2012 19th Asia-Pacific Software Engineering Conference

Syntax Map: A Modeling Language for Capturing Requirements of Graphical DSML

Hyun Cho, Jeff Gray, and Eugene Syriani
Department of Computer Science
University of Alabama
Tuscaloosa, AL USA
hcho7@crimson.ua.edu, {gray, esyriani}@cs.ua.edu

***Abstract*—Domain-specific modeling languages (DSMLs) are designed to model particular domains of interest using graphical, textual, or mixed syntax. Although most metamodeling tools offer an environment that automates several language development tasks, there is still a lack of support for aiding the domain expert to capture the requirements of a graphical DSML. We introduce the Syntax Map as an approach to address the challenges of graphical DSML requirements specification. It is designed to aid domain experts in describing a first-class graphical DSML requirement using a set of graphical notations. In addition, the Syntax Map can be used to generate a metamodel by means of model transformations. This short paper motivates the need for the Syntax Map, provides several brief examples, and discusses lessons learned in our investigation of this approach.**

Keywords; Domain-Specific Modeling Languages, General-Purpose Modeling Languages, Abstract Syntax, Concrete Syntax, Semantics, Requirements Specification

I. INTRODUCTION

The emergence of Model-Driven Engineering (MDE) and General-Purpose Modeling languages (GPMLs), such as the UML, have assisted software engineers in raising the level of abstraction in their designs. In addition, software engineers are able to generate implementations automatically from models by combining model transformation and source code generation techniques. Although GPMLs offer several benefits (e.g., productivity improvement, better portability, and quality improvement), one of the drawbacks of using a GPML is the investment of time and effort needed to learn a large set of GPML constructs, which may not be relevant or needed for a specific problem domain and may be too challenging to use for end-users who are not computer scientists.

As an alternative to GPMLs, Domain-Specific Modeling Languages (DSMLs) were introduced to satisfy specific modeling, scripting, and testing needs and allow end-users and domain experts to focus on abstractions that best fit the purpose of the modeling task. DSMLs assist in raising the level of abstraction and often are designed to provide a light-weight syntax (or notation), which is closer to the problem space and describe the notion of the domain precisely. Several different DSMLs have been developed for specific domain needs such as mobile health monitoring [3], real-time software design [8], IT management [9], and mobile phone application tests [12].

To develop a DSML, at least two experts (i.e., domain and language development experts) collaborate, as shown in Figure 1. The role of the domain expert(s) is to identify the requirements of the DSML and confirm whether the developed DSML meets the requirements for a new language describing the domain. After the requirements of the DSML are described by a domain expert, language development experts analyze the requirements and design a DSML development strategy. In addition, language development experts need to determine which syntax, abstract or concrete, is developed first. Generally, the concrete syntax of a DSML is explored before developing a DSML. This is often the case because domain experts may use symbols, which represent the idioms of discourse and jargon of specific problem domains, to document and/or share the notions of the domain with the language developers. Thus, many DSMLs may begin by designing concrete syntax and then extracting the essence of the language by designing the abstract syntax. Abstract syntax-driven DSML development may be preferable if the domain expert is not clear on some new aspects of the domain. This situation often happens when the domain is relatively new and subject to change due to external forces (e.g., new requirements placed on the domain). Thus, both domain and language development experts often work together to concretize the key concepts of the domain and characterize the abstractions required in the domain, with late elaboration of the abstract syntax [15].

A DSML is developed by performing a set of complex tasks iteratively and incrementally. This is challenging because few experts have both domain knowledge and language development expertise, thus the frequent need for interaction between both expert roles. DSML development may face other challenges, such as supporting sketch-level shapes that are provided as examples from the domain experts, as well as the challenges in specifying the semantics of the DSML formally [7]. These two challenges are often faced by the language development experts. Domain experts may face other types of challenges as they try to describe the information needed to capture the essence of a language for their domain. For example, there is no appropriate guideline for describing the requirements of a DSML and there is no clear systematic way to verify the developed DSML between the two expert roles.

To address the challenges in DSML development, we introduce in this short paper a new language-capture DSML that assists in specifying the requirements of another DSML using graphical symbols. The goal of the language-capture

1530-1362/12 $26.00 © 2012 IEEE
DOI 10.1109/APSEC.2012.20

DSML is to assist domain experts in the description of syntax. In particular, our new language focuses on describing the elements of the syntax and the structure of syntax elements, such as flow, and then generates the desired modeling environment for the new DSML for the domain of interest.

