Semi Automatic DSL Generation

Iris Reinhartz-Berger
Department of Management Information Systems, University of Haifa, Haifa 31905, Israel
iris@mis.haifa.ac.il

Zeev Tavor
Department of Computer Science, University of Haifa, Haifa 31905, Israel
ztavor@gmail.com

Arava Tsoury
Department of Management Information Systems, University of Haifa, Haifa 31905, Israel
aravab@mis.haifa.ac.il

Domain models capture the common knowledge gained while developing applications in the domain as well as the possible variability allowed among them. Hence, domain models may assist in the creation of valid applications in that domain, improving productivity and software quality and reducing the domain and development expertise needed. However, the creation of such domain models is not a trivial task: it requires expertise in the domain, reaching a very high level of abstraction, and providing flexible, yet formal, artifacts [1]. The field of domain engineering (also known as product line engineering) [4] aims at identifying, modeling, constructing, cataloging, and disseminating the commonalities and differences among applications in a specific domain. Several domain engineering methods have been proposed over the years, but most of them can be criticized as making the domain engineer the only responsible for the development of domain models and artifacts. Since domains may cover broad areas and are usually understood only during the development process, creating domain models can be a very demanding task. However, they may be beneficial in both creating (instantiating) new applications in the domain and validating them according to the domain rules. In this research, we aim at semi-automatically generalizing application models into domain models in order to create Domain Specific languages (DSL) [3]. Van Deursen et al. [7] define DSL as "a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused, and usually restricted to, a particular problem domain". Part of the domain knowledge that is needed for creating DSLs can be extracted from domain models.

For representing the domain knowledge, we use a domain engineering approach, called Application-based DOmain Modeling (ADOM) [5], which perceives that applications and domains require similar facilities for their modeling, design, and development, thus it enables specifying and constructing domain artifacts with regular application engineering techniques. The ADOM framework comprises of three layers: the application layer, which consists of models of particular systems; the domain layer, which consists of specifications of application families (i.e., domains) including their commonality and variability; and the language layer, which includes metamodels of modeling languages. This separation of ADOM into three layers enables its adaptation to different modeling languages. However, when adapting ADOM to a particular modeling language, it is used in both application and domain layers, so that the language layer imposes constraints on both the application and domain layers. Furthermore, the domain layer enforces constraints and provides guidelines for the models in the application layer.

The proposed approach consists of two steps: creating domain models from applications that have already been developed in the domain and generating DSLs from the created domain models.

Creating domain models: In this step, draft models of emerging domains are created from families of relevant applications [6]. These applications are being matched, integrated, and generalized into domain models. The matching phase includes three kinds of similarity measurements, calculated for each pair of elements. First, the linguistic similarity compares element names using hierarchies and semantic relationships embedded in the WordNet, a semantic lexicon for the English language [9]. Second, the dependee similarity takes into consideration the dependent elements of certain elements.

Dependent elements of an element *e* are all those model elements that the omission of *e* from the model implies their omission too. In the object-oriented paradigm, for example, we can assume, to some degree of confidence, that classes that exhibit similar attributes and methods are similar too and thus can be matched. Finally, relational similarity takes into consideration the relationships in which the compared elements participate (as sources or destinations). Two elements are considered matches if their general similarity measurement, which is a weighted calculation of their linguistic, dependee, and relational similarities, exceeds a certain threshold. After the matches are established, the application models are correspondingly generalized into draft domain models, expressed on a higher abstraction level: elements that are in the same similarity group (calculated using the elements' general similarity measurements) are generalized into the same domain element.

The domain models creation step refers to both structural and behavioral aspects of model matching and merging, enabling specification of the domain functionality and not just its structure. Sequence diagrams, for example, can be considered as the dependees of their elements, namely their combined fragments, messages, and lifelines (objects). Furthermore, the approach is not language specific and can be applied to a variety of modeling languages. However, for each language the main concepts and their dependencies and relationships should be defined first (once).

