Towards Next Generation Metamodeling Tools*

Kirk Schloegel, David Oglesby, Eric Engstrom

Aerospace Electronic Systems Research Lab, Honeywell International
3660 Technology Drive, Minneapolis, MN 55418
{kirk.schloegel, david.oglesby, eric.engstrom}@honeywell.com

Abstract

In this paper, we discuss four areas in which next-generation metamodeling tools have the potential for improvement. These include (i) support for multi-domain system modeling, (ii) support for commercial off-the-shelf and homegrown tool integration, (iii) increased code generation and analyses, and (iv) increased model reuse through the use of design patterns as first-class modeling entities.

1. Introduction

Over the last decade, a great deal of focus has gone into investigating and developing metamodeling tools. Such tools provide graphical modeling capabilities for the design of metamodels that in turn define graphical modeling notations. As such, metamodeling tools are domain-specific visual language (DSVL) tools whose modeling domain is DSVLs themselves. In addition, metamodeling tools compile or interpret metamodels to result in design tools for their specified DSVL.

Such derived modeling tools have a number of advantages over traditional modeling tools. Since domain-specific knowledge (represented as metamodels) is separate from generic tool and user-interface knowledge, (i) the creation of graphical modeling tools requires only domain experts and not tool creation experts; (ii) modeling tools can be developed in little time; and (iii) they are highly customizable and flexible. Metamodeling tools have shown great ability to lower the cost of developing DSVL tools. As such, they have been instrumental in increasing the use of DSVLs across diverse domains, and especially in niche domains where small user bases have previously made special-purpose tools prohibitively expensive.

Metamodeling tools not only allow the syntax of a modeling domain to be defined, but also allow semantics to be defined. Doing so increases the usefulness of models, as semantics can automatically be interpreted to support various types of modeling services. Examples of such services include (i) the generation of code and other types of textual artifacts from models, (ii) the plug-in of domain-specific analysis routines, and (iii) the implementation of other domain-specific modeling services (e.g., constraint checking). This functionality has helped to improve both design productivity and product quality.

* This material is based upon work supported by the United States Air Force under Contract No. F33615-00-C-1705. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force.

2. Next generation metamodeling

Despite these successes, metamodeling tools can be improved in a number of ways. In this position paper, we discuss four areas in which next-generation metamodeling tools have the potential to make significant gains. These are (i) multi-domain system modeling, (ii) commercial off-the-shelf (COTS) and homegrown tool integration, (iii) increased code generation and analyses, and (iv) increased model reuse through the use of design patterns as first-class modeling entities.

A. Multi-domain modeling

The motivation behind DSVLs is the use of languages and modeling notations that are simpler but more expressive, narrower in focus but deeper in expressive power than general-purpose languages. While the ratio of expressive power to linguistic simplicity makes DSVLs promising for describing a single domain deeply, such a narrow focus is not sufficient for the increasingly complex systems being designed today. There are two possible solutions to this problem. Either DSVLs can be made broader in their expressiveness or else complex systems can be described using a number of interacting modeling notations in which some of the concepts from different notations interact. In general, we recommend the latter solution that adheres to the principals of modularity and separation of concerns.

For example, embedded systems can be modeled as (at least) two interacting domain models: one describing hardware and the other describing software. The hardware and software concepts of processors, processes, and threads interact. Embedded system design can further benefit by allowing users to describe additional non-functional aspects such as quality-of-service, middleware, and I/O [1][9] using appropriate modeling notations. As another example, linking concepts from Unified Modeling Language (UML) [3] class diagrams and data flow diagrams [2] provides a mechanism from which more complete code can be generated than either type of modeling notation can easily provide alone.

