Online Collaborative Environment for Designing Complex Computational Systems

Miklós Maróti[2], Róbert Kereskényi[1], Tamás Kecskés[1], Péter Völgyesi[1], and Ákos Lédeczi[1]

[1] Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA
akos.ledeczi@vanderbilt.edu
[2] Bolyai Institute, University of Szeged, Szeged, Hungary
mmaroti@math.u-szeged.hu

Abstract
Developers of information systems have always utilized various visual formalisms during the design process, albeit in an informal manner. Architecture diagrams, finite state machines, and signal flow graphs are just a few examples. Model Integrated Computing (MIC) is an approach that considers these design artifacts as first class models and uses them to generate the system or subsystems automatically. Moreover, the same models can be used to analyze the system and generate test cases and documentation. MIC advocates the formal definition of these formalisms, called domain-specific modeling languages (DSML), via metamodeling and the automatic configuration of modeling tools from the metamodels. However, current MIC infrastructures are based on desktop applications that support a limited number of platforms, discourage concurrent design collaboration and are not scalable. This paper presents WebGME, a cloud- and web-based cyberinfrastructure to support the collaborative modeling, analysis, and synthesis of complex, large-scale scientific and engineering information systems. It facilitates interfacing with existing external tools, such as simulators and analysis tools, it provides custom domain-specific visualization support and enables the creation of automatic code generators.

Keywords: model-based software, online collaboration, automatic code generation, web-based design environment

1 Introduction

While models play a central role in almost all science and engineering disciplines, model-based development of information systems is still the exception, rather than the rule. Note that in the context of this paper, models mean such formalisms as finite state machines, flowcharts, class diagrams, etc. Even when modeling is utilized during the development of an information system, for example, by using the Unified Modeling Language (UML) [18] to describe object-oriented design in the Model-Driven Development approach (MDD) [14], its primary

purpose is to facilitate communication about the system among team members and to provide documentation [19]. Software is still created overwhelmingly by writing source code manually.

In the early nineties, predating MDD, an approach to model-based design of software intensive systems was developed independently at multiple places. As such, it is called by various names: domain modeling [8](Honeywell), MetaCase [10](Metacase Consulting) and Model Integrated Computing [24](Vanderbilt). For the rest of the paper, we shall refer to this approach as MIC. The underlying idea behind MIC is to facilitate the design of domain-specific graphical modeling languages (DSML) for various engineering, science, and other domains. The approach was motivated by the fact that most disciplines already have their own mature formalisms and it is better to adapt the design tools to match these than to force domain experts to use a "universal" language. MIC tools provide facilities to formally define the DSML with metamodels and to automatically configure a tool suite to support the language. The tools come with multiple APIs to enable system analysis, simulation, and code generation and to interface with existing tools.

MIC has numerous advantages. The DSML abstracts the commonalities in the domain, while the models capture the information specific to the given system being modeled. A well-designed DSML will be inherently familiar and natural to users, the domain experts. The same set of models can be used to analyze the system, provide input to simulators, generate test cases, create documentation, and synthesize (parts of) the system. Hence, the technique provides guaranteed consistency, because every tool utilizes the same set of shared models. Maintenance and evolution of information systems are much easier with MIC, as many times only the system models need to be updated without the need to change the underlying software.

MIC has enjoyed considerable success. Metacase has been in business for two decades selling their tools and providing consulting services for companies that use them. Many new environments have been created such as the Eclipse Modeling Framework (EMF) [22]. The open source MIC toolsuite built around the Generic Modeling Environment (GME) [11] has been applied successfully in many different domains [12, 13, 20, 4, 5, 6, 21].

In spite of the many advantages MIC brings to system development, however, the various MIC tools have numerous drawbacks as well. Scalability is probably the single most important issue as the desktop-based tools were never designed for the development of very complex languages and huge models built by geographically distributed teams. Model and DSML version control is another missing, yet very important feature for domain evolution support. GME has been often criticized for limited extensibility as far as model visualization and the user interface was concerned. During the 15 years of its widespread application, feature creep has set in. It was time to return to the drawing board and rethink how MIC tools should be architected from the ground up.

