Key Challenges for Modeling Language Creation By Demonstration

Hyun Cho, Jeff Gray
Department of Computer Science
University of Alabama
Tuscaloosa, AL 35401
hcho7@crimson.ua.edu
grey@cs.ua.edu

Yu Sun
Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294
yusun@cis.uab.edu

Jules White
Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA 24060
julesw@vt.edu

ABSTRACT
Modeling languages are used by software engineers to design and document the structure and behavior of a system. Although many modeling languages have been proposed, Domain-Specific Modeling Languages (DSMLs) have emerged recently due to their simplicity and expressiveness for a particular domain. However, it is not easy for end-users to design and implement DSMLs because DSML development requires both domain knowledge and language development expertise. To help domain experts, who do not have language development expertise, in creating their own DSMLs, we propose the notion of “Modeling Language Creation By Demonstration,” which can infer a metamodel and associated semantics from example demonstrations. The goal of the approach is to assist domain experts in developing and using their own DSMLs that support the use of formal (or pre-defined) and free-form shapes for modeling concepts in a specific domain. In this position paper, we identify the key challenges in using a demonstration-based approach to define modeling languages.

Keywords
Domain-Specific Modeling (DSM), Flexible Modeling, Metamodeling, By-Demonstration

1. INTRODUCTION
Modeling languages have become an effective approach to address the increasing complexity of software systems by raising the level of abstraction from programming languages [14]. They are used to describe the system architecture, specify the structure and behavior of the system, and document the system. Some modeling tools are integrated seamlessly through forward and backward generation techniques, which link different phases of the software lifecycle. General-Purpose Modeling Languages (GPMLs) (e.g., UML) are one type of modeling language that are used for a wide variety of purposes across a broad range of domains. For example, GPMLs are used for business process modeling [5], software architecture design [7], and database design [9]. The major disadvantage of GPMLs is their size and complexity, which make it challenging for non computer scientists to learn and apply to projects that have a computational need. For example, 246 classes and 583 properties are used to define the UML 2.0 metamodel. In addition, GPMLs often require tailoring for a specific domain because they contain too many irrelevant constructs that are not used and may confuse non-specialists. Unlike GPMLs, Domain-Specific Modeling Languages (DSMLs) assist domain experts in focusing on the level of abstraction relevant to their problem space by providing notations and constructs tailored specifically to that domain.

Although DSMLs offer numerous advantages over GPMLs, there are several challenges and limitations that have emerged. Because DSML creation and development require both domain knowledge and modeling language development expertise (e.g., metamodeling experience), end-users and domain experts who are not computer scientists often find the traditional approach for creating DSMLs daunting due to the following challenges:

DSML Challenge 1: Domain models are often specified with office tools such as word processing or drawing/presentation tools, rather than formal metamodeling tools. Due to the increasing number of sketch-based input devices (e.g., tablets) and a general preference for hand-drawn sketches in capturing high-level requirements and software design, modeling tools will need to adjust to new forms of input. This is particularly challenging for demonstration-based language design because the initial sketch is rather unconstrained and needs to process a wide range of open notations for different domains. Recognizing new shapes that are more informally drawn in new environments and on new input devices is a challenge that will increase in need.

DSML Challenge 2: Modeling language creation requires familiarity with metamodeling environments. Domain experts who are not familiar with metamodeling often do not understand the deep implications of domain analysis in DSML implementation. The common practice of DSML development is to define a metamodel for the specific domain based on the captured notations. As the metamodel specifies the abstract syntax and static semantics for the domain, domain experts need to understand the implications of metamodeling before they develop their own DSML. The lack of language creation expertise, especially metamodeling expertise for modeling languages, may undermine the quality of a DSML implementation if developed by an end-user.

DSML Challenge 3: The captured visual notations tend to be informal and incomplete, often requiring multiple iterations to reach a final version of the DSML. However, the iterative process of creating a DSML is tedious, error-prone and time-consuming if done manually. Therefore, simplifying the DSML development process such that end-users and domain experts can create their own languages offers many advantages.
DSML Challenge 4: Specifying the semantics of a modeling language using formal techniques is often challenging even for computer scientists. The major focus of many by-demonstration approaches is on structural and syntactic issues, with little contribution toward mechanisms for describing semantics. In the DSML context, specifying the semantics of a modeling language is even challenging for computer scientists and language design experts. Enabling end-users to describe the semantics of their free-form language will require new innovations to separate the underlying formality needed from the level of abstraction expected by an end-user. This is perhaps one of the most challenging issues facing flexible modeling, and without progress in the area, the tools and associated languages will not be as effective in providing a complete automated solution for performing many tasks that are typical in other modeling contexts.

