Softw Syst Model (2012) 11:7–10
DOI 10.1007/s10270-010-0184-6

GUEST EDITORIAL

Editorial for the theme issue on model-based interoperability

Tony Clark · Jorn Bettin

Published online: 4 February 2011
© Springer-Verlag 2011

1 Introduction

The commercial benefits claimed for software based on visual and graphical modeling languages are well documented. Many domain specific modeling tools exist and are being used as point solutions. Tailoring of notations to the specific application domain and combined use of several languages define the nature of the approach, and constitute the source of the achievable benefits. Unfortunately, data representations and the mechanisms used to integrate modeling languages tend to be highly tool specific. This compromises the use of modeling languages in building tool chains that may contain components from several suppliers.

The main motivation for the use of domain-specific models is the desire to express problems in a compact form that reflects the natural terminology of human domain experts, and that is easily accessible to software tools. In short, domain-specific models are raising the level of abstraction of software specifications and of knowledge representation in general. When models are used to formalize the results of domain analysis, the outcome is a clean separation of concerns in the problem space. This is a major advance over aspect oriented programming, where separation of concerns is only achieved in the solution space. The value of a model increases with the intuitiveness of the concrete syntax. Graphical elements may be needed to increase usability, and often such languages are referred to as domain specific modelling languages (DSML).

To extend the reach of DSML-based approaches, significant improvements in the level of interoperability between DSML tools are required. Software development has become highly decentralized, and an assumption that all parties in a global software supply chain will use identical tooling is simply not realistic. Today's software supply chains are much less automated than supply chains in other, more mature industries.

For example, the typical outsourcing relationship between two software manufacturers is based on informal specifications. Either the tooling used by the outsourcer is unknown to the customer, or if the customer is more discerning, the outsourcer sets up an environment that matches the one of the customer, usually at a substantial cost (licenses, staff training, etc.). The inefficiencies are such that it is not uncommon to set up mixed teams with staff from both organisations, and to rely heavily on extensive travel and face-to-face collaboration to achieve the desired outcome.

The challenge lies in identifying the ingredients needed to create highly automated software supply chains that minimise the effort to integrate new suppliers of specialized software artefacts. Meeting this challenge requires a significant increase in the use of formal (yet highly compact) specifications, and advanced tooling for creating, managing, and exploiting formal specifications to the fullest.

Ideally, supply chain participants would be able to exchange formal language definitions and models, and would rely on a shared implementation of basic services for managing such model-based artefacts.

T. Clark (✉)
School of Engineering and Information Sciences,
Middlesex University, The Burroughs, Hendon,
London NW4 4BT, UK
e-mail: t.n.clark@mdx.ac.uk

J. Bettin
Sofismo AG, Saegestrasse 50, 5600 Lenzburg, Switzerland
e-mail: jbe@sofismo.ch

2 History

Technologies for generating domain specific language (DSL) based tools and tool-chains date back to *meta* Computer Aided Software Engineering tools in the 1980s (even earlier if textual DSLs are taken into account e.g. Lisp). These tools are representative of a class of tools that allow the user to construct meta-models of languages and development processes. The models are then instantiated to produce tools that support the language and the process in terms of editors, code generators and wizards. Many of the meta-tools that support this technology suffer from bespoke data representations which makes interoperability via third party developers difficult. Such tools are usually capable of participating in tool-chains, however there is currently little guidance on how the tools can be adapted in order to fit in.

The Unified Modeling Language (UML) was developed to address the issues of non-standard model representation. The UML definition is written with respect to a meta-language MOF that aims to provide a basis for standard model interchange and for standard language extensions. There are several reasons why UML can be argued as unsuitable as a basis for DSL tool-chains. Firstly, its relationship with MOF is weak: UML is an instance of MOF, but not an extension of it; therefore MOF–UML does not implement a so-called *golden braid*. UML extensions are not written using meta-language extensions, they are written using a restricted form of meta-access via profiles and stereotypes. Secondly, UML aims to be a general purpose modelling language which means that it provides a large array of general purpose features, leading to a comprehensive language for expressing application domains rather than languages. Thirdly, the UML interchange technology, XMI, is generally considered to be difficult to implement completely. Few (if any) UML tools achieve complete interoperability.

3 The future of software system interoperability

Imagine a world where software systems are created by composing models and tools that operate on these models. Each tool covers a specific aspect of the problem space or the solution space. Each tool is a component that may have originated from a different supplier, some open-source and some proprietary.

So there are tool developers and there are tool-chain compositors. There is a requirement on developers to provide sufficient information for the compositors to be able to their job. This amounts to the compositor being able to tell what the tool does and what data it supplies and consumes.

Imagine a tool framework that supports tool interoperability and the construction of tool-chains. In order to specify such a framework it must be possible to manipulate tools as first class citizens and to access the key tool-features that contribute to the construction of a tool-chain.

The key to interoperability is that data and interface descriptions are in a format that other tools can use. This is achievable through the use of a tool description that provide three components:

1. Data models (meta-models) describe the externally visible data provided and consumed by the tool.
2. Interfaces describe the functionality offered by the tool.
3. A model of the tool behaviour.

The use of behaviour models is a key feature in achieving interoperability. The behaviour models may arise in different ways. For example, an open-source reference implementation of a new category of tool is an attractive way of achieving consensus. The open nature of the process removes technical obstacles imposed by proprietary interests and is likely to attract the best development talent. In addition, making the process open fosters a Darwinian approach that makes the selection of the best technical solutions more likely.

An open-source behavioural model may be used as-is or may just define sufficient behaviour to standardize other implementations. Behavioural models do not restrict other implementations from extending the behaviour and thereby adding proprietary commercial value to the tool, however the additional functionality will achieve lower levels of interoperability.

Given tool models including behavioural descriptions, it is possible to develop a notion of tool equivalence. There are various levels of equivalence from very weak to very strong. For example, equivalence based on interfaces tends to be quite weak, data equivalence is much stronger and finally behavioural equivalence is the strongest. Whilst in theory behavioural equivalence is undecidable, it is practical in terms of a collection of specific test-cases that is incrementally built-up over time by the tool user community.

An equivalence based on data or behaviour requires a common representation of the information in tools. In turn, this requires a common language to express the equivalence rules, i.e. a common meta-language—a language for describing languages.

A number of candidate meta-languages exist, including MOF from the Object Management Group, Ecore—the meta-model for the EMF language, as well as XMF-Mosaic and Gmodel—simple meta-languages that implement a golden braid architecture.

A key differentiator between meta-languages such as MOF or Ecore and meta-languages based on a golden braid architecture is the ability of the latter to provide a recursively defined *universal model artefact* concept, which is available

for use and possible extension in all artefacts expressed in the meta-language.

4 Current state of the art and the challenges

Today, the most widely meta-language is the XML schema language, which provides a means for defining the structure, content and semantics of XML documents. However, schema conformance of an XML document is often not enough for useful interoperability. The XMI standard for exchange of UML model data is good example of the limitations and typical problems.

In the same way that XML schemas are formal language definitions, relational database table definitions are formal language definitions. Hence the title of second most popular meta-language arguably belongs to SQL DDL. By providing the rules for verifying the well-formedness of textual program source code, even programming language definitions can be interpreted as meta-languages. Program source code in turn may consist of object oriented class definitions that describe the structure and semantics of instances pertaining to a particular application. Given that SQL provides no mechanism for interoperability across SQL schema boundaries, and given that each programming language relies on idiosyncratic tooling, achieving interoperability at the level of traditional source code remains to be a challenge.

MOF is a simple meta-language used to define UML and other languages. Therefore, MOF is a suitable candidate as a basis for tool interoperability. However, MOF tends to be instantiated to produce languages (e.g. UML); any semantic definitions in MOF are weakly connected to models expressed in modelling languages defined with MOF.

The Eclipse Modelling Project provides a collection of open-source technologies that can be used to develop modelling tools. These include EMF and GMF. EMF is a technology that can be used to develop data definitions against a common meta-language called Ecore. Ecore provides reflection and can be used to raise events when Ecore-based data changes. The original aim of Ecore was to provide a basis for Java tool chains via data interchange and listening for data change events. EMF is a collection of Ecore-based tools for simple editor generation. GMF is a collection of EMF-based tools that supports graphical editor generation.

Ecore is a good candidate as a basis for tool interoperability. Its strength is that it has a clear pragmatic semantics: its implementation in Java. Its weakness is that some of the features are based on implementation concerns and it is incomplete. Both EMF and GMF are very useful and are good examples of the kinds of technologies required when processing meta-models; however, they are only part of the solution.

The Visualization and Modeling SDK (DSL Tools) initiative at Microsoft aims to allow tool-chains to be developed using Visual Studio. The DSLs are mixtures of graphical and textual languages developed using a bespoke meta-language and specifically aimed at generating code into Microsoft solution architectures. Like the Eclipse/EMF/GMF technology described above, this is one part of the tool-chain solution. Unfortunately, the languages and technologies are bespoke and therefore not an open solution for building tool-chains. However, Microsoft has emphasized its commitment to interoperability and has recently engaged with the Object Management Group stating that it hopes to influence meta-technologies.

Many existing tools for DSMLs are closed in the sense that it is difficult to access and manipulate the languages and models in ways that were unforeseen by the tool developers. A tool-centric view of languages does not readily support interoperability since the use-cases for languages become fixed and difficult to extend. Interoperability would be facilitated by taking a more independent user-centric view of languages where each tool contributes part of the language semantics. In this way, no tool could claim exclusive ownership of a language and would be forced to make allowances for other use-cases and extensions to its own use-case via interoperability.

In conclusion, many technologies exist that support working with DSMLs. Some have bespoke notations and others are standard but are suboptimal for a variety of reasons. Lessons can be learned from looking at the process of standardisation from a wider angle, and from considering the practical realities of lock-in as experienced by software users.

