Towards Integration of Policies into DSMLs

Frank Hernandez and Peter J. Clarke
School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
{fhern006, clarkep}@fiu.edu

ABSTRACT

As domain-specific modeling languages (DSMLs) become more widely used, it is important to develop approaches for creating DSMLs that allow different aspects of the language to be incrementally added. For these approaches to be effective the new aspects (or features) of the DSML should be added using an automated or semi-automated approach. By creating such approaches the developers of a DSML can start with the constructs to describe the main functionality in the domain, then add those features to the DSML necessary to specify the non-functional constraints in the domain. In this paper we present a semi-automatic approach to integrate a policy language into an existing DSML. As proof of concept we apply the approach to a simple DSML from the bookstore domain.

Keywords

Graphical Domain-Specific Language, Model-Driven Development, Model Composition, Policy

1. INTRODUCTION

The use of domain-specific modeling languages (DSMLs) continues to grow as software developers seek new techniques to increase their productivity in developing applications for specific domains [9]. A key aspect of domain-specific modeling (DSM) is raising the level of abstraction by thinking about the solution to a problem in concepts from the given domain. During the past three years we have been involved with the development of a DSML for user-centric communication services, the Communication Modeling Language (CML) [14]. CML models created by the user (novice or expert) are interpreted and realized by the Communication Virtual Machine (CVM). We have recently started the development of a DSML for the Smart Microgrid domain, Microgrid Modeling Language (MGridML) [1], using a similar approach to CML. Models created in MGridML will be interpreted and realized using the MGridVM.

An iterative approach was used to develop CML thereby making it easy to get a prototype working quickly in order to test the feasibility and practicality of our DSML. The first iteration of CML focused on the basic communication primitives such as create connection, add party to communication, send files, stream media and so on. We are about to add a policy component to CML that will allow users to specify constraints on communication services, such as (1) if bandwidth falls below a given value do not use audio video streaming, or (2) if remote a party has a given user role then encrypt all files sent to that user. The naive approach would be to extend the CML meta-model using a manual approach by integrating the policy aspects into the existing CML meta-model[1]. We would then have to use a similar approach to MGridML.

An alternative approach to integrating a policy component into CML would be to create a partially automated approach base on the research done on model composition [8, 13] and aspect-oriented modeling [6]. We expect that our approach will be generic enough to easily integrate a policy language semi-automatically into both CML and MGridML. In this paper we present our approach to integrating a policy language into an existing DSML with less effort than a totally manual approach. We illustrate how our approach will work using a simple DSML from the bookstore domain.

The remainder of this paper is organized as follows: Section 2 presents some of the current approaches for model composition. Section 3 presents our proposed approach to integrating policies into domain-specific languages. Section 4 has our concluding remarks.

2. LITERATURE REVIEW

The work on model composition has recently gained much attention in the modeling community. Bézivin et al. [2] describe a canonical scheme for model composition that provides a core set of common definitions based on the comparison of three composition frameworks. In this section we describe work in the areas of model composition and modeling aspects using transformations, which is similar to our approach. Although we do not completely use any of the approaches described in this section we do extract some of the basics from model composition and modeling aspects in our approach.

2.1 Model Composition

There are several approaches used to perform model composition including signature-based composition and the use of languages built to support model composition. The signature-based composition approach aims to merge separate models into a single model [8]. These models could be different views of a system containing differing information e.g., security concerns or persistence concerns. This approach is symmetric and requires a mapping to be created between the models based on signatures. Signatures could be retrieved from the model elements and their properties. The signature can be anything such as a class name or the name and

[1]http://cml.cs.fiu.edu//

type of an attribute. This approach suffers from two issues: conflicts and misalignment. Conflicts occur when two elements have two matching signatures but are different on one or more properties. Misalignment on the other hand, happens when two elements have the same signature but refer two separate concepts. Recent work in this area used the notion of composition directive that allows the modeler to define simple model modification that can be done to the model before or after the merge [5].

Kolovos et al. [3, 10] describe the Epsilon Merging Language (EML) which is a rule-based language for merging models of diverse metamodels and technologies. EML includes model comparison and model transformation languages as subsets, and is built on top of the Epsilon (Extensible Platform for Specification of Integrated Languages for mOdel maNagement) platform. EML uses the Epsilon Object Language (EOL) which is a generic model management language that provides other model management facilities such as, model modification, statement sequencing error reporting, among others. The approach by Kolovos et al. uses the following four phases to merge models: comparison, conformance checking, merging, and reconciliation and restructuring. Unlike EML, which is built on the Epsilon platform, we intend to use a more lightweight approach. Our approach use similar phases during the integration of policies into a DSML.

2.2 Modeling Aspects using Transformations

The Modeling Aspects using a Transformation Approach (MATA) [7, 13] uses an asymmetric approach. In asymmetric approaches, one model plays the role of a base model, and another model plays the role of an aspect. In MATA, the aspect model is interpreted as a graph transformation to be applied to the base model. Composition in MATA is achieved in two steps: (1) search for a match pattern in the base model which represents the location where the aspect will be applied. (2) If a match is found, the base model is modified per the composition specification in the aspect model.

In the case of our approach, see Section 3, most of the compositions follow the MATA approach. Such examples are the composition of the *Linker* and the *DSML Specific Linker* where the policy and the DSML meta-models are interpreted as node additions, and the base model is the *Abstract Linker* and the *Linker* respectively.

3. PROPOSED APPROACH

This section presents our model composition approach for integrating a policy language into an existing DSML. This section also presents an example of our approach which uses a simplified meta-model for DSML from the bookstore domain and integrates a policy language. The policies added to this sample language are of the event-condition-action (ECA) paradigm.

3.1 Approach

The first step in integrating a policy language into an existing DSML is to choose the policy language itself. Once the policy language is selected it is provided to the *Linker Composer* along with the *Abstract Linker* model in order to generate the *Linker* meta-model. This linker model is a variation of the the weave model mentioned in the literature [4]. The *Linker* meta-model represents the definition of the

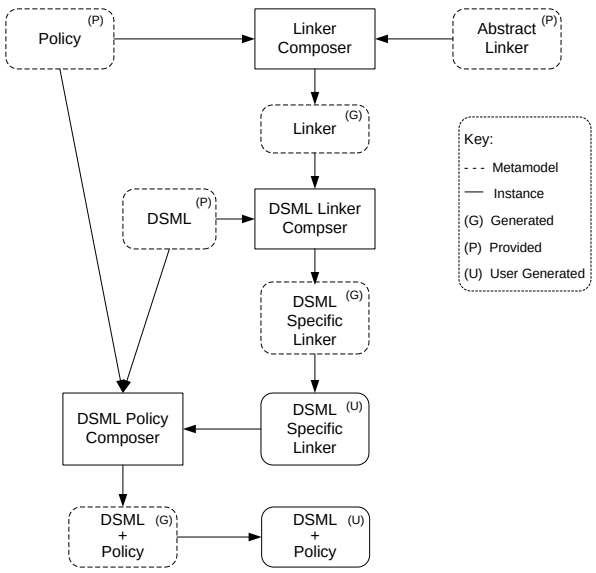

Figure 1: Proposed Approach.

weave [8] model as it pertains to the policy language. It contains the components related to the policy language used. The *Linker* meta-model is then passed to the *DSML Linker Composer* in order to produce a linker that is specific to the domain to which policies will be applied to. During this stage, the possible anchor points to which policies can be added are identified and become part of the final *DSML Specific Linker* meta-model. The attributes of the nodes in the language are then identified and provided as targets for the policies observers.

Once the *DSML Specific Linker* meta-model is generated the user can then specify the specific join points to which to apply policies. This step in the process requires human intervention due to the fact that only the developer of the DSML himself knows what specific nodes require policies. Until this point, all the *DSML Specific Linker* does is provide the user with the set of possible join points from which the user must choose. After the instance model is generated, it is then fed to the *DSML Policy Composer* along with the policy language meta-model and the DSML meta-model. This final step produces a meta-model for the new DSML which now supports policies. This is achieved by providing the nodes specified in the *DSML Specific Linker* instance model with a policy attribute which is an instance of the policy component of the policy language.

3.2 Illustrative Example

In this illustrative example both models (base and aspect) are created using ECore [12], since dealing with models developed in different formats is beyond the scope of this work. The input DSML for this example is the bookstore DSML [11] (Figure 2). For the policy DSML we selected a simple policy language for expressing policies using the event-condition-action paradigm (Figure 3). As can be seen from Figure 4 the *Linker* meta-model has a similar structure to the policy language. The *Linker* meta-model has some of the features from the policy language and it is intended to allow the user to provide information that cannot be extracted

from the DSML but is required for the policies.

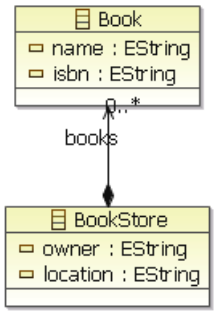

Figure 2: DSML Initial Meta-Model.