To address the expertise gap and repetitive tasks associated with DSML development, we have applied demonstration-based techniques to improve the creation and development of DSMLs. This idea stems from our recent work on Model-Transformation by Demonstration (MTBD) [15], in which end users describe specific transformation tasks by working with instance models of a particular DSML. The idea was motivated by existing “by-demonstration” approaches, which were first introduced by Smith in the context of Programming-By-Demonstration (PhD) in 1975 [4]. PbD aims to enable computers to learn and generate programs automatically by observing end-user interactions, as opposed to having a programmer manually implement a computational need. PbD showed how users who do not have programming experience can demonstrate how a program should behave by executing a series of concrete tasks. In PbD, the sequence of user operations is recorded, such that programs can be generated from the recorded information. Several applications have been proposed based on the concept of PbD [1][2].

Our focus is to apply the by-demonstration concepts to aid domain experts in the creation of DSMLs - we call this Modeling Language Creation By Demonstration (MLCBD). The specific concentration is on generating the metamodel and associated semantics automatically from a demonstration that is based on an end-user submitting concrete model examples. This idea requires a language creation tool that: 1) allows users to sketch domain models freely with visual notations (either formal notations such as UML or free-form shapes) and demonstrate the desired domain models; 2) automatically generates the metamodel and the final DSML environment by inferring and analyzing the recorded demonstration information. The result is that end-users are able to create the desired DSML without having technical skills related to language creation, as typified by the traditional DSML iterative design process.

The rest of the paper is structured as follows. Section 2 offers an overview of the approach. Section 3 lists the challenges that need to be addressed to move the area forward, and Section 4 offers concluding remarks.

2. OVERVIEW OF MLCBD

This section describes the key steps of our proposed approach for applying by-demonstration concepts across the modeling lifecycle. The overall process of MLCBD is depicted in Figure 1. To generate the metamodel for a DSML representing the user-demonstrated concepts, MLCBD requires three steps: capturing domain notations, annotating the captured notations, and inferring the metamodel and semantics.

Capturing domain notations is the first step of the approach, and all unique shapes that are used for designing the domain are captured through a by-demonstration technique, which records all user operations (i.e., creating and deleting shapes, and connecting shapes) and then captures new and unique shapes that are used for domain design (i.e., specific icons suggested by the end-user to represent domain concepts). Domain experts can draw domain models with pre-defined (or formalized) shapes or free-form shapes, which can be imported or drawn with input devices. This step targets Challenge 1 from the previous section.

After domain notations are captured, they are annotated by domain experts for metamodel and semantics generation. As the terms used for the annotation guide the metamodel definition, they should be general enough to represent the notion of the captured model elements clearly and completely. The metamodel of a DSML can be inferred from the captured and annotated notations. At this stage, the captured notations are formalized, and their relationships are defined in the inferred metamodel. These two steps can assist with Challenges 2 and 3 from Section 1 (i.e., the necessity of modeling language development expertise, and the formalization of the captured shapes).

Finally, the semantics of the modeling language are defined to specify static constraints, express dynamic behaviors, and verify the completeness and consistency of the model instances. The formal and explicit semantic specification avoids the semantic mismatch between design models and modeling languages of analysis tools. This prevents the instances of the DSML from being interpreted many different ways.

Figure 1. Overall Development Process of MLCBD

3. CHALLENGES OF MLCBD

The main goal of MLCBD is to allow domain experts, who do not have language development expertise, to design domain models with free-form and/or formal shapes. This enables the inference of the metamodel and semantics from example instances of domain models. The automation of DSML development with a by-demonstration approach requires deep investigation into the following issues:

- Capturing and formalizing the free-form shapes. Due to the advancement of input devices, more touch-based input devices may be used by domain experts to sketch domain models with free-form shapes on the computer, in a similar fashion as with
pen and paper, or whiteboard sketches. Thus, the approach needs to capture and formalize the free-form shapes. Many researchers have already proposed several approaches to capture and formalize free-form shapes, but their approaches can only be applicable to specific contexts. For example, Chen et al. [3] demonstrated capturing and formalizing sketch-level UML diagrams. Plimmer and Apperley [13] proposed a computer-aided sketch approach for capturing a preliminary UI design in Visual Basic. Thus, to provide more flexibility for using free-form shapes, the proposed approach needs to employ techniques and algorithms for capturing and formalizing free-form shapes that are not bounded to a specific context.