While a key characteristic of metamodeling tools is that they can be used to quickly develop numerous derived DSVL tools, current metamodeling tools have little or no support for interacting DSVL tools. In order to work with multiple models from different, but interacting domains at one time, it is typically required that a DSVL tool be derived from a single metamodel in which the multiple metamodels of interest are encapsulated. In the embedded system example above, an “embedded” modeling domain could be defined by composing the hardware and software metamodels into a single encompassing “embedded” metamodel and then specifying additional relationships and constraints between the interacting meta-entities. However, this approach discourages metamodel reuse. Instead, we propose that next generation metamodeling tools support the linking together of multiple *independent* metamodels in a flexible and customizable manner.

The MILAN [1] and SAGE [9] projects have both developed embedded systems design environments built upon metamodeling tools. (MILAN is built upon Vanderbilt’s Generic Modeling Environment (GME) [12] and SAGE is built upon Honeywell’s Domain Modeling Environment (DOME) [4].) While these environments utilize metamodeling technology to link modeling domains, they do so in a non-customizable way. Next generation metamodeling tools should generalize such approaches by providing support for user-definable cross-domain interactions.

One mechanism that has been proposed to provide this support is to create a linkage modeling domain [25]. Linkage models explicitly specify relationships between entities that exist in different metamodels. Essentially, proxies to these *meta-entities* are instantiated in a linkage model and then various types of relationships (e.g., equivalence, inheritance, composition, etc.) are specified among them. The tool environment can interpret specific types of relationships automatically and users can extend these by defining and implementing new types of relationships. We have implemented such a linkage DSVL using DOME. This DSVL is specified by a stand-alone metamodel. Top-level system models may contain pointers to one or more linkage models. Any cross-domain relationships that are specified are automatically supported across DSVL tools.

B. COTS Tool Integration

While significant, the increase in popularity of metamodeling tools has paled in comparison with that of COTS design tools such as Rational Rose® [23] and MATLAB® and Simulink® [16]. These tools have captured large industrial projects, while metamodeling tools have been primarily relegated to niche domains.

Since most COTS tools contain little or no domain-specific semantics, they often provide only a partial design solution (e.g., incomplete coverage in code generation). The remainder requires additional domain-specific or cross-domain information and semantics. Furthermore, we would argue that each incremental increase in a tool's usefulness drops its applicability to a certain market segment. Since it is in the vendors' best interests for their tools to be applicable to the widest possible market, it is unlikely that the remainder will ever be made up. And so, manual effort or at least the development of special-purpose tools is usually required. Another disadvantage of COTS tools is that they can be notoriously difficult to integrate and maintain in an integrated environment [5].

However, while they suffer in comparison with DSVLs in many respects, COTS tools are still useful and they represent major user bases and increasing amounts of legacy models. In fact, the popularity of COTS tools (along with effective marketing) has often led users to downplay the significance of cross-domain interactions. Instead, users often regard COTS tool suites as complete solutions (even though they do not generate complete code). COTS tools are well supported and well tuned to common software design approaches (e.g., OO and data flow). They represent de facto industry standards. They typically support modeling within multiple interacting domains (e.g., UML class diagrams interacting with collaboration and state diagrams), although in rigid and non-customizable ways. They provide useful modeling service functionality as well as automatic code generation and certain types of model analyses.

The conclusion is that metamodeling tools should be able to plug into COTS tools effectively. This integration should be nearly transparent to the user. Metamodeling tools should be able to import (and export) data from (and to) external tools easily, even as file formats evolve.

While the realization of such functionality poses real challenges, recent efforts in tool integration via metamodeling tools [11][14][21] have shown promising results. In [11], a two-step approach is described in which first a syntactic mapping of modeling notations is performed followed by a semantic mapping. The authors argue that such an approach provides a separation of concerns that simplifies the tool integration problem. The authors of

[14] describe efficient model transformation algorithms in support of tool integration. In [21], a high-level model transformation language is described that maps down to XSLT [27]. Here, textual metamodels are specified using XML Document Type Definitions (DTDs) [26]. IBM AlphaWorks' XSLerator [28] tool can also generate XSLT scripts from mappings that are defined using a visual interface.

In related work, the development of the Meta-object Facility (MOF) [18] and XML Metadata Interchange (XMI) [19] formats provide standards for metamodel specifications. Such standards are crucial for the transparent integration of models between metamodeling tools and should be the base upon which future integration support is built.

C. Increased Code Generation

Code generation within modeling domains has seen a tremendous improvement in recent years. However, in order to generate a complete system, code generation across DSVLs is required. That is, single-domain code generation is typically not sufficient to provide high levels of coverage for the complex applications of today. Instead, increased code generation requires at least interaction, and often cooperation, between different modeling domains and/or tools.

To this point, little general metamodeling support has been developed for cross-domain code generation. The result is that many point solutions have been developed. These typically consist of hand-written, stand-alone programs that read in data from models of different tools (e.g., through a COM interface) and combine and format the data appropriately for output. Such techniques are exceedingly slow and painful to develop, even harder to maintain, extend, or verify, and vulnerable to changes in tool interfaces. As such, cross-domain code generation would benefit from a more flexible and customizable approach. Since cross-domain code generation requires cross-domain modeling support, this remains an open area of research.

D. Model Reuse

Support for reuse is an important feature for any model-based development tool. Metamodeling tools also need to support reuse, and on three distinct levels: (i) metamodel reuse, (ii) model reuse, and (iii) modeling entity reuse. One dimension of metamodel reuse involves being able to customize metamodels while still supporting legacy models. The authors of [13] describe a metamodel-inheritance approach to reuse for this purpose. Another dimension of metamodel reuse is the ability of multiple DSVL tools that are derived from independent metamodels to interact. The importance of this issue has been discussed above. The second level is model reuse [22]. A number of tools, such as MetaEdit+® [17] and GME, provide good support for model reuse. The third level of reuse is model entity reuse. This includes reuse (i) within a single model (i.e., copy-and-paste), (ii) across models, and (iii) across modeling domains. The former two are typically well supported. However, current metamodeling tools do not provide adequate support for reuse of model entities across domains. Support for cross-domain reuse is extremely useful for cross-domain modeling.

The realization of design patterns [7] as first-class graphical modeling entities holds the potential of greatly increasing reuse on the levels of model and modeling entity. However,

the variant and cross-domain nature of design patterns has led to difficulties in modeling and interpreting them in graphical modeling notations. Various approaches to capturing patterns for reuse have been investigated in [6][8][10][15][24]. However, all of these approaches have been tied to a single modeling domain. We believe that widespread use of design patterns as first-class modeling entities requires extensive cross-domain support. This is one of the reasons that new design patterns are typically described both with models and text. When a single notation (usually UML) is insufficient to describe them in adequate detail, text is used to compensate.

We have developed *archetypes* [20] to support the modeling and reuse of design patterns. Archetypes have mechanisms that specifically address the variant and cross-aspect nature of design patterns. They can have multiple *implementations* that can cut across modeling domains, *portals* through which information can flow between modeling domains, and implementation- or archetype-specific *operations* to accommodate their interpretation from within different domains. Archetypes insulate developers from their implementations and can interact with native modeling entities according to the syntax and semantics of the domain in which they are applied. They support reuse through a user-accessible *shelf* of previously designed archetypes as well as powerful composition capabilities. For example, archetype implementations can contain archetype instances. In this way, higher-level design patterns can be modeled.

3. Conclusions

We have described four areas in which next generation metamodeling tools can be improved to support the increasing complexity of current systems. The common thread among these is metamodeling support for cross-domain interactions, regardless of whether these occur between DSVLs derived by different metamodels or between derived DSVLs and external tools. It is our opinion that this key enabling technology is an important area for future work.

4. References

[1] A. Bakshi, V.K. Prasanna, and A. Ledeczi. *MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems*, In Proc. of Workshop on Languages, Compilers, and Tools for Embedded Systems, 2001.

[2] L. Bichler, A. Radermacher, and A. Schurr. *Combining Data Flow Equations with UML/Realtime.* In Proc. 4th Int. Symp. on Object-Oriented Real-Time Distributed Computing, pages 403-410, 2001.

