Spread: A Remote Collaboration Architecture for Domain Specific Modeling

Scott Campbell, Jules White, and Douglas C. Schmidt
scott.h.campbell@vanderbilt.edu, {jules,schmidt}@dre.vanderbilt.edu
Vanderbilt University

Abstract

Domain-Specific Modeling Languages (DSMLs) are graphical modeling languages designed to mirror the notations and terminology of a particular target domain. This paper presents an approach, called Spread, to create a generalized remote collaboration framework for DSML modeling. Spread allows meta-programmable modeling environments to use a common remoting framework across a diverse set of DSML languages, despite differences in the terminology, structure, and semantics of each language.

1. Introduction

Domain-Specific Modeling Languages (DSMLs) [7] are graphical modeling languages that have customized terminology and semantics for a specific target domain. For example, in the automobiles domain multiple DSMLs are commonly used, , including MatLab Simulink [9] for modeling the continuous and discrete behavior of a car's Electronic Control Units (ECUs), AUTOSAR [6] to describe the functional behavior of the automobile's software and its interaction with ECU functions, and Feature Models [10] to capture variations in how the ECUs and software components can be configured.

A challenging aspect of developing a new DSML is creating a graphical environment that allows modelers to visualize and manipulate instances of a modeling language. Meta-programmable modeling environments (such as the Eclipse Graphical Modeling Framework (GMF) [5], the Generic Modeling Environment (GME) [8], or the Generic Eclipse Modeling System (GEMS) [11]) apply model-driven engineering techniques to create DSML modeling environments. Developers use meta-programmable modeling environments to specify a DSML using a metamodel specification language—the grammar and semantics of a language. The meta-programmable environment then generates the requisite code to implement a visual modeling environment with the terms, notations, and constraints of the DSML specified by the metamodel.

Frequently in large-scale development scenarios, developers need the ability to collaborate on a single or group of DSML models. Despite the ability of meta-programmable modeling environments to rapidly generate DSML modeling environments without requiring manual coding, they have previously been unable to automatically generate a remote collaboration framework that allows multiple developers to interact with a set of models simultaneously. Developers have thus hereto been forced to implement these complex remote collaboration frameworks from scratch.

To address this limitation of meta-programmable modeling environments, this paper presents a DSML modeling collaboration architecture, called *Spread*, that provides a general remote collaboration framework that can be used across multiple DSMLs. For each DSML, Spread uses a predicate-logic based remoting protocol to manage model updates across modeling clients. Spread also employs a hierarchical peer-to-peer based modeling client topology that allows model contents to be distributed selectively across clients. Each client maintains a separate view of the model and a total ordering of model updates to prevent inconsistent states.

The remainder of this paper is organized as follows: Section 2 outlines an example from the automotive domain to motivate the need for Spread; Section 3 describes the challenges of building a generalized remote collaboration framework for DSMLs; Section 4 summarizes the Spread architecture; Section 5 compares Spread with related work; and Section 6 presents concluding remarks.

2. Motivating Example

As a motivating example for this paper, we use the collaborative design of an automotive application. Figure 1 depicts an example modeling scenario for the development of an automobile. A lead developer is responsible for the overall architecture of the automobile. Directly beneath the lead developer are two integration engineers responsible for integrating the parts provided by third-party software and hardware suppliers into the overall automobile architecture. Finally, the third-party suppliers receive requirements models from the integration engineers and fill in the models

with the specifics of the components that they are supplying.

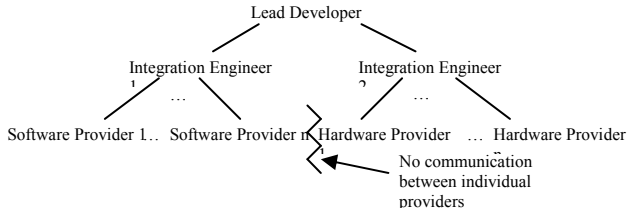

Figure 1: Collaborative Automotive Modeling

3. Challenges of DSML Modeling Collaboration

This section describes four challenges associated with DSML modeling.