This paper introduces WebGME, the cornerstone of a new generation of MIC tools. The main design drivers were scalability, version control and collaboration support. WebGME supports the design of domain-specific modeling languages via metamodeling. The metamodels and system models are tightly integrated and stored in a version controlled database in the cloud. Online collaboration is transparently supported. Clients are web browser-based, resulting in platform independence and seamless client updates. The user interface supports multiple built-in visualization techniques as well as tools to create highly domain-specific views/editors. Multiple APIs are provided to interface with existing external tools, such as other modeling environments, databases, simulators, and analysis tools, as well as to enable the development of code generators.

This paper provides a description of the design of the WebGME environment along with our initial impressions using the first prototype.

2 DSML Design

Figure 1 shows the classical four-layer meta/modeling architecture [16]. At the top, the meta-metamodel defines the metamodeling language using the metamodeling language itself. In the figure, we have used the UML class diagram notation for the metamodeling language. The metamodels specify the DSML, in this case a simple Finite State Machine (FSM) language. The domain models are created using the DSML and they define the target information system.

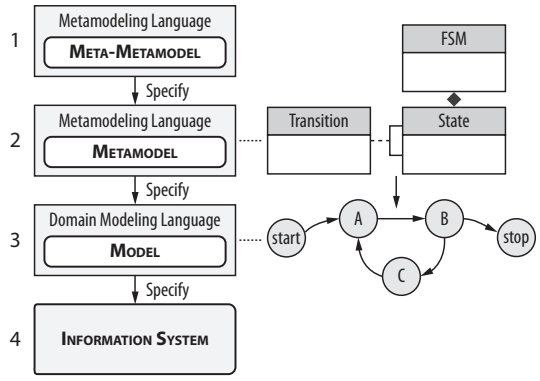

Figure 1: Four-layer metamodeling

Typically, only certain aspects of the target system are captured in the domain models. One of the most important decisions when designing a DSML is the appropriate abstraction level. In this simple example, we captured the state machine logic of a communication protocol or an embedded device. In a more realistic case, one might include more details about the FSM such as events and conditions, and provide orthogonal modeling aspects, for example, to capture hardware configuration, message formats, data types, etc.

The meta-metamodel defines the elementary modeling concepts that are the fundamental building blocks of the approach. These typically include composition, inheritance, aggregation, various associations, and attributes. Which ones to include, how to combine them, and what editing operations to support are the most important design decisions that will ultimately decide the expressiveness of the DSMLs one can create and the usability of the modeling environment itself.

Composition is the principal model organization principle in WebGME. The lifecycle of the objects contained within a model, that is, its children, is tied to that of the container. Copying, moving, or deleting a model will copy, move, or delete its children recursively down the composition hierarchy. A unique concept in GME, and WebGME as well, is prototypical inheritance. Each domain model, that is, a component at any point in the composition hierarchy, is a prototype that can be derived to create an instance model. An instance is a deep copy, i.e., a copy of the entire composition hierarchy rooted at the model, but it establishes a dependency relationship between the original objects and their corresponding new instances. Of course, instances can be used again as prototypes to create an inheritance tree. Any changes in the prototype automatically propagate down this tree. Similarly to the composition hierarchy, deleting a model deletes the entire inheritance tree rooted at the given model.

In WebGME, we have applied this inheritance concept to merge the top three layers in Figure 1. Traditionally, the meta-metamodel, the metamodel, and the models are well separated layers. For example, GME takes the metamodels and generates a domain specification document that a new instance of GME reads in and configures itself to support the new DSML. This strict separation makes it cumbersome to evolve the DSML or to refine/specialize the DSML for certain subsets of the models or for certain users. For example, a system integrator may want to force a team member to work with a prebuilt component library only. With current tools, this is not possible.

On the other hand, WebGME blurs the line where metamodeling ends and domain mod-

eling begins by utilizing inheritance to capture the metamodel/model relationship. In this scheme, when the user creates a new project, she only has the built-in meta-metamodel to work with. The user then creates subtypes of various meta-metamodel components to create the metamodel and hence, define the DSML. A subtype of any metamodel component then automatically becomes a domain model. It inherits all of the rules and constraints that define its own "sublanguage," and it can further refine it by adding additional metamodel components. The advantages of this approach are numerous. Metamodel changes propagate automatically and immediately to every impacted model supporting seamless DSML evolution. A metamodel can be specialized anywhere in the inheritance hierarchy and, consequently, any domain model can become a first class language concept to serve as a building block for other models.

