. This means that the metamodel has not been loosely interpreted by humans and then converted in software code. Instead, the metamodel was used as a parameter in the direct generation of the tool.
- It is delivered together with the metamodels
- It is delivered with model injectors and extractors

Its various versions are delivered with a metamodel difference

3 Tool Interoperability

3.1 General Interoperability

Interoperability refers to the ability of two or more systems to exchange information and to use the information that has been exchanged [11]. These systems usually have been independently defined or designed. According to the nature of these systems, we have different kinds of interoperability. Typical examples are enterprises, information systems, software component, computer network, software application, combined hardware/software system, digital libraries and many more. The subject of this paper is tool interoperability. Dealing with this subject at an abstract level means that many of the conclusions may also apply to different system interoperability situations like enterprise interoperability.

3.2 Tool Interoperability

The "one engineer uses one tool" approach has been quite usual in computer science. It followed the hard specialization of tasks in the software development cycle. The typical example is the programmer using a compiler as his/her main tool. This is quite an interesting example because of the narrow interface of compilers (source input, code output, error output). Of course the programmer has to prepare/modify source programs with a text editor, but the interface between the text editor and the compiler is rather simple. A compiler is usually grammar-based and not metamodel-based, but this makes no major difference conceptually. Some compilers are directly generated from the grammar. But the simple grammar is obviously not sufficient for total generation of the tool. Finally, many different tools may use the same grammar. This generates an obvious interoperability opportunity by using the common grammar as a way to interoperate between different compilers for the same language.

Many additional tasks have been identified for the programmer and these could be supported by different tools. The notion of Integrated Development Environment (IDE) may be considered as a composite tool.

3.3 Illustrative projects

This section presents some illustrative projects on Tool interoperability solved by using a model-based tool approach. Lessons learnt when developing these examples are the focus of the next section.

UML to MS Project

One of the initial examples of model transformation was to manipulate UML activity diagrams with the Microsoft Project tool [19]. On the source side, the metamodel used was the relevant part of the official OMG UML metamodel. On the target side, one metamodel had to be defined for the MS Project tool. The input consisted in XMI serialized activity diagrams, while the XMI output of the transformation had to be

converted into the XML import format of MS Project. This additional transformation from MSP/XMI to MSP/XML is typical of the injectors/extractors used in such situations.

Most of the experiments we have done use unidirectional transformations between $Tool_A$ and $Tool_B$. However, once we have such a transformation, the inverse transformation from $Tool_A$ to $Tool_B$ is generally quite easy to implement. The metamodels can be reused as is, and usually the injectors/extractors and transformations are not complex to adapt manually. One of the authors has implemented the transformation from MS Project to UML activity diagrams in a couple of hours. This allows one to start the blueprint of a process model with MS Project, to tune it, and later if needed, to transform it into a UML activity diagram for insertion in a more stable project.

Version management

The particular case of making two versions of the same tool collaborate is important in theory and practice. Both tools work on similar domains and synchronizing them should not be difficult. However, in practice this is not always so simple. In an experiment [20], we considered the AutoSar industrial metamodel (about 5,000 elements), considering that this metamodel could be used by an industrial tool. The problem was to deal with legacy models conforming to the AutoSar 2.0 version and to use them in another tool working with the later defined AutoSar 2.x metamodel. The first step was to perform a matching operation to get a difference of the two metamodels. The difference is the set of additions, deletions and modifications to the first metamodel. This difference, expressed as a model could then be input to a higher order transformation which would generate another concrete transformation that will do the conversion from one version to the next one. This experiment shows how MDE can help making tools versions interoperate in a very regular pattern, when the metamodels of these tools are explicitly available. Of course, if the version difference between models is provided by the tools providers and has not to be computed, this is an important improvement.