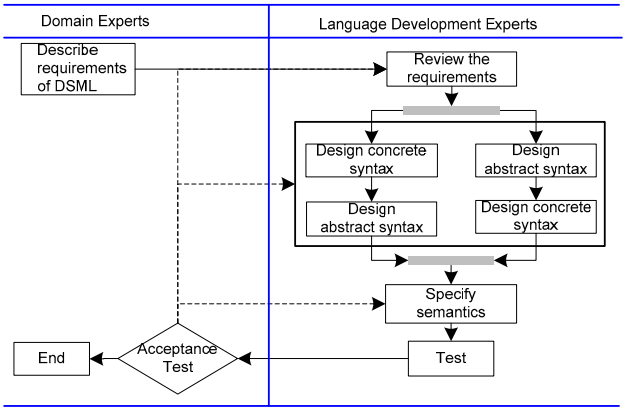


Figure 1. DSML Development Process

II. DSML Requirements Capture Using Syntax Maps

A language typically consists of three components: abstract syntax, concrete syntax, and semantics. Semantics define the meaning of the language and can be specified using one or more formal methods (e.g. denotational/translational semantics, axiomatic semantics, operational semantics, or attribute grammars). Concrete syntax is used to represent the surface-level notion of a language. In programming languages, the concrete syntax is often textual and described using Backus-Naur Form (BNF). In modeling languages, the concrete syntax may be textual, graphical, or mixed. Abstract syntax describes the notion of a language, the relationships between the notions, and the well-formedness rules, which specify how the concepts can be combined. Generally, the abstract syntax is defined using an abstract syntax tree for programming languages and a metamodel for modeling languages.

To develop a new graphical DSML, the requirements of the DSML can be described using three language components: the abstract syntax, a mapping relationship between the abstract syntax and concrete syntax, and semantics. However, it is not easy to describe the requirements of a DSML completely and precisely, and domain experts may suffer from the lack of an appropriate method, guidance, or tool support for capturing the requirements of a DSML. We have investigated an approach to address these issues and have developed the idea of a Syntax Map, which assists in capturing the requirements of a new graphical DSML. The Syntax Map's focus is to assist domain experts, who may have little experience or knowledge about development of graphical modeling languages, in describing the requirements of their own graphical DSML. The Syntax Map captures elements of syntax and the relationship (or structures) between the elements.

In order to describe the requirements of a graphical DSML using the Syntax Map, the following steps are observed by a domain expert:

- *Define classifiers*. A classifier is an abstract metaclass that is used to describe a set of instances that have common features. The classifier is used for defining a namespace, type, and redefinable elements.
- *Add attributes to each classifier if necessary*. A classifier may have structural and/or behavioral characteristics in a specific domain. An attribute is one way to represent such characteristics of a classifier.
- *Define relationships and relate with classifiers*. Relationships play important roles in modeling languages because they describe structural relationships between classifiers.
- *Specify additional information about the relationship* (e.g., relationship type, cardinality, and directional information). This can be added to provide semantic information about the relationship after it is assigned to classifiers.
- *Link concrete syntax*. If the concrete syntax of a graphical DSML has been identified already, the concrete syntax can be associated with the corresponding abstract syntax elements.

To describe the requirements of a graphical DSML, the Syntax Map provides eight symbols, as shown in Table 1. The Syntax Map must contain at least two symbols, *Start* and *End*, which represent the start and end of the requirement description. These two symbols are represented by a circle (*Start*) and bar (*End*). A *Classifier* is denoted as a rectangle, and at least one classifier should be linked to the *Start* and *End* symbols. If necessary, attributes can be defined to a *Classifier*. When defining attributes, the type of the attribute should be provided. Primitive data types (e.g., String and Number) are provided as built-in *Data Type*s. In addition, domain experts can define composite data types by combining primitive data types. To represent relationships between *Classifiers*, a rounded rectangle is placed between *Classifiers*.

To link between the Syntax Map symbols, two types of link symbols are provided. One is a line and the other is an arrow-headed line. If an arrow-headed line is used to link between *Classifiers* and *Relationships*, it implies that there is directional dependency or information flow between the two. If a line is used, domain experts need to assign whether the link is directive or not. A directive link indicates that information can flow in both directions between *Classifiers* and *Relationships*. After domain experts connect *Classifiers* to *Relationships*, they need to specify the attributes of the relationship. The relationship has four attributes to characterize the relationship between *Classifiers*: Type, Directional, and Cardinality of source and target. Attribute *Type* defines four types of relationships by default: association, aggregation, composition, and inheritance. If domain experts assign one of these types as the name of the relationship, the Syntax Map automatically sets the *Type* attribute corresponding to the assigned name. If domain experts assign a name different from the name in the *Type* attribute, they need to assign the type of relationship. Attribute *Directional* specifies the direction of information flow or dependency. The semantics of the direction attribute is determined by combining the types of links that connect

between *Classifiers* and *Relationships*. The *Cardinality of Source and Target* describes the number of possible *Classifiers* linked by the relationship. Finally, the symbol *Mapped* is used to map the abstract syntax to the concrete syntax. *Mapped* has two attributes: *Constraint* and *Rendering*. The former is used optionally when an abstract syntax needs to be mapped to multiple different types of concrete syntax. The mapping constraint is specified informally using natural language because the Syntax Map is used to capture requirements of a graphical DSML in the early development stage. The *Rendering* attribute specifies a path where a graphic symbol is stored. When domain experts specify the path attribute in *Rendering,* the Syntax Map reads the file and displays the concrete syntax in the symbol *Mapped*.

Table 1. Symbols of the Syntax Map

Symbol	Name	Description
	Start	Indicate start of the Syntax Map.
	End	Indicate end of the Syntax Map.
	Classifier	Represent classifiers (or entity) in abstract syntax.
	Relationship	Represent relationship between Classifier.
	Attribute	Associate attribute to classifier. If necessary, relationship can have an attribute.
	Data Type	Represent attribute type. Basic data type (e.g., String and Numbers) is provided.
Mapped	Mapped	Mapping between abstract syntax and concrete syntax.
	Link	Link between the Syntax Map elements.

An example Syntax Map is shown in Figure 3. It captures the requirements of an abstract and concrete syntax of a state transition diagram for the Deterministic Finite Automation (DFA). The DFA computes whether the input binary numbers are multiples of 3. If a binary number 11 is entered, the DFA jumps to S_1 for the first 1 and then returns to S_0, which is the accepting state that represents the input binary number is multiples of 3. As shown in Figure 3, the FSM uses three unique symbols, *Accepting State*, *State* and *Transition*, to model the DFA for checking multiples of three.

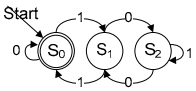
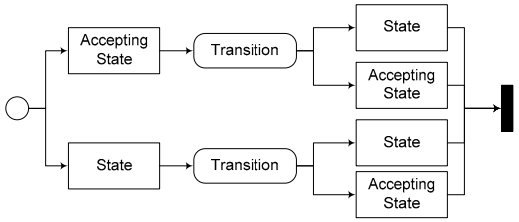
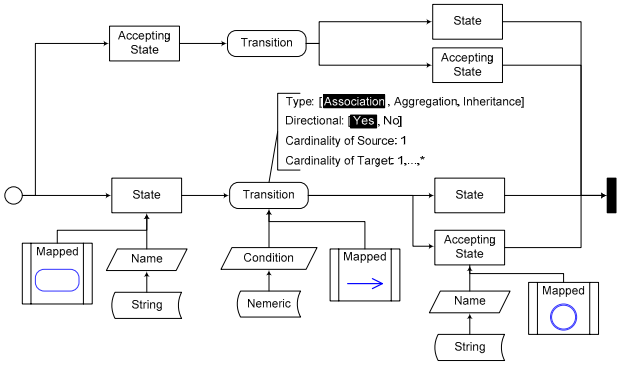