- Capturing and formalizing a metamodel from model instances. In programming languages, syntax defines the formal relations between the language constructs, and deals with the form and structure of those constructs. Similar to programming languages and their grammar-based definitions, a metamodel is used to define the syntax of a modeling language to constrain the structure of the modeling elements. Our proposed approach generates a metamodel from example instances through inference. The use of inference results in several issues, such as the selection of effective inference algorithms, and the selection of model instances. This work is similar to previous work on metamodel inference that recovered a metamodel from lost instances [8].

Normally, the accuracy of an inference task can be improved if the training sample is well designed, and the inference engine is trained with a large sample of instances. However, preparing a large set of model instances for demonstration is not practical at the initial domain analysis stage because the desired notations and core concepts are sometimes immature. Thus, finding an effective algorithm that can infer an accurate metamodel from a small set of model instances is a critical factor for success of by-demonstration approaches.

Model instance selection (or model selection) [9] is another important factor for metamodel inference. Model instance selection refers to selecting the examples that would collectively provide the best opportunity for inferring a metamodel. Thus, model instance selection has an important impact on the accuracy of metamodel inference. However, model instance selection has received little attention in the literature, especially selecting model instances for metamodel inference.

- Capturing and formalizing semantics from the model instances. Semantics plays an important part in programming language specification by defining the meaning of syntactically valid strings. In modeling languages, semantics is used to specify static constraints, express dynamic behaviors, and verify the completeness and consistency of the model instances, as well as the model properties (e.g., concurrency, communication, temporal and other physical properties). If semantics are not defined precisely, the negative result is that the model instances can be interpreted differently by various users [12] and a correct usage of the modeling language cannot be ensured. In DSMLs, semantics is often defined by transforming abstract syntax, which is represented using a metamodel, into a well-known semantic domain (e.g., state machines) through semantic mapping [6]. Thus, identifying and defining the semantic domain and semantic mapping from model instances are crucial for verifying the correctness of the model and its properties.

- Capturing and formalizing the evolution of a metamodel and associated semantics from model instance changes. After a DSML is created by following the development process as shown in Figure 1, domain experts have two artifacts: domain model instances and the metamodel to which they conform, in addition to the semantics of the modeling language. Normally, the metamodel and semantics are the first-artifacts of modification if a modeling language needs to evolve. However, this is challenging to address when domain experts do not have language development expertise. Thus, the evolution of a metamodel and associated semantics of the modeling language should be performed using a similar process as the initial DSML definition. Following the same process for metamodel and semantics evolution may reveal additional challenges. If metamodel and associated semantics are evolved in order to support new types of shapes or semantics, the issues of the metamodel evolution and semantics may simply be resolved by regarding the evolution as a special case of MLCBD. However, although this resolution is simple, it may require the same or more time and effort to infer the metamodel and semantics. Thus, effective processes or algorithms should be devised that evolve the metamodel and semantics only for the delta of two model instances, existing model instances and model instances that have a new modeling notation.

On the contrary, if a metamodel and the associated semantics need to evolve to support the changes of existing notations, determination of intent or purpose of the change can be more challenging. Changes may result from adding new shapes and/or relations, or by modifying existing notations for the purpose of changing the metamodel and/or semantics. If there is no guided information for the intent of a change, then the process, as shown in Figure 1, will evolve the metamodel and associated semantics by dealing with changes as adding the new shapes and/or relations. Thus, even if end users change the notation in order to modify the existing metamodel and associated semantics, the metamodel and associated semantics are evolved by adding new information continually. Consequently, the metamodel and the language’s associated semantics will have different language concepts, constraints, and behavior. Thus, an algorithm or technique that identifies the change intent is needed to assist with metamodel evolution.

4. CONCLUSION
In this position paper, we presented the concept of demonstration-based DSML creation and evolution as well as the challenges of our approach as it relates to the vision of flexible modeling tools. MLCBD allows domain experts, who do not have language development expertise, to create DSMLs by capturing and formalizing their intentions and abstractions that they use when performing informal free-form modeling. We identified several issues, which we believe are critical to the area of flexible modeling. Our position is to suggest the identified issues as topics for discussion at the workshop.

5. ACKNOWLEDGMENTS
This work is supported by NSF CAREER award CCF-1052616.
6. REFERENCES