Bugzilla to Mantis

The domain of software bug tracking or bug tracing tools is very interesting to study. Only as open source, there are nearly fifty tools that could be used for similar purposes. This is a real problem since many organizations have to use different tools. The experiment has been done in different phases. First metamodels of two typical tools (Bugzilla and Mantis) were defined. Then transformations between these tools were implemented. It then became clear that a pivot metamodel between these two metamodels could be most useful. Looking at the practical usage of bug tracking, we noticed that some companies were using ad-hoc tools like Excel to record and report bugs. We then added a bridge to/from this simple tool to Mantis/Bugzilla through the pivot metamodel. These experiments are reported in [15]. But one additional issue became more apparent: the lack of exact correspondences between some concepts of the different tool domains. The strict transformation approach is not able to provide

sufficient flexibility to handle such situations. In [20] we show how we can establish more abstract correspondences through model weaving to address these issues.

Other examples

Many other experiments have been conducted in model-driven tool interoperability. Bridges have been established with Office or Open Office tools, for handling general services. Graphical visualization has been implemented by bridges to SVG, to DOT/GraphViz, to GraphML/Prefuse, etc. Very often, when such a bridge is built, it is rapidly reused and sometimes improved. The tabular output was initially handled separately for Excel and HTML. Rapidly a pivot tabular metamodel was however built, allowing submitting tabular presentation data not only to the two aforementioned tools but to other ones as well. We can see the tabular pivot as the definition of an abstract data presentation virtual tool.

A more extensive one year study in an industrial context has been performed in the domain of BRMS (Business Rules Management Systems). The ILOG Rule language and similar tools have been successfully bridged with a number of metamodels, projectors and ATL transformations [21].

One example of tools that have been often targeted is constraint solving engines [22]. Long chains of transformations going from various problem statements to these engines have demonstrated the practicality of the approach, and the interest of using pivot metamodels to handle the different fine possibilities of these various types of solvers.

Some bridges between model transformation tools are also described in [3].

4 Discussion

4.1 Lessons learnt

Methodology for interoperability

One important conclusion that comes out of our preliminary work is the need to develop a methodology supporting tool interoperability. In presence of two tools that need to interoperate, we suggest first to define the data metamodel of each tool, then to build injectors/extractors for the tools, and finally to write the transformations. Each of these steps may be quite complex, and space does not allow for a detailed description in this paper. For the first step, for example, the metamodel may be manually defined from the tool user manual[1]. If there is an XML import/export scheme, this may be used to guide semi-automatically the building of the metamodel. In some cases the tool may use a concrete textual syntax and the metamodel can be

[1] We have successfully experimented “pair modeling” here, a practice similar to “pair programming”.

built from the grammar of the language. It is also quite frequent that the tool is based on some DB layer like MySQL and the corresponding internal data schema may be used as a first sketch of the metamodel. Sometimes several solutions may be followed and different metamodels may have to be aligned to produce a suitable solution.

Flexible modeling

As discussed in the FlexTools workshops [23], a common problem is interfacing formal modeling tools and more informal but flexible free-form approaches. Practitioners throughout the software lifecycle are currently forced to choose between them. Whichever they choose, they lose the advantages of the other, with possible frustration, loss of productivity and sometimes of traceability and even quality. In the list of ATL model transformation use cases [5], several examples fall in this category. Therefore, we believe our model-based strategy can help in this matter.

Tool services interoperability

Metamodel based tool interoperability can be used in many different situations to build conversion bridges. The first situation is when we have to link two versions of the same tool (AutoSar example). The second situation is when we have two different tools within the same application domain (Bugzilla/Mantis example). But a very important number of cases concern tools that are not in the same domain, and that just provide general service reusability. Two examples of services are data entry and data display. A tool like Excel, for example, may be used for tabular data entry and tabular data display. When we need to have graph visualization (a common practice in MDE), we may use tools like DOT or GraphViz. According to the type of visualization needed, a wide spectrum of services may be implemented by open source or proprietary tools.

Global tool maps

Ideally, in a given context, it should be possible to build a relationship map with all the available tools and all the conversion paths between these tools. The brainstorming list given in section **Erreur ! Source du renvoi introuvable.** may give an idea of the basis of such a list. We may see a future facility as showing the various possible connections between all the tools used in a given group, and managing the conversions between them. Note that we may get indirect conversion paths between tools by transitivity through a pivot intermediate tool.