Figure 2. DFA for Checking the Multiples of 3

To describe the requirements of the DFA with the Syntax Map, domain experts first create the skeleton scenarios by placing classifiers and relationships between the Syntax Map symbol *Start* and *End* according to their usage scenarios. For instance, if the current state is the *Accepting State* and a transition is triggered, the next state is determined by the transition condition and can be either *Accepting State* or *State*. In addition, State can transition to either *Accepting State* or *State*. The skeleton of the DFA is shown in Figure 3(a). The upper part describes the state transition scenario from the *Accepting State*, and the lower part is for the *State*. After the skeleton of a Syntax Map is created, domain experts add additional information to each Syntax Map modeling elements if necessary. Classifier *Accepting State* and *State,* even though they are mapped to different symbols, and have an attribute, *Name*, with type String to denote instance name. If the classifiers (or relationships) have the same attributes across the Syntax Map, domain experts specify attributes to only one classifier. Then, the rest of the classifiers will share the attributes. For example, although another classifier *State*, which is connected with symbol *End,* is modeled without any attributes, attribute *Name* will be associated automatically with classifier *State*, which is connected to symbol *Start*. The complete Syntax Map is shown in Figure 3(b).



(a) The Skeleton of the Syntax Map



(b) The Complete Syntax Map for the DFA

Figure 3. Example of the Syntax Map for DFA

III. METAMODEL SYNTHESIS FROM SYNTAX MAP MODEL

Since the Syntax Map is a modeled language, model transformation and code generation can be applied on Syntax Map models. In fact, after the Syntax Map is created, we can generate the metamodel of the graphical DSML for a target development environment. To generate the metamodel for a target development, the Syntax Map is first transformed into an internal graph representation. Graph representations are widely used in modeling-driven engineering, especially in model transformation because a graph can assist in the creation, editing, analysis of a model from the other model through transformation [1][4]. From the graph and its transformation, one can generate other types of output by applying production/replacement rules through graph rewriting [5][13]. To transform the Syntax Map into the graph representation, classifiers, relationships, and attributes are mapped onto nodes. The other elements are defined as attributes of each node. After the Syntax Map is transformed into the graph representation, the corresponding transformation engine (e.g., KM3 Transformation, Class Diagram Transformation, and GME transformation) generates the syntax as defined by metamodels for each tooling platform such as KM3, Class diagram, or GME.

IV. Related Work

Use Case Maps (UCMs) were proposed by Amyot et al. to provide "a notation to aid humans in expressing and reasoning about large-grained behavior patterns in systems [6]." The basic idea of UCMs is to capture the requirements of a system by introducing a scenario-based software engineering technique, which can describe causal relationships between responsibilities of one or more use cases [18]." Due to the nature of a scenario, UCMs are useful in capturing informal (or functional) requirements, and validate logical errors in requirements. In addition, UCMs can be transformed into LOTOS [2] or SDL [10]. Due to simplicity, UCMs have been applied successfully for capturing requirements of software systems, documenting standards, and evaluating the alternates of architecture. The main ideas of UCMs, especially scenario-based requirement capturing and graphical representation affect to develop our approach. To develop a graphical DSML using the Syntax Map, domain users should define concrete syntax first and then design abstract syntax. In order to design abstract syntax, domain users may examine possible scenarios of each element of concrete syntax, especially focusing on the possible relationships between concrete syntax and attributes of each element of concrete syntax. Thus, similar to UCMs, the Syntax Map helps domain users to design abstract syntax by considering the usages (or relationships) of each concrete syntax element using graphical notations.

A Syntax Graph [14] is represented in a directed graph and also called a syntax diagram, or syntax chart. The syntax graph was first used to document the syntax of ALGOL 60 in a condensed form for reference during compiler development. The syntax graph is similar to a flow chart, which can represent flow. It is designed to be able to define language constructs as metalinguistic formulas, which consist of metalinguistic variables and basic symbols. The syntax graph helps check the syntax of a program and can be used to train engineers. As the syntax graph is useful to illustrate syntax structure and/or data structure visually, it is used to illustrate the syntax for several different contests, such as SQL [16][17] and web services [18]. Similar to the syntax graph, the Syntax Map can also be used to document abstract syntax or illustrate syntax structure. Besides these, the Syntax Map can be used as an input in order to develop a graphical DSML through transformation.