[3] G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[4] DOME is an open source research project and is available from http://www.htc.honeywell.com/dome.

[5] A. Egyed and R. Balzer. *Unfriendly COTS Integration – Instrumentation and Interfaces for Improved Plugability.* In Proc. of 16th IEEE Conference on Automated Software Engineering (ASE2001), November 2001.

[6] G. Florijn, M. Meijers, and P. van Winsen. *Tool Support for Object-oriented Patterns.* Lecture Notes in Computer Science, Vol. 1241, 1997.

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[8] J. Hannemann and G. Kiczales. *Design Pattern Implementation in Java and AspectJ.* In Proc. of OOPSLA 20002, 20002.

[9] Honeywell International, *The Systems and Applications Genesis Environment (SAGE)*, htttp://www.honeywell.com/SAGE.

[10] G. Karsai. *Tool Support for Design Patterns.* NDIST 4 Workshop, 2001.

[11] G. Karsai and J. Gray. *Component Generation Technology for Semantic Tool Integration.* Proc. of the IEEE Aerospace 2000, CD-Rom Reference 10.0303. 2000.

[12] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai. *Composing Domain-specific Design Environments.* Computer, pages 44-51, November 2001.

[13] A. Ledeczi, G. Nordstrom, G. Karsai, P. Volgyesi, and M. Maroti. *On Metamodel Composition.* In Proc. of IEEE CCA 2001, 2001.

[14] T. Levendovszky, G. Karsai, M. Maroti, A. Ledeczi, and H. Charaf. *Model Reuse with Metamodel-based Transformations.* In Proc. of 7th International Conference on Software Reuse, 2002.

[15] D. Maplesden, J. Hosking, and J. Grundy. *Design Pattern Modelling and Instantiation using DPML*. In Proc. of the Tools Pacific 2002, February 2002.

[16] The MathWorks, Inc. *MATLAB User Guide*. Natick, MA 01760-1500, 1998.

[17] MetaCASE Consulting. *MetaEdit+, Version 3.0 User Guide*, 2000.

[18] Object Management Group (OMG). *Meta-object Facility (MOF), Version 1.4.* http://www.omg.org/technology/documents/formal/mof.htm.

[19] Object Management Group (OMG). *XML Metadata Interchange (XMI), Version 1.2.* http://www.omg.org/technology/documents/formal/xmi.htm.

[20] D. Oglesby, K. Schloegel, D. Bhatt, and E. Engstrom. *A Pattern-based Framework to Address Abstraction, Reuse, and Cross-domain Aspects in Domain Specific Visual Languages*. In Proc. of OOPSLA 2001, 2001.

[21] M. Peltier, F. Ziserman, J. Bezivin. *On Levels of Model Transformation.* In Proc. of XML Europe 2000, 2000.

[22] R. Pohjonen and S. Kelly. *Domain-specific Modeling.* Dr. Dobb's Journal, #339, pages 26-35, August 2002.

[23] T. Quatrani. *Visual Modeling with Rational Rose and UML*. Addison-Wesley Object Technology Series, 1997.

[24] A. Radermacher. *Support for Design Patterns through Graph Transformation Tools.* In Proc. Applications of Graph Transformations with Industrial Relevance (AGTIVE'99), pages 111-126, 1999.

[25] K. Schloegel, D. Oglesby, E. Engstrom, D. Bhatt. *A New Approach to Capture Multi-model Interactions in Support of Cross-domain Analyses*. Honeywell Laboratories Technical Report, 2001.

[26] World Wide Web Consortium (W3C). *Extensible Markup Language (XML), Version 1.0*. http://www.w3.org/XML/.

[27] World Wide Web Consortium (W3C). *XSL Transformations (XSLT), Version 1.0*. http://www.w3.org/TR/xslt.

[28] XSL Accelerator (XSLerator) is described and available at http://www.alphaworks.ibm.com/tech/xslerator.