Challenge 1: Selective client views. One challenge that must be met by a collaborative DSML modeling solution is how to provide different levels of access to the model for different users. It is entirely possible that for some applications it makes sense to have some users who only need to see part of the model, other users that need to see other parts of the model, and yet other users that need to see the entire model. Since some users only need to see part of the model, it follows that these users also only need to to update these same parts of the model; otherwise unexpected changes and errors could result. Any collaborative DSML modeling solution should be able to provide these selective views to individual users, as well as selective access to only the parts of the model that the individual user can see.

Our motivating example from Section 2 shows this limitation of access clearly. The third-party suppliers each receive the requirements of only the components that they are supplying, which they use to fill the models with their specific components. These specifics must then be incorporated in the overall model of the automotive architecture. It must be possible for the two integration engineers to make any changes necessary to the specific models provided by the third-party suppliers, and have those changes appear on the models of affected suppliers.

Challenge 2: Preventing update duplication without domain semantics. A major challenge to the prospective collaborative DSML modeling solution is to define how updates propagate through the topology of connected models. The simplest solution is to have every model broadcast an update to every other model it is connected to whenever it receives one from another source or generates one on its own. This could result, however, in a model receiving an update, implementing it, and passing it on to another model, only to receive the same update again from another source, try to implement it again, and pass it on again. This process could result in an infinite loop of the same update going around the topology repeatedly, causing unwanted changes and tying up system resources so that no new updates can be initiated. Any collaborative DSML modeling solution will have to devise with a way of detecting and removing these duplicate updates, at the level of each individual model.

Creating a generic remote collaboration framework for DSMLs is challenging because domain knowledge in a model is used to determine if the model already has the change specified by the incoming update. For example, if an update event that creates a new state arrives at a Statechart model, the infrastructure needs to find the root state machine, iterate over its states, and see if the specified state already exists. If the model is a CAD model of the automobile suspension, however, a completely different series of checks will be needed to determine if the incoming change has already been applied to the model.

Challenge 3: Prioritizing updates without domain semantics. Another challenge that must be handled by a collaborative DSML modeling solution is what to do if two updates reach a model at the same time. This situation is complicated with DSMLs because the event that takes precedence can vary depending on the language. In our automotive case study, for example, it might make sense in some DSMLs for a delete update to take precedence over a change attribute update because the change attribute update would not change whether or not the object required deletion, but this might not always be the case. Providing a DSML-independent method of handling update ordering is complicated.

Challenge 4: Collaboration protocol generality. The final challenge a collaborative DSML modeling solution must solve involves being sufficiently generic to apply to all models. Each DSML has a unique blend of terminology, notations, and semantics that make it hard to create a generalized collaboration framework for each language instance. The solution cannot rely on the particulars of any one DSML, such as fields with a certain name or certain types of entities always being present in the model, but must be independent of these restraints. In addition, the collaborative solution must be capable of finding an unknown number of fields and entities of unknown names that might or might not exist in any particular DSML. The need for generality further complicates the previous challenges

since the solutions cannot depend on any type of field or entity being present in all models.

4. The Spread DSML Collaboration Architecture

The Spread DSML modeling collaborative architecture provides a peer-to-peer client topology to achieve truly generic collaborative capabilities. By choosing a peer-to-peer approach instead of a client-server approach, Spread makes it easier to provide unique views and access to all participating peers. In a client-server system, it is often necessary to standardize all views around the one provided by the server. In contrast, by eliminating the server, Spread makes it possible for each peer to maintain its own view.

Spread uses a message-based architecture built on message-oriented middleware, meaning that the individual peers can exist separately until an update is disseminated, process the update when it arrives, and then return to independent operation. In this design each model is a topic that can be subscribed to by other models to create the topology. Moreover, models can subscribe to message topics that come from other devices, such as sensors. They can also only subscribe to certain types of messages, which is enough to provide unique access when necessary. By implementing a peer-to-peer client topology, Spread easily provides separate client views and security.