For example, in this case, the original DSML had no information regarding events, which is needed to create ECA policies. While this information is not in the language it is known to the language developer. Using the linker the DSML developer can now choose the set of valid events that each of the nodes can have. Some information obtained from the DSML that is required for the integration process is the list of possible anchor points or join points where the policies can be applied. This initial list contains every node in the original DSML that the user can select when generating an instance of the linker model. A user is allowed to select those nodes in the DSML that ultimately will have policies applied to them, since not all nodes in the DSML can have policies applied to them. Only the DSML creator is aware of which nodes one can apply policies to.

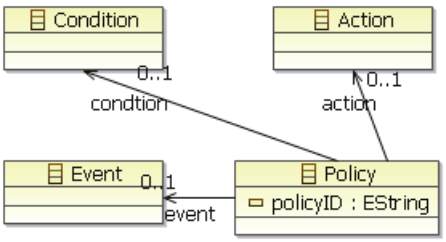

Figure 3: Policy Meta-Model.

As shown in Figure 4 the *Linker* model now contains a notion of events, conditions, and actions. This information was derived from the policy meta-model (Figure 3). In the case of this example, during the composition step, every node in the policy DSML was transfered to the final linker while maintaining their relationships in the original language. At the same time, the nodes have been provided with a *source* field of type `SourcesTypes`. This field represents the possible application location for the events, policies or actions. At this stage of the process, the `SourceTypes` are yet to be initialized thus it remains as an empty enumeration. The `SourceTypes` will be instantiated during the next phase when we examine the original DSML (Figure 2).

Next, we provide this newly created model to the *DSML Linker Composer* along with the bookstore DSML (Figure 2). During this step, the **SourceType** will be populated with

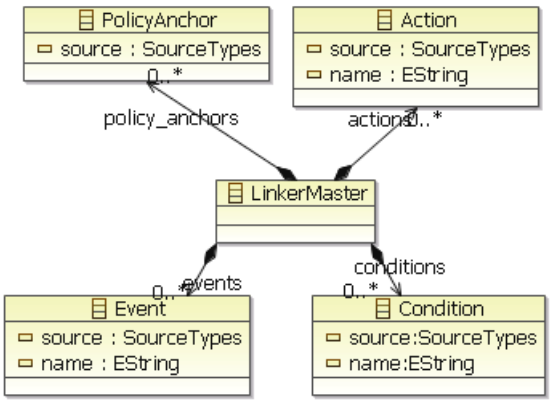

Figure 4: Policy Linker Meta-Model.

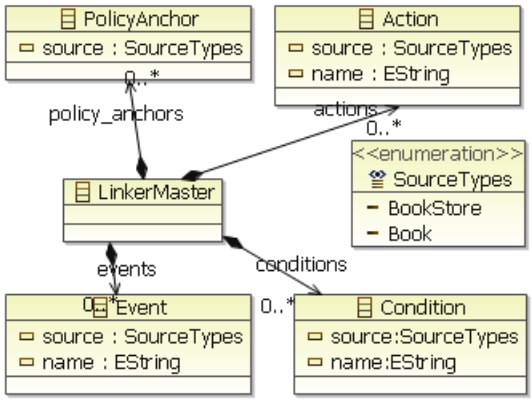

Figure 5: DSML Specific Linker Meta-Model.

all the nodes found in the library DSML. As shown in Figure 5 the `SourceType` enumeration now contains both the `BookStore` and the `Book` nodes from the original DSML. This step is performed to limit the number of choices available to the user to those found in the original DSML. This is done to reduce the number of errors that could happen during the creation of the linker instance if we just allowed the user to type the names for the language nodes himself.

Once the linker instance is created, it is provided as an input for the *DSML Policy Composer* along with the policy meta-model and the DSML meta-model to produce the final DSML meta-model (Figure 6). The linker instance is used by the user to specify the actual nodes that will be part of the final language. In the case of this example the user specified only the `BookStore` as a valid join point for policies (Figure 6). Also, the `bookOld` event and the `sellBook` action, and the bookstore node has been enhanced with a reference to `BookStore_Policy`. This meta-model can now be used to generate models for the bookstore that supports the policies.

4. CONCLUSION

In this paper we presented our proposed approach to integrating policies into existing domain-specific modeling lan-

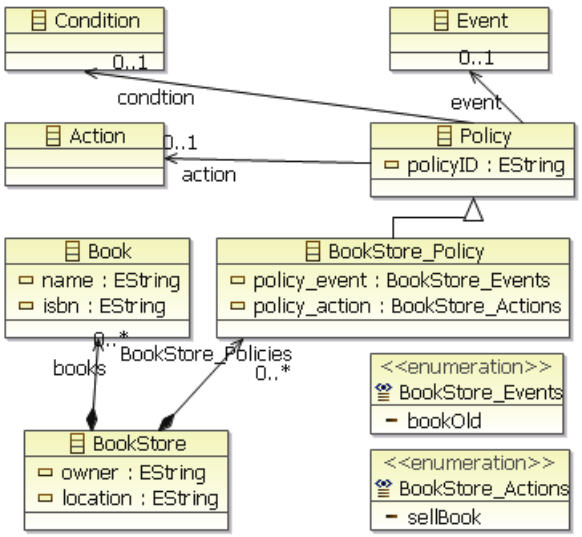

Figure 6: Final DSML + Policy Meta-Model.

guages (DSMLs). We believe that most of the work behind this integration can be automated with some human intervention to supply the information that cannot otherwise be derived from the models. This extra information is provided in our approach via the use of a weave model used to link the DSML meta-model and the policy language meta-model. Finally, we also discussed an initial example of our current implementation of our proposed approach. Our future work includes integrating policies into a DSML for user-centric communication services (*CML - Communication Modeling Language*) and updating the semantics for realizing CML models to include policies. We also plan to perform a similar task on the a DSML for modeling applications in the Smart Microgrid domain, (*MGridML - Microgrid modeling language*).

5. REFERENCES

[1] M. Allison, A. A. Allen, Z. Yang, and P. J. Clarke. A software engineering approach to user-driven control of the microgrid. In *In Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 59–64, 2011.

[2] J. Bézivin, S. Bouzitouna, M. Del Fabro, M.-P. Gervais, F. Jouault, D. Kolovos, I. Kurtev, and R. Paige. A canonical scheme for model composition. In A. Rensink and J. Warmer, editors, *Model Driven Architecture âĂŞ Foundations and Applications*, volume 4066 of *Lecture Notes in Computer Science*, pages 346–360. Springer Berlin / Heidelberg, 2006.

[3] R. P. Dimitios Kolovos, Louis Rose. Epsilon merging language, September 2011. `http://www.eclipse.org/gmt/epsilon/doc/eml/`.

[4] M. D. D. Fabro, J. Bezivin, F. Jouault, E. Breton, and G. Gueltas. Amw: a generic model weaver. In *Proceedings of the 1ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM05)*, 2005.

[5] R. France, F. Fleurey, R. Reddy, B. Baudry, and S. Ghosh. Providing support for model composition in metamodels. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, pages 253–, Washington, DC, USA, 2007. IEEE Computer Society.

[6] R. B. France, I. Ray, G. Georg, and S. Ghosh. Aspect-oriented approach to early design modelling. *IEE Proceedings - Software*, 151(4):173–186, 2004.

[7] P. Jayaraman, J. Whittle, A. Elkhodary, and H. Gomaa. Model composition in product lines and feature interaction detection using critical pair analysis. In G. Engels, B. Opdyke, D. Schmidt, and F. Weil, editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *Lecture Notes in Computer Science*, pages 151–165. Springer Berlin / Heidelberg, 2007.

[8] C. Jeanneret. An analysis of model composition approaches. Master's thesis, Colorado State University and Ecole Polytechnique Federale de Lausanne, 2006.

[9] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, Mar. 2008.

[10] D. S. Kolovos, R. F. Paige, and F. A. C. Polack. Merging models with the epsilon merging language (eml. In *In Proc. ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (Models/UML 2006*, 2006.

[11] N. Singh and R. Babbar. Build metamodels with dynamic emf, August 2011. `http://www.ibm.com/developerworks/library/os-eclipse-dynamicemf/`.

[12] The Eclipse Foundation. Eclipse modeling framework, March 2010. `http://www.eclipse.org/modeling/emf/`.

[13] J. Whittle, A. Moreira, J. Araújo, P. Jayaraman, A. Elkhodary, and R. Rabbi. An expressive aspect composition language for uml state diagrams. In G. Engels, B. Opdyke, D. Schmidt, and F. Weil, editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *Lecture Notes in Computer Science*, pages 514–528. Springer Berlin / Heidelberg, 2007.

[14] Y. Wu, A. A. Allen, F. Hernandez, R. B. France, and P. J. Clarke. A domain-specific modeling approach to realizing user-centric communication. *SP&E*, 2011. `http://onlinelibrary.wiley.com/doi/10.1002/spe.1081/pdf`.