We believe that the time is right to consolidate the experiences of the last 10 years in terms of modelling, tool-chains and DSML development. In 2009, the KISS initiative made the first steps by reaching a consensus on fundamental values and principles for designing and using domain-specific languages. The big remaining challenge relates to the second objective of the KISS initiative: *progress towards interoperability between tools*; creating open technologies that interpret the agreed principles in a way that enables interoperability between different DSML tool ecosystems.

5 In this issue

The previous sections have established the need for model-based interoperability, in terms of standard notations such as UML, DSMLs, and tools to support these technologies. Interoperability poses a huge challenge, but is essential to the use of heterogeneous model driven approaches to the design and development of industrial scale systems. This issue of SoSym contains a collection of papers that address aspects of the interoperability challenge.

Interoperability requires a technology infrastructure that allows multiple concurrent access using a variety of client platforms. One solution to this issue is the use of *cloud computing* and in *Supporting the Internet-Based Evaluation of Research Software with Cloud Infrastructure* Pieter Van Gorp and Paul Frefen discuss a cloud-based case study that is motivated by reviewing software modelling research submissions.

Interoperability requires multiple views of a system to be synchronized. There are several ways to achieve this including change propagation and model transformation. The article *A model-driven approach to automate the propagation of changes among Architecture Description Languages* by Romina Eramo, Ivano Malavolta, Henry Muccini, Patrizio Pelliccione, and Alfonso Pierantonio describes a system that has been implemented to support interoperability and synchronization in the ADL domain based on the DUALLy, ATL, AMW, TCS, and ASP frameworks in Eclipse.

A key feature of model driven development is the construction of model transformation chains (MTCs) that take high-level descriptions of systems and produce low-level implementations as target platform code. MTCs for real-world systems become large and complex, therefore it is attractive to decompose them into smaller chains. However, as described in *Realizing Model Transformation Chain Interoperability* by Andres Yie, Rubby Casallas, Dirk Deridder and Dennis Wagelaar, interoperability problems arise when large MTCs are composed from smaller units. This article describes the problems and proposes a solution based on representation and resolution of correspondences between MTCs.

Interoperability issues can arise at all levels of system descriptions and modelling can help at each level. Execution traces are an example of low-level run-time system information where interoperability issues arise due to the multiple tools available for trace analysis. The article *A Metamodel for the Compact but Lossless Exchange of Execution Traces* by Abdelwahab Hamou-Lhadj and Timothy C. Lethbridge, addresses these interopability issues by proposing a single *Compact Trace Format* that is universal and solves scalability issues using a graph-based approach.

Industrial scale models tend to be very large and pose interesting challenges for model-based technologies such as interopability. This issue is addressed by the article *Model Interoperability in Building Information Modelling* by Jim Steel, Robin Drogemuller and Bianca Toth, where the authors have direct experience of handling semantically rich three-dimensional models of buildings where collaboration is a vital part of the business. The authors reflect on their experiences building tools to process such models and outline the challenges that need to be addressed.

Model transformations tend to be developed for a use with a single source meta-model. However, often meta-models are very similar, for example there are a huge number of class-association based meta-models. The article *Reusable Model Transformations* by Sagar Sen, Naouel Moha, Vincent Mahe, Olivier Barais, Benoit Baudry and Jean-Marc Jezequel addresses the issue of allowing a single model transfromation to interoperate across a family of similar meta-models by identifying the effective input meta-model of a transformation and providing a mechanism to mark-up similar meta-models so that the transformation can apply to them.

Acknowledgments The editors would like to thank Martin Schindler and Geri Georg for their help in putting this theme issue together. We are indebted to the reviewers for their timely efforts and their comments which led to improvements in many of the articles. We would also like to thank the authors for submitting their work and for addressing comments and suggestions.

6 About the editors

Tony Clark is a Professor and Head of Department of Business Information Systems at Middlesex University, London. Tony's research background is in programming language design and development, and in software modeling. He worked as a research scientist with Marconi Ltd. where he developed many Lisp-based systems for AI applications. After becoming a lecturer, he worked on executable versions of meta-languages as a basis for UML and contributed to the OMG UML 2.0 revision. This latter work led to the development of the XMF-Mosaic meta-modelling toolkit and associated company that Tony co-founded in 2003. Tony has worked as a consultant with many companies and is the co-author of *Applied Metamodelling: A Foundation for Language Driven Development*.

Jorn Bettin is a co-founder of Sofismo in Switzerland and of SoftMetaWare in New Zealand. He leads the development of the Gmodel meta-language and repository technology, and has co-authored three books that cover the technology, engineering, and management aspects of domain engineering. Jorn is convinced that model oriented domain analysis techniques and domain-specific models are the only viable mechanism to capture deep domain expertise in a form that is accessible to future generations of software professionals and software tools. Jorn has worked in methodology leadership roles in an IBM product development lab, initiated the Eclipse Generative Modeling Tools project, and—back in 1994/5—led the development of LANSA /RUOM, a widely used model driven CASE tool for the IBM iSeries platform.