Generating DSL: Having the created domain models, we can automatically generate DSLs. DSLs, which have become popular in recent years partially due to their support for expressing solutions in the level of abstraction of the problem domain, aim at enhancing quality, productivity, reliability, maintainability, portability and reusability of implementations. The most noticeable approaches for implementing DSLs [7] are: interpretation or compilation, embedded languages or domain-specific libraries, and preprocessing or macro processing. We chose to apply in this research a domain-specific libraries approach, which is known as Domain Specific embedded Languages (DSeL), due to its maturity (through existing tools, such as [2]) and simplicity. However, as opposed to other DSL tools and approaches, we aim at guiding the domain variability and referring to its behavioral aspects. Identifying the commonality of a domain is important as it describes the important concepts and rules and, indeed, most DSL approaches support commonality generation. However, Gomma [1] stated that variability management is even more important, as this factor distinguishes domain analysis from "regular" reuse. Webber and Gomaa [8] identified four types of variability mechanisms: (1) parameterization, in which the application designer may change the values of attributes defined as parameters in the domain model, (2) information hiding, in which the application designer uses the same interface for defining similar components, (3) inheritance, in which the application designer may extend the interface of domain elements within a specific application and even override them, and (4) variation points definition, in which the application designer may create new variants and connect them to the variation points specified within the domain model. All these variability mechanisms will be supported by the suggested approach. Furthermore, similarly to the matching and generalization operations, the DSL generation will support the development of DSLs' behavioral aspects and not just structural ones.

For providing a proof of concept, we have implemented the first step of the approach (namely, the automatic creation of domain models) and run it for a domain of project management systems that included just four applications (see [6]). The application models were expressed in UML class and sequence diagrams and were specified by different advanced undergraduate Management Information Systems students (MIS). We further conducted an experiment in which the resulted domain model, along with an evaluation questionnaire, was given to a class of about 20 undergraduate MIS students at

the University of Haifa, Israel. This questionnaire included a paragraph describing the project management domain, the domain model as generated by the implemented algorithm, a domain model that was created by advanced MIS students who analyzed and reverse engineered open-source applications in the domain, and a domain model which was created by advanced MIS students who studied domain ontologies and reviewed the domain literature. The participants were required to map the core elements of the automatically created domain model, such as classes and messages, to their counterparts (if exist) in the two human-made domain models. In addition, they were required to evaluate the automatically created domain model according to different criteria, such as correctness, completeness, redundancy, and consistency. The results showed that above 98% of the participants mapped the classes correctly and above 70% of the participants succeeded in correctly mapping the objects and messages in the sequence diagrams. Moreover, the participants graded the correctness of the automatically created domain model as 3.9 (out of 5), its completeness – as 3.7, its redundancy – as 4.5 (i.e., the participants thought that the model did not include too many redundant elements), its consistency – as 4.0, and its overall quality grade – as 3.8. The main criticism of the participants regarding the draft model was that the names of some relevant elements were too abstract or not compatible with their expected roles in the domain. However, since the repository that was used in this experiment was very small (4 applications) and despite of it the participants managed to correctly map the model concepts to the domain terminology, this shortcoming is tolerable.

In the future, we plan to improve the implemented algorithm in general and the calculation of the different types of similarities in particular by using additional meta-information on the elements, such as data types, scopes, relationship types, relationship cardinality, and so on. We also plan to improve the variability specified in the automatically created domain models in order to support the modeling of variants and alternatives. Furthermore, we plan to implement the DSL generation step.

References:

1. Gomma, H., Designing Software Product Lines with UML, 2004.
2. Microsoft Domain-Specific Language (DSL) Tools, available at: http://www.microsoft.com/downloads/details.aspx?FamilyId=57A14CC6-C084-48DD-B401-1845013BF834&displaylang=en.
3. Mernik, M., Heering, J. and M.Sloane, A., When and how to develop Domain-Specific Languages. ACM Computing Surveys (CSUR), Volume 37, Issue 4 (December 2005), Pages: 316 - 344, 2005.
4. Prieto-Diaz, R. Domain Analysis and Software Systems Modeling. Los Alamitos, CA: IEEE Computer Society Press, 1991.
5. Reinhartz-Berger, I. and Sturm, A. Enhancing UML Models: A Domain Analysis Approach, Journal on Database Management (JDM) 19 (1), special issue on UML Topics, pp. 74-94, 2007.
6. Reinhartz-Berger, I. and Tavor, Z. SADoM - A Semi–Automated Domain Modeling Approach, available at: http://mis.hevra.haifa.ac.il/~iris/research/SADoM/.
7. Van Deursen, A., Klint, P. and Visser, J. Domain – