A Demonstration-Based Approach for Designing Domain-Specific Modeling Languages

Hyun Cho
University of Alabama
Department of Computer Science
Box 870290, Tuscaloosa, AL, USA
hcho7@ua.edu

Abstract

Domain-Specific Modeling Languages (DSMLs) have been recognized as a viable solution for reducing the gap between domain abstractions and computational expression within specific domains. In several domains and contexts, DSMLs have been applied successfully to various areas (e.g., finance, combat simulation, and image manipulation) and have shown improvements to productivity and quality. However, development of a new DSML is not an easy task for either computer scientists or end-users because designing and implementing a DSML requires profound knowledge of the domain and deep experience in modeling language development. To address the challenges of DSML development, this doctoral symposium abstract outlines a new approach for building DSMLs that represents a demonstration-based technique for specifying the details of a new modeling language. The approach provides an environment for describing and generating the abstract and concrete syntax of a DSML. Initial work on describing the semantics of a new DSML is also a focus of the work. The research represents an investigation into a technique that allows end-users to sketch (or demonstrate) a domain model with free-form shapes. The goal of the proposed research is to develop the underlying science and tool support to enable end-users to assist in designing a DSML for their domain, while minimizing the typical mundane tasks of DSML development involving many accidental complexities.

Categories and Subject Descriptors D.3.3 [**Programming Languages**]: Language Constructs and Features – abstract data types, Frameworks.

General Terms Design, Languages.

Keywords Domain-Specific Modeling Language; By-Demonstration; Flexible Modeling Tools.

1. Description of Purpose

Domain-Specific Modeling Languages (DSMLs) are languages that support encapsulation and abstraction of a particular domain. Normally, DSMLs provide notations tailored to the domain, and the customized notations offer substantial benefits, such as rich expressiveness and shortened learning curves for non computer scientists who need to express some computational task in a specific domain [9].

As an example, consider embedded report designers who do not have language development expertise, but would benefit from a domain-specific language to capture the diagrams that they currently draw informally. The formats of reports slightly vary according to the characteristics of embedded devices and customer requirements, but much of the reports are common. The job responsibility of these designers is to customize report formats for each embedded device and link each data field with predefined APIs. However, it may be challenging for them to implement quality report formatting because report formats may be hard-coded with a specific programming language. Therefore, the report designers may desire a DSML that can customize a report format through a WYSIWYG editor.

The barriers that prohibit non-programming language experts from developing a DSML are that development of a DSML requires both domain and language development expertise. But learning language development techniques in a short time is difficult and hiring experts that have both domain and language development expertise may be challenging. Thus, in general, at least two experts (domain expert and language development expert) collaborate when creating a new DSML. In addition, DSMLs are often engineered with an iterative and incremental process, but the complexity of the process makes the DSML development tedious, error-prone and time-consuming. The purpose of the research described in this doctoral symposium abstract is to provide a framework that assists domain experts, who have domain knowledge but do not have language development expertise, in building their own DSMLs.

2. Goals of the Research

A DSML is developed by iterating over complex language creation tasks (shown in Figure 1) until the language meets the needed requirements and/or reaches desired quality.

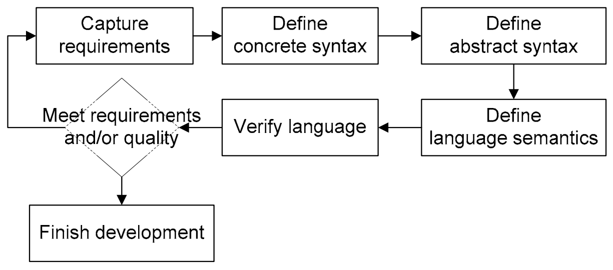

Figure 1. DSML Development Process

DSML development begins by identifying the requirements of a problem domain. After requirements are captured, domain experts and language development experts collaborate to design the concrete syntax, which represents a problem domain. Based on the concrete syntax definition, language development experts define the abstract syntax. In addition, the semantics of a DSML is attached to the abstract syntax (often represented as a metamodel). Finally, domain experts and language development experts collaborate to verify the correctness of the DSML. As described, DSML development requires language development expertise as well as domain knowledge so that domain experts who do not have language development expertise may face several challenges [3] if they try to develop their own DSMLs. From the DSML development process as shown in Figure 1, defining a DSML grammar (both concrete and abstract syntax) and its semantics are challenging tasks even for computer scientists, and automated and systematic approaches are crucial toward resolving the challenges of DSML development. Thus, the framework aims to simplify and automate DSML development by (1) capturing concrete syntax as end-users perform modeling tasks in their domain (or demonstrate their domain through modeling), (2) inferring abstract syntax from concrete syntax and model instances, and (3) associating semantics to abstract syntax.

3. Technical Approach

The overall process for developing a new DSML based on demonstration approach is shown in Figure 2.

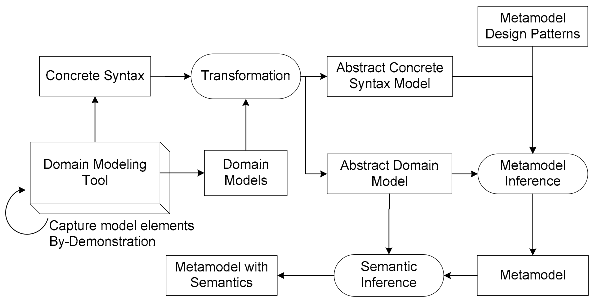

Figure 2. Framework of By-Demonstration Approach

The framework provides a modeling canvas that allows domain experts to draw their domain notations with free-form or sketch-level shapes. The framework infers a candidate concrete syntax by recording the actions of domain experts on the modeling canvas as they interact with the tool to describe example systems in their domain. After the concrete syntax is identified, the concrete syntax and domain models are transformed into an abstract model before the abstract syntax is inferred. The inference process induces abstract syntax in the form of a metamodel by comparing abstract models of the concrete syntax and the domain models with patterns representing common metamodel idioms. Metamodel design patterns represent a set of training data that are passed on to an abstract syntax inference engine. The framework completes its process by inducing and associating relevant static semantics to the inferred metamodel.

3.1 Technical Challenges and Possible Solutions

In this section, we identify four major technical challenges that this research may encounter. We describe the challenges and our candidate solutions below.

- *Support for free-form or sketch-level shapes*: Due to the likely preference of domain experts to work in more unconstrained environments (i.e., whiteboard and paper-based sketches) and advances in pen-based input devices, the need to support free-form or sketch-level shapes is a new challenge for language inference research. Chen et al. [2] developed a software design tool named SUMLOW to capture and formalize sketch-level UML constructs. Ossher et al. [10] introduced the concept of flexible modeling tools, which use predefined free-form shapes for modeling pre-requirements. They built a prototype to combine the advantages of office automation tools and traditional modeling tools. In our approach, we support a combination of these two approaches. At first, the framework will provide a shape authoring tool. End-users draw shapes that represent their domain and store them as pre-defined shapes. After that, we will add free-form recognition functionality to the framework, which is similar to Chen's work. With this functionality, end-users can use any free-forms without defining them prior to use.
- *Concrete syntax definition*: Concrete syntax represents the visual representation of a DSML. It should be designed to avoid ambiguity and assist with readability by domain experts. Concrete syntax typically consists of a set of domain notations and their relations. To define concrete syntax automatically, the framework captures the domain expert's operations when they demonstrate a new concept using the notation of a specific domain on the modeling canvas. Concrete syntax is finally defined after a domain expert reviews the candidate concrete syntax and annotates with appropriate names. The core technology for capturing and inferring concrete syntax for a DSML is based on a by-demonstration technique, which is similar to earlier work on Program By Example (PbE) or Query By Example (QBE) [5][12]. Technically, a by-demonstration approach can be implemented by hooking events into a recording tool (e.g., a plug-in for Visio or Eclipse). A challenge of the recording process is maintaining an optimized sequence of actions (i.e., pruning unnecessary actions) while preserving the demonstrators intent. For example, a domain expert may draw a rectangle as a symbol for some domain-specific process and tweak the size and appearance. The intermediate interactions in the editor are of little concern to the inference process.
- *Abstract syntax inference*: Inferring abstract syntax is a special case of inductive learning and represents a core research area for this doctoral research. The goal of this step is to generate abstract syntax (i.e., metamodel for a DSML) by applying machine learning algorithms to concrete syntax and domain models, which are created in a previous by-demonstration step of the process. To induce abstract syntax, many approaches have been proposed, such as genetic programming [6], Bayesian learning [1], hill climbing [4], and graph-based approaches [8]. In our research, we plan to adopt a graph-based approach to leverage the benefits of graphs (e.g., analysis of structural and dynamic properties, design for verification, and model transformation). Thus, before

proceeding to abstract syntax inference, the domain models and concrete syntax will be rewritten as a (typed) graph grammar. Generally, to induce accurate abstract syntax, an inference engine requires a large set of training data (or model instances in our case). Presumably, it is not possible to provide a large set of model instances because a demonstration-based approach will ask domain experts to construct domain models manually, which can be a mundane and error-prone task. Thus, it is challenging to predict all possible cases of input for the inference process. To resolve this issue, we plan to investigate the notion of metamodel design patterns, which describe a set of patterns that are commonly used in metamodel design. Accordingly, the abstract syntax will be induced by searching and composing a set of maximum-likelihood patterns. The metamodel design patterns approach will help to infer accurate abstract syntax in the presence of insufficient training data.