V. Conclusion and Future Work

In this paper, we introduced the idea of a Syntax Map for capturing the requirements of a graphical DSML. The requirements of a graphical DSML, as described in the Syntax Map, can be used to generate the syntax for a target DSML development platform (e.g., KM3, GME) by applying model transformations (or source code generation). We also presented a brief guideline (or procedure) to describe the requirements of a graphical DSML using the Syntax Map. From our experience, we believe that the Syntax Map could reduce DSML development effort and time by eliminating duplicated efforts, especially specification and analysis of DSML requirements. We plan to conduct experimental user studies to confirm this expected benefit. The Syntax map provides an opportunity to automate the generation of abstract syntax definition from captured DSML requirements by applying model transformations.

Acknowledgment

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

References

[1] M. Andries, G. Engels, A. Habel, B. Hoffmann, H-J Kreowski, S. Kuske, D. Plump, A. Schürr, & G. Taentzer, "Graph transformation for specification and programming," Journal of Science of Computer Programming., vol. 34, no. 1, pp. 1-54.

[2] D. Amyot, & L. Logrippo, "Use Case Maps and Lotos for the prototyping and validation of a mobile group call system", Computer Communications, vol. 23, no. 12, Jul. 2000, pp. 1135-1157.

[3] F. T. Balagtas-Fernandez,.& H. Hussmann"Applying Domain-Specific Modeling to Mobile Health Monitoring Applications," Sixth International Conference on Information Technology: New Generations, Apr. 2009, pp.1682-1683.

[4] J. Bézivin, "Model Driven Engineering: An Emerging Technical Space," Generative and Transformational Techniques in Software Engineering 2005 (GTTSE 2005), July 2005, Braga, Portugal, 2005, pp. 36-64.

[5] D. Blostein, & A. Schürr, "Computing with graphs and graph transformations", Software: Practice and Experience, vol. 29, no. 3, 1999, pp. 197-217.

[6] R. J. A. Buhr and R. S. Casselman, Use Case Maps for Object-Oriented Systems, Prentice Hall, 1995.

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

[8] A. Cortier , J. P. Bodeveix , M. Filali , G. Garcia , E. Mor , M. Pantel , A. Rugina , M. Strecker , & J. P. Talpin, "Synoptic: A Domain-Specific Modeling Language for Space On-board Application Software", in Synthesis of Embedded Software, Springer, 2010, pp. 79-119.

[9] U. Frank, D. Heise, H. Kattenstroth, D. Fergusona, E. Hadarb, & M. Waschkec, "ITML: A Domain Specific Modeling Language for Supporting Business Driven IT Management", In Proceedings of the 9th Workshop on Domain-Specific Modeling (DSM '09), Orlando, FL, Oct. 2009.

[10] Y. He, D. Amyot, & A. W. Williams, "Synthesizing SDL from Use Case Maps: An Experiment", In Proceedings of SDL Forum'2003., Stuttgart, Germany, Jul. 2003, pp.117-136.

[11] M. Mernik, J. Heering, & A. M. Sloane, "When and How to Develop Domain-Specific Languages", ACM Computing Surveys, vol. 37, no. 4, , Dec. 2005, pp. 316-344.

[12] Y. Ridene, N. Belloir, F. Barbier, N. Couture, "A DSML for mobile phone applications testing", In Proceedings of the 10th Workshop on Domain-Specific Modeling, Reno/Tahoe, NV, Oct. 2010.

[13] G. Rozenberg (ed.), Handbook on Graph Grammars and Computing by Graph Transformation: Foundations, vol.1-2. World Scientific, Singapore, 1997.

[14] W. Taylor, L. Turner, and R. Waychoff. "A syntactical chart of ALGOL 60", Commun. ACM, vol. 4, no. 9, Sep. 1961, p393.

[15] D. S. Wile, "Abstract Syntax from Concrete Syntax", In Proceedings of ICSE, pp. 472-480, ACM, Boston MA, 1997.

[16] IBM Database Fundamentals, http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=%2Fcom.ibm.db2.luw.sql.ref.doc%2Fdoc%2Fr0006726.html .

[17] Oracle Syntax Diagram, http://docs.oracle.com/cd/B28359_01/server.111/b28286/ap_syntx.htm#i624534.

[18] SharePoint 2010 REST Service Syntax Diagram, http://blogs.msdn.com/.b/sharepointpictures/archive/2011/03/30/sharepoint-2010-rest-service-syntax-diagram.aspx.

[19] Use Case Maps, http://www.usecasemaps.org.