4.2 Related work

Tool interoperability has been a subject of research for a long time. The dream of being able to plug together tools to achieve interoperability between them has motivated many efforts, based on the fact that these tools have not initially been designed to cooperate. The recognition that the number of tools was high led many investigations to consider a hub or more often a bus to avoid $O(N^2)$ point to point

connections between N separate tools. PCTE [7] is an ECMA standard framework for software tools developed in the Esprit programme and defining the way they may access a common hub. Another well known effort is the AD/Cycle lead by IBM in the late 80's [14]. These efforts are characteristic of the tool-bus vision. The CDIF (Case Data Interchange Format [2]) initiative was also very successful in the 90's and many of these ideas were later reused in OMG MOF-based standards. Many efforts in defining graph-based interchange formats for various tools may also be mentioned [9]. The work on format interchange is obviously complemented by the work on semantic interoperability [16].

The notion of virtual tool has been presented in [4]. This work is related and complementary to the present one. The issue addressed in this proposal is to build a virtual tool from a set of concrete tools. The goal is achieved by precise operations on the metamodels associated to the different tools.

4.3 Further work

In the 80's, one main argument to support object technology was reusability. Nowadays different other ways to look for reusability are being explored. By opposition to implementation reusability offered by object technology, service technology is providing some form of functionality reusability. This is also one objective of this work. We do not have usually access to the implementation of a tool, but we can have access to the services it exposes, and we can reuse these services, possibly by combining several of them. Our conviction is that to build this reusability on a broad scale, we need to use precise metamodels.

We are planning to develop new experiments in tool interoperability in order to gain understanding on the possibilities and challenges.

Many ideas concerning tool interoperability equally apply to more general situations of system interoperability. For example enterprise interoperability is currently a field were similar techniques could be developed.

Only some characteristics of model driven tool integration have been presented in this position paper. The state of the art in tool integration has yet to be expanded. Previous achievements of bus-based and service-based tool integration have to be more completely analyzed. The additional advantages brought by MDE should also be more thoroughly assessed. Among the key techniques used to achieve tool interoperability, one finds various forms of metamodel alignment. The conditions for this need to be investigated and suggestions made for the proposed approach to scale up.

We have mainly focused this paper on loosely coupled tools, with the help of data metamodels. This study needs to be completed by the study of state and event metamodels for more tightly coupled tools.

One important issue that has not been mentioned here is the need for a practical support to transformation chaining. As has been seen in several occasions, the interchange between two tools may use several intermediary or pivot metamodels.

This in turn may need some support for transformation typing, but these issues go much beyond the scope of the current paper.

5 Conclusion

We have presented here some pragmatic issues about the possibility to use MDE in tool interoperability situations. We all agree on the importance of tools in the software development landscape. However, we do not yet understand all the characteristics of a tool. Until now tool interconnection has been traditionally handled within the classical “tool bus” vision. The “service vision” may help improving these solutions. The arguments developed in this paper in favor of the “metamodel vision” may still seem a bit premature, but we have checked their relevance and potential on a number of cases. More importantly, this approach seems economically feasible if there are a number of open source libraries of modeling artifacts (metamodels, transformations, projectors) that may be easily reused. The current AtlanMod zoos [5] show how this could be implemented in practice.

6 Acknowledgments

This work has been supported by the IDM++, Lambda and CESAR projects that all contributed different solutions to the tool interoperability experiments. We acknowledge the help of all past and present members of the AtlanMod team. We acknowledge also many discussions with Antonio Vallecillo. The ideas presented here are similar to his "village metaphor" proposal [17].

References

[1] Bézivin, J. *On the Unification Power of Models*. Software and System Modeling (SoSym) 4(2):171-188, Springer Journals, (2005)

[2] Bézivin, J., Ernst, J. & Pidcock, W. *Model Engineering with CDIF* OOPSLA'98, Vancouver, post-proceedings, Summary of the workshop, (October 1998)

[3] Jouault, F., Kurtev, I. *On the Architectural Alignment of ATL and QVT*, ACM Symposium on Applied Computing (SAC 06), Model Transformation Track, Dijon, Bourgogne, France, (2006)

[4] Jouault, F., Guéguen,T. *Integration by Model-driven Virtual Tools*, ECMDA, Oslo, (2009)

[5] AtlanMod *AtlanMod MegaModel Manager* (AM3). http://www.eclipse.org/gmt/am3/ and the zoos, repositories of open source metamodels.