In addition to these two fundamental model organization principles, composition and inheritance, there are other modeling concepts that need to be supported. These include associations such as a pointer or reference (one to one); an object with a source and destination pointer (one to two; typically visualized as a connection); and aggregation (one to many). Attributes are also fundamental concepts to capture textual and numerical data or to tie models to external entities. WebGME has very flexible support for these modeling primitives as well.

3 Architecture

The architecture of WebGME is based on the AJAX paradigm, where most of the business logic and all of the visualization logic is executed in the browser. The server provides scalable access to the model database, coordinates the collaboration of clients and manages the execution of data intensive jobs. This architecture enables responsive and highly interactive user interfaces on the client, excellent scalability on the server, and facilitates the graceful handling of network disconnects and even offline editing of models. Figure 2 illustrates the architecture of the tool.

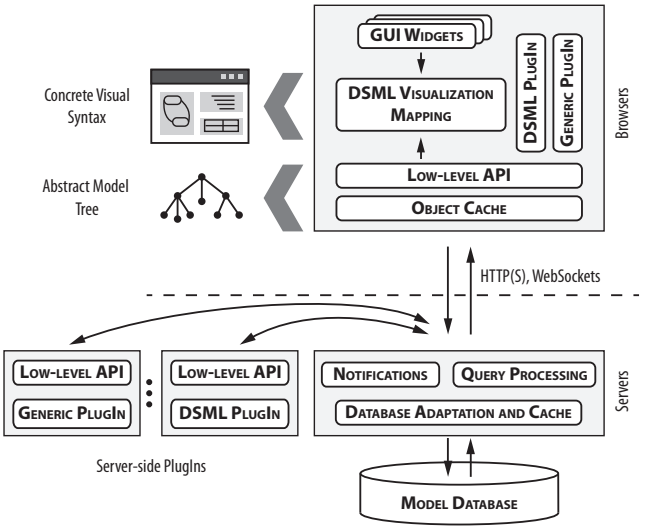

Figure 2: High-level system architecture

The server side of WebGME provides a consistent and versioned model database that clients and analysis tools can query and update. Version control is implemented by treating almost all objects in the database read only, and creating new versions of models (and updating references) whenever they are modified. This design choice allows the use of very efficient document store databases on the server side (e.g., MongoDB), the caching of models on the client side, and a consistent and light-weight notification mechanism of clients during concurrent editing. The client does not need to download all models from the database, only a tiny fraction that are currently visible to the user, so it can work with large databases that would not even fit in the memory of the client.

The primary reason to build domain-specific models is to execute various analysis, simulation, verification, code generation and model transformation tasks on the models stored in the database. Some of these tasks are automatically triggered and executed by certain changes in the model on the client side, but most are executed as batch jobs on the server since they typically need to traverse the whole model hierarchy. Sometimes these jobs can take a long time to execute, but they work on a fixed consistent snapshot of the models, and hence, they are not affected by users making concurrent updates to the model database.

Based on our observation of the evolution of many DSML tools in the past, we believe that at least two different API levels need to be supported. A high-level *domain-specific API* is the most powerful tool in any domain designer's hands, since it uses the concepts of the domain. On the other hand, a low-level *generic API* is also needed for developing domain independent tools. Currently WebGME provides only the lower-level API through standard web interfaces, but we are working on automatically generating the domain-specific API from the metamodels.

The low-level *generic API* allows the scalable manipulation of the abstract model tree stored in the model database, handles versioning, concurrent updates and conflict detection. It is implemented as an asynchronous JavaScript library that communicates with the server database via WebSockets. The same high performance code is executed both in the browser and on the server side in a Node.js container, so we do not have to maintain different versions of the same functionality in different languages. WebGME also provides a highly accessible and language neutral REST API to access the models.

4 Example Domain

To describe a programmable graphical design environment, such as WebGME, in the limited space available here, it is best to go through an example. We picked a simplified hierarchical Finite State Machine (FSM) domain to present the main ideas of the approach. Figure 3 shows a screenshot of the WebGME tool hosted on the Amazon cloud at http://webgme.org. The picture shows the metamodel of the simple FSM language and an example model in the two main windows.

The metamodel on the right hand size is visualized with a notation based on UML class diagrams. The FSM language has three components: FCO, State and Transition. FCO stands for First Class Object and it is the root of the inheritance hierarchy. Any model object created in WebGME is part of the single inheritance tree rooted at FCO. When a new project is created, it comes populated with a model called Root which is the root of the single containment hierarchy, and inside it, there is FCO. Neither Root, nor FCO can be deleted.

In the metamodel, inheritance relationships are shown with a red line ending in a small triangle at the base class (that is, prototype model). The models State and Transition are instances of FCO. The black lines ending in diamonds show composition (containment). More precisely, they specify rules stating that States can contain Transitions and other States, in this particular example. Finally, blue arrows specify pointer relationships. In this case, Transitions have two pointers, named src and dst, such that each points at States. Again, this is a rule saying that each instance of Transition will have an src and a dst pointer that can point to any instances of the State. The src-dst pointer pair has a special meaning: the WebGME built-in default visualizer will display objects having this pair of pointers as connections. (Pointers can be specified with any other name as well, but they are visualized with a simple icon. Double clicking it takes the user to the target of the pointer.)

Consider the left side of Figure 3 showing an example FSM model (called Example) with four States (A, B, C, D) and Transitions between pairs of them. The right side of the figure

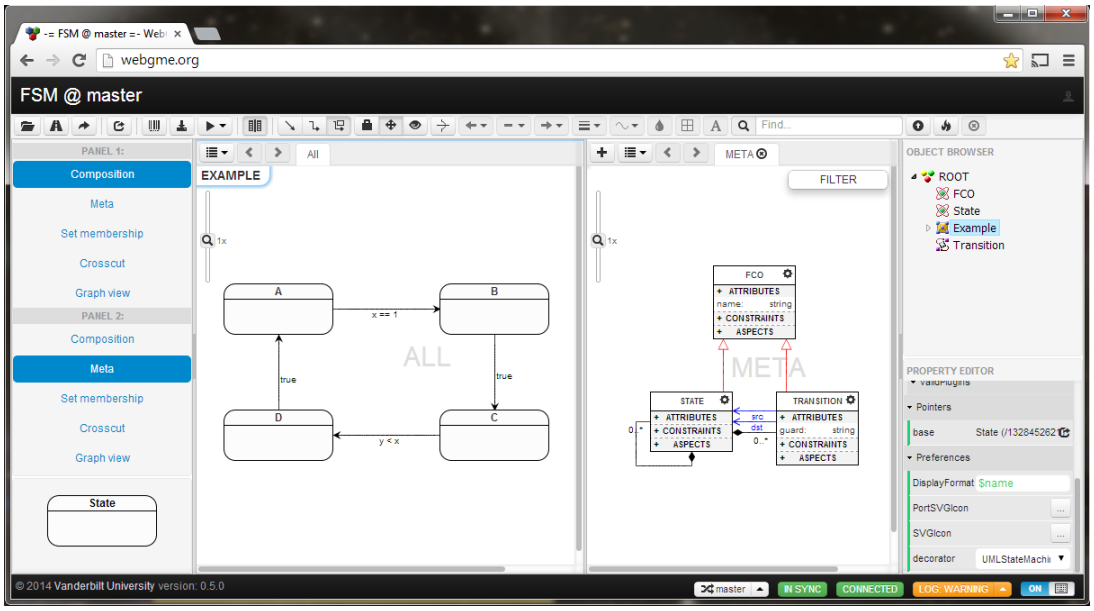

Figure 3: Example domain: Hierarchical Finite State Machines

shows the Object Browser window that presents the composition hierarchy of the project. It can be seen that Transitions are objects just like States (for example, they could contain other objects if it made sense in the domain and were allowed by the metamodel). They just happen to be visualized with lines in the model editor. On the other hand, the lines in the metamodel do not correspond to model objects: red lines represent inheritance relationships, black lines represent composition rules and blue lines represent pointer specifications. This simple example already shows two different visualization techniques for models. WebGME comes with a few more built-in, for example, a spreadsheet like table editor. Even more importantly, anybody can provide custom visualizations for their domain since WebGME is open source and extensible.

The window in the lower left corner is the Part Browser. It shows the set of models that can be inserted (more precisely, instantiated) into the currently open model. In this case, the model Example can only accept States as children (since Transitions are connections, they are not shown here; they are created by connecting two objects in the model editor.) When the modeler drags State from the Part Browser and drops it in the model editor, it creates a new instance of State. If we were to create a new instance of State called XYZ in the metamodel, it would show up as a new box there connected with a red line to the State. It would inherit everything from the State. What that means is that States would be able to contain XYZs, XYZs would be able to contain States, and Transitions would be able to connect pairs of States and XYZs (in any combination). Finally, XYZ would appear in the Part Browser alongside State as a model that can be instantiated into States. Of course, additional rules should be specified for XYZ by the domain designer in the metamodel to make it distinct from State.

Let us briefly outline a few more important concepts of WebGME:

- **Sets**. A pointer specifies a one-to-one relationship. WebGME also supports sets, that are basically pointer lists, that define one-to-many relationships, for example, aggregation. These can be specified with a different colored arrow in the metamodel.

- **Attributes**. Models can have as many textual attributes as the modeler wishes. An example is the guard condition of Transitions in the FSM example. It also illustrates that attributes can be displayed on connections with a preference setting. Attribute specification belongs to the metamodel. Attributes are inherited, but their values can be overwritten in derived models.

- **Aspects**. Aspects provide a way to handle the complexity of models. They specify subsets of children of a model to be shown together in a special view. For example, a model of a car may have three separate aspects where the mechanical, electrical and hydraulic components are shown. The default aspect shows all children of a model. Additional aspects are specified in the metamodel.

- **Crosscuts**. Specifying and visualizing relationships (pointers, sets) that cut across the composition hierarchy is difficult and typically non-intuitive. WebGME introduces the concept of a crosscut that provides a way to show models from across the composition hierarchy together in a shared window. Crosscuts show relationships directly and allows the user to modify them or create new ones. What models show up in a crosscut is controlled by the users: they can drag in models or specify one-time queries on the models database to collect models together. Note that the meta editor is a special crosscut. Any number of crosscuts can be created on-demand.

- **APIs**. The presented web-interface provides the means to specify the modeling language and to create and view models. The main goal of any such environment is to use these models to run various analyses and simulations and to generate test cases, documentation and software. WebGME provides a native Javascript API and a generic REST API to interface with external tools and to enable code generation. A high-level domain-specific API (to be automatically generated from the metamodels) is in the works also.

The above FSM example was necessarily simple to be able to highlight the major concepts in WebGME. For a quick illustration of the scalability of the approach, we selected the domain of digital circuits because it is well known, non-trivial and has a widely accepted visual formalism. Figure 4 depicts a part of the metamodel and an example circuit model. This example illustrates nicely that even the metamodel can become complex, so WebGME allows the user to break it up into multiple sheets. More precisely, there are multiple meta crosscuts in this project each showing a user-selected subset of the metamodel. In the figure, the Gates meta crosscut is shown with the tabs indicating that three additional ones are also defined (Meta, IOs, Connections). Note that the exact same WebGME software is used here as for the FSM example. In the digital circuit domain, however, another visualizer is used in the model editor as can be seen in the figure.

5 Related work

The Unified Modeling Language [18] is a widely accepted modeling language for software system design. However, as a single standard language cannot fit all domains well, UML has two extensibility methods [9]: metamodeling (e.g. the Common Warehouse Metamodel (CWM) [15]) and UML profiles (e.g. SysML [17] among many others). The first extensibility method usually results in a very complicated metamodel, while the second solution retains too much of the original UML to be truly customizable for each domain. In both cases, the main advantage of UML—that is, being a standard with agreed upon semantics—is lost, and there is a separate