Spread's peer-to-peer client topology does have its drawbacks, however, since it is harder to provide protection against update duplication and update ordering because it is possible for the server to maintain an authoritative version of the model, while all clients simply copy the server's model. The client-server approach avoids the problems of update duplication and update ordering.

To avoid update duplication using Spread, each model checks whether an update will cause any changes before actually applying it. If the update does not cause changes, it is simply discarded and the model remains in its original state. If the update *does* cause changes, it is applied to the model and broadcast to all connected models. Each change will therefore be applied to each model only once and all other redundant updates will eventually be dropped.

Spread achieves update ordering by forcing the peer-to-peer topology into a tree structure, as shown in Figure 2. This tree structure is used to enforce priority in the event of an update collision, by determining that each model has priority over all models beneath it in the tree. In Figure 2, the clients are numbered in order of their priority, with 1 being the highest priority and 8 being the lowest. Clients 2 and 3 actually have the same priority, because they are on the same level in the tree, but since there is no direct connection between them, this will not cause problems.

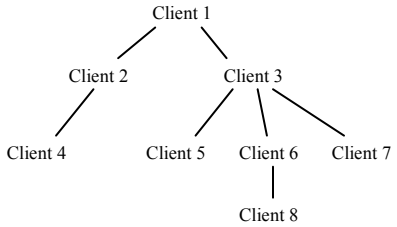

Figure 2: The Peer-to-peer Client Topology

By forcing a tree structure onto the peer-to-peer client topology, Spread does not lose any generality. In software design applications, a tree is almost always the natural topology of the connected users. Typically, a project manager, technical lead, or other arbitrator makes the final decision on how to reconcile conflicting design forces. In our automotive example, Figure 1 shows how the client topology is easily placed into a tree structure, with the lead developer as the root. A client-server software development approach can also be forced into a natural tree structure with the server as the root and the clients as the branches.

Predicate-logic Collaboration Protocol:

A key distinction between Spread and other modeling collaborative architectures is Spread's generality. Our initial implementation of Spread is an extension of the GEMS meta-programmable modeling environment, so it can work with any DSML generated using Eclipse Modeling Framework models and GEMS for visualization. It relies on the basic constructs provided by GEMS (described in [11]) to apply updates for any DSML. This approach provides a terminology for Spread to use that is common across all GEMS DSMLs, so Spread can provide the collaborative capabilities necessary for any of these DSMLs. Spread provides these capabilities using a predicate-logic collaboration protocol.

Update events in Spread are written using the ordered triple defined by <type of update, id of object being updated, new value>. The different types of updates possible are defined by the specific DSML being used, but there is a small set of update types that are always implemented for every entity in the DSML. New updates are always defined using a recognizable pattern when the code is generated for the DSML, two users running the same DSML will always have the same set of types of updates defined. These updates are resolved using a predicate-logic approach, which can resolve updates without Spread knowing the exact nature of the update.

For example, if some specific DSML has an object with an attribute called *name*, when an update to

the name of that object is changed, the update <self_name, id1, New_Name> will be generated. Any model that receives this update will know that all update types begin with the prefix "self_", so it will remove that and then be left with the attribute to change (*name*). It can then call a generic function and pass it the attribute to change (*name*) ,the id of the object on which to perform the change (*id1*)m and the new value to change the attribute to (*New_Name*). Other types of updates in Spread are handled similarly.

Since Spread need not know the specifics of any DSML using it, it can provide the generality necessary for use by a large number of DSMLs. This generality stems from the fact that the Spread need not change, no matter how different or complex two different models that use it. In this way, Spread solves the challenge of generality, and therefore all challenges inherent in building a DSML modeling collaborative architecture.

5. Related Work

There have been previous attempts to develop collaborative modeling techniques for limited domains, but none have been as generic as Spread. Most attempts have been for educational applications, such as the ModellingSpace architecture [1]. ModellingSpace provides an interactive online environment where students and teachers can cooperate in virtual science experiments. It provides a set of primitive building blocks that users can assemble into whichever experiment they want to perform, and then fully collaborate with any other users while performing the experiment. This architecture is limited since it requires a certain set of primitive building blocks for collaboration to occur. Spread, however, can provide the same collaborative capabilities as ModellingSpace for a much larger and more generic group of applications.

Another example of a collaborative modeling technique for a specific DSML is the Collaborative Spiral Process Model [4]. This collaborative process is specific to one DSML, meaning that it is not as generic as Spread. A final example of a collaborative modeling technique for a specific DSML is webSPIFF [2] [3], which provides a web-based collaborative modeling platform for Feature Models in CAD systems similar to Spread, but again not as generic. Both techniques provide features similar to Spread in their respective domains, but they do not provide the same level of generality as Spread.

6. Concluding Remarks

We have presented the Spread generic DSML modeling collaborative architecture to address a lack of such capabilities in the meta-programmable modeling environments available today. Spread facilitates collaboration between any number of users using a peer-to-peer client topology, with allowances for unique views and unique access. Spread uses a predicate-logic based collaboration protocol that allows it to handle generic updates. Its primary advantage over other modeling collaborative architecture is its generality, i.e., it can be used with any DSML that can be modeled using the GEMS modeling environment.

7. References

[1] Avouris, N. and Margaritis, M. and Komis, V. and Saez, A. and Melendez, R., "ModellingSpace: Interaction Design and Architecture of a collaborative modelling environment", *Proc. of 6th CBLIS*, 2003, pages 993—1004,

[2] Bidarra, R. and van den Berg, E. and Bronsvoort, W.F., "Collaborative modeling with features", *2001 ASME Design Engineering Technical Conferences*, September 9-12, 2001, Pittsburgh, Pennsylvania, United States

[3] Bidarra, R. and van den Berg, E. and Bronsvoort, W.F., "Web-based collaborative feature modeling", *Proceedings of the sixth ACM symposium on Solid modeling and applications*, 2001, pages 319-320

[4] Boehm, B. and Bose, P., "A Collaborative Spiral Software Process Model Based on Theory W", *Software Process, 1994. 'Applying the Software Process', Proceedings., Third International Conference on the*, Oct 10-11 1994, pages 59-68

[5] Eclipse Consortium, *Eclipse Graphical Modeling Framework (GMF)*, 2005, available at <http://www.eclipse.org/gmf>

[6] Heinecke, H. and Bielefeld, J. and Schnelle, K.P. and Maldener, N. and Fennel, H. and Weis, O. and Weber, T. and Ruh, J. and Lundh, L. and Sandén, T. and others, "AUTOSAR--Current results and preparations for exploitation", *7th EUROFORUM conference: Software in the vehicle*, May 3-4 2006, Stuttgart, Germany

[7] Ákos Lédeczi, Árpád Bakay, Miklós Maróti, Péter Völgyesi, Greg Nordstrom, Jonathan Sprinkle, Gábor Karsai, "Composing Domain-Specific Design Environments," *Computer*, vol. 34, no. 11, Nov., 2001, pages 44-51

[8] Ledeczi, A. and Maroti, M. and Bakay, A. and Karsai, G. and Garrett, J. and Thomason, C. and Nordstrom, G. and Sprinkle, J. and Volgyesi, P., "The Generic Modeling Environment", *Workshop on Intelligent Signal Processing,* May, 2001, Budapest, Hungary

[9] MatLab Simulink, available at <http://www.mathworks.com/products/simulink/>

[10] Riebisch, M., "Towards a More Precise Definition of Feature Models", *Modeling Variability for Object-Oriented Product Lines*, 2003, pages 64-76

[11] White, J. and Schmidt, D.C. and Mulligan, S., "The Generic Eclipse Modeling System", *Model-Driven Development Tool Implementors Forum at TOOLS*, June 24, 2007, Zurich, Switzerland