[6] Didonet Del Fabro, M., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.: *AMW: a Generic Model Weaver* IDM05, (2005)

[7] Didonet Del Fabro, M., Bézivin, J., Jouault, F., Valduriez, P.: *Applying generic model management to data mapping*. In: Proceedings of the Journées Bases de Données Avancées, BDA05 (2005)

[8] Long, F., Morris, E. *An Overview of PCTE: A Basis for a Portable Common Tool Environment*. Technical report CMU/SEI-93-TR-1, (March 1993)

[9] Elliott Sim, S., Koschke, R. *ICSE Workshop on Standard Exchange Format (WoSEF)***,** ACM Sigsoft Software Engineering Notes, V.26, January 2001, pp. 44-49

[10] Sjöstedt, C.J., Shi, J., Törngren, M., Servat, D., Chen, D., Ahlsten, V., Lönn, H.: *Mapping Simulink to UML in the design of embedded systems: Investigating scenarios and transformations***.** OMER4 Workshop: 4th Workshop on Object-oriented Modeling of Embedded Real-Time Systems. (2007)

[11] Institute of Electrical and Electronics Engineers. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. New York, NY: 1990.

[12] Kuhn, D.L. *Selecting and effectively using a computer aided software engineering* **tool**. Annual Westinghouse computer symposium; 6-7 Nov 1989; Pittsburgh, PA (USA); DOE Project.

[13] K. Robinson (1992). *Putting the Software Engineering into CASE***.** New York: John Wiley and Sons Inc.

[14] IBM *AD/Cycle strategy and architecture*, IBM Systems Journal, Vol 29, NO 2, 1990; page 172

[15] Bézivin, J, Brunelière, H, Jouault, F, and Kurtev, I (2005). *Model Engineering Support for Tool Interoperability* In: Proceedings of the 4th Workshop in Software Model Engineering (WiSME 2005), Montego Bay, Jamaica.

[16] Heiler, S. *Semantic interoperability*. ACM Comput. Surv. 27, 2 (Jun. 1995), 271-273.

[17] Vallecillo, A. *A Journey through the Secret Life of Models*. Dagstuhl Seminar on Model Engineering of Complex Systems (MECS), Germany, Aug. 2008. ISSN 1862-4405. http://drops.dagstuhl.de/opus/volltexte/2008/1601

[18] Didonet del Fabro, M., Bezivin, J., Valduriez, P. *Model-driven Tool Interoperabilituy: An Application in Bug Tracking*, ODBASE 2006 international conference, LNCS V.4275, (2006), pp. 863-881.

[19] Bezivin, J. , Breton, E. *Applying The Basic Principles of Model Engineering to The Field of Process Engineering* Upgrade, Vol. V, N.5, October 2004 http://www.upgrade-cepis.org/issues/2004/5/up5-5Bezivin.pdf

[20] Vara, J.M., Didonet Del Fabro,M., Jouault, F., Bézivin, J. *Model Weaving Support for Migrating Software Artifacts from AUTOSAR 2.0 to AUTOSAR 2.x*. 4th European Congress on Embedded Real Time Software (ERTS 2008), Toulouse, France

[21] Didonet Del Fabro,M., Albert, P., Bézivin, J., Jouault, F. *Industrial-strenght Rule Interoperability using Model Driven Engineering* INRIA research report, RR-6747, (2008)

[22] Chenouard, R. Granvillers, L. Soto, R. *Rewriting Constraint Models with Metamodels* Proc. 8th Symposium on Abstraction, Reformulation and Approximation, SARA2009, USA, (July 2009)

[23] Flexitools2010 ICSE Workshop on Flexible Modeling Tools, http://www.ics.uci.edu/~tproenca/icse2010/flexitools/