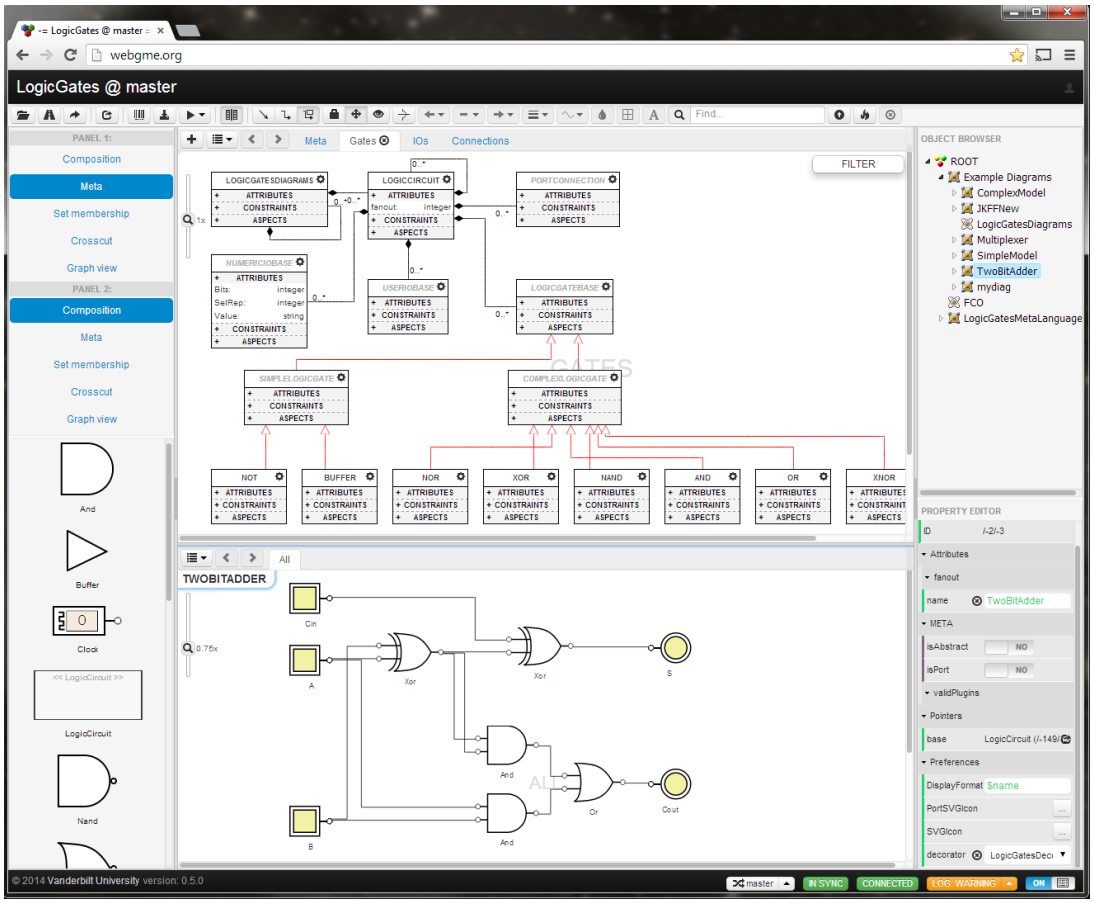

Figure 4: Example domain: Digital Logic Circuits

standardization process for every new language. Hence, we believe that there is no compelling reason to build DSMLs on top of UML for many engineering and scientific disciplines.

Today, graphical model building tools are predominantly desktop applications. The best known environment is the Eclipse framework [22] and many tools are implemented as Eclipse plugins. While Eclipse runs on many platforms, installing and maintaining a complex Eclipse environment is cumbersome and does not solve collaboration. VisTrails [7] is another standalone graphical tool for the domain of scientific workflow management that focuses on exploration, visualization, and data analysis. It is extensible and maintains a history of changes, but does not support collaboration other than an exchange service where users can share their workflows. Web technologies, on the other hand, have advanced to the point where it is feasible to build user-friendly and visually appealing user interfaces with good performance inside a web browser. Lucid Charts [3], CircuitLab [1], or Creatly [2] are excellent examples of what is possible. Some of them even support online collaboration. On the other hand, these tools employ relatively simple, typically flat data models and are very specific to their respective domains. They do not solve the challenges associated with evolutionary language design, configurability, branching, and extensibility. AToMPM [23], a web-based metamodeling and transformation tool for Multi-Paradigm Modeling, is the most similar approach to our work.

6 Conclusions

The paper presented a novel collaborative web-based software environment for the development of domain-specific modeling languages and corresponding models. The current alpha release of WebGME is available at http://webgme.org. It is already a fully functional environment. However, there are quite a few additional features that we are working on to provide an even more powerful tool for the community. While WebGME currently supports model versioning, semi-automated merging of different versions of the same project is not yet available. Another useful feature is the concept of model libraries. The idea is very similar to that of software libraries: an entire project with a language and a set of models can be packaged as a library and distributed for others to use. Note that both of these problems are much harder to support for a rich hierarchical data model that includes inheritance than for flat data such as source code and binaries.

6.1 Acknowledgement

This work was sponsored in part by the Defense Advanced Research Project Agency (DARPA) and by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013).

References

[1] CircuitLab. `https://www.circuitlab.com`. Cited 2013 Mar 14.

[2] Creately. `http://creately.com`. Cited 2013 Mar 14.

[3] Lucidchart. `http://www.lucidchart.com`. Cited 2013 Mar 14.

[4] Hamid Bagheri and Kevin Sullivan. Monarch: model-based development of software architectures. *Model Driven Engineering Languages and Systems*, pages 376–390, 2010.

[5] Jean Bézivin, Christian Brunette, Régis Chevrel, Frédéric Jouault, and Ivan Kurtev. Bridging the generic modeling environment (GME) and the eclipse modeling framework (EMF). In *Proceedings of the Best Practices for Model Driven Software Development at OOPSLA*, volume 5, 2005.

[6] Peter Bunus. A simulation and decision framework for selection of numerical solvers in. In *Proceedings of the 39th annual Symposium on Simulation*, ANSS '06, pages 178–187, Washington, DC, USA, 2006. IEEE Computer Society.

[7] Steven P Callahan, Juliana Freire, Emanuele Santos, Carlos E Scheidegger, Cláudio T Silva, and Huy T Vo. Vistrails: visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 745–747. ACM, 2006.

[8] Eric Engstrom and Jonathan Krueger. Building and rapidly evolving domain-specific tools with dome. In *Computer-Aided Control System Design, 2000. CACSD 2000. IEEE International Symposium on*, pages 83–88. IEEE, 2000.

[9] Lidia Fuentes and Antonio Vallecillo. An introduction to UML profiles. *UPGRADE, The European Journal for the Informatics Professional*, 5(2):5–13, 2004.

[10] Steven Kelly and Juha-Pekka Tolvanen. *Domain-specific modeling: enabling full code generation*. Wiley-IEEE Computer Society Press, 2008.

[11] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G Karsai. Composing domain-specific design environments. *IEEE Computer*, pages 44–51, 2001.

[12] Earl Long, Amit Misra, and Janos Sztipanovits. Increasing productivity at saturn. *Computer*, 31(8):35–43, 1998.

[13] Janos L Mathe, Akos Ledeczi, Andras Nadas, Janos Sztipanovits, Jason B Martin, Liza M Weavind, Anne Miller, Peter Miller, and David J Maron. A model-integrated, guideline-driven, clinical decision-support system. *Software, IEEE*, 26(4):54–61, 2009.

[14] Stephen J Mellor, Tony Clark, and Takao Futagami. Model-driven development: guest editors' introduction. *IEEE software*, 20(5):14–18, 2003.

[15] OMG. *Common Warehouse Metamodel Specification 1.1*, 2003. OMG doc. smsc/2003-03-02.

[16] OMG. Meta object facility (MOF) 2.4.1 core specification, 2003. OMG doc. formal/2011-08-07.

[17] OMG. *OMG Systems Modeling Language Specification 1.3*, 2011. OMG doc. ptc/2011-08-10.

[18] OMG. *Unified Modeling Language Specification 2.4.1: Infrastructure and Superstructure*, 2011. OMG doc. smsc/2011-08-05.

[19] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[20] S. Shetty, Sandeep Neema, and T. Bapty. Model based self adaptive behavior language for large scale real time embedded systems. In *Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the*, pages 478–483, May.

[21] J.A. Stankovic, Ruiqing Zhu, R. Poornalingam, Chenyang Lu, Zhendong Yu, M. Humphrey, and B. Ellis. Vest: an aspect-based composition tool for real-time systems. In *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, pages 58–69, May.

[22] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Addison-Wesley Professional, 2008.

[23] Eugene Syriani, Hans Vangheluwe, Raphael Mannadiar, Conner Hansen, Simon Van Mierlo, and Huseyin Ergin. Atompm: A web-based modeling environment. MODELS, 2003.

[24] Janos Sztipanovits and Gabor Karsai. Model-integrated computing. *Computer*, 30(4):110–111, 1997.