- *Semantics inference*: Designing and implementing semantic inference are other core areas of the research. This will be the most challenging area of the research because specifying semantics in a formal way is difficult even for language development experts. For specifying modeling language semantics, several methods have been proposed, such as OCL [7] and Attribute Grammars [11]. The research goals of semantic inference of modeling languages will focus on (1) devising efficient and accurate algorithms for semantics inference, and (2) finding an appropriate method to associate the inferred semantics with the inferred abstract syntax, with representation of both abstract syntax and semantics in a (typed) graph.

4. Current status and Future Work

A prototype has been developed to understand the feasibility of the approach. The prototype was developed with Microsoft Visio because Visio provides various predefined shapes that support the creation and definition of user-specific shapes. The prototype was implemented with an understanding that a Visio template has a correspondence to the metamodel of a DSML. The prototype implementation focused on (1) the identification of concrete syntax, (2) creation of a Visio template, and (3) the identification of rules between shapes as a part of an initial static semantics.

Although we believe that the prototype demonstrates the feasibility of a "by-demonstration" approach to DSML creation, we will also develop our framework for other development platforms (e.g., Eclipse). The new framework will be developed according to the following schedule.

- Phase 1: The initial phase will focus on a preprocessor for the framework. During Phase 1, we will develop a by-demonstration technique on the new development platform and devise efficient algorithms for capturing concrete syntax and graph transformation.
- Phase 2. An abstract syntax inference engine will be implemented and compared to verify our approach in terms of accuracy and performance.
- Phase 3. Finally, semantic inference will be designed and implemented. At this phase, we will try to design efficient algorithm for inducing semantics as well as their association with abstract syntax.

Acknowledgments

This work is supported by NSF CAREER award CCF-1052616.

References

[1] Chen, S. F. 1995. Bayesian grammar induction for language modeling. *In Proceedings of the 33rd annual meeting on Association for Computational Linguistics (ACL '95)*, Stroudsburg, PA, USA, 228-235.

[2] Chen, Q., Grundy, J., & Hosking, J. 2008. SUMLOW: early design-stage sketching of UML diagrams on an E-whiteboard. *Software Practice and Experience*, vol. 38, no. 9, Jul. 2008, 961-994.

[3] Cho, H., Sun, Y., Gray, J., & White, J. 2011. Key Challenges for Modeling Language Creation By Demonstration, *ICSE 2011 Workshop on Flexible Modeling Tools*, Honolulu HI, May 2011.

[4] Cook, C. M., Rosenfeld, A., & Aronson, A. R. 1976. Grammatical inference by Hill Climbing. *Information Science*. vol. 10, no. 2, 59-80.

[5] Cypher, A., Halbert, D. C., Kurlander, D., Lieberman, H., Maulsby, D., Myers, B. A., & Turransky, A. (Eds.). 1993. *Watch what I do: Programming by Demonstration*. MIT Press, Cambridge, MA, USA.

[6] Dupont, P. 1994. Regular Grammatical Inference from Positive and Negative Samples by Genetic Search: The GIG method. *Proceedings of the 2nd International Colloquium on Grammatical Inference and Applications*, ICGI'94, LNAI, vol. 862, 236-245.

[7] Evans, A. S. and Kend, S. 1999. Core Meta-modeling semantics of UML: The pUML approach. *In Proceedings of the Second International Conference on the Unified Modeling Language*, B. Rumpe and R. B. France, Eds., Lecture Notes in Computer Science, vol. 1723.

[8] Jonyer, I., Holder, L. B., Cook, D. J. 2004. MDL-based context-free graph grammar induction and applications. *International Journal on Artificial Intelligence Tools (IJAIT)*, vol. 13, no. 1, 65-79.

[9] Kelly, S. & Tolvanen. 2008. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Press.

[10] Ossher, H., Bellamy, R., Simmonds, I*., Amid, D., Anaby-Tavor*, A., Callery, M., Desmond, M., de Vries, J., Fisher, A., & Krasikov, S. 2010. Flexible modeling tools for pre-requirements analysis: conceptual architecture and research challenges*. Object-Oriented Programming Systems Languages and Applications (OOPSLA)*, Reno/Tahoe, NV, Oct. 2010, 848-864.

[11] Paakki, J. 1995. Attribute grammar paradigms-a high-level methodology in language implementation. *ACM Computing Surveys*, vol. 27, no. 2 (June 1995), 196-255.

[12] Zloof, M. M. 1975. Query-by-example: the invocation and definition of tables and forms. *In Proceedings of the 1st International Conference on Very Large Data Bases (VLDB '75)*. ACM, New York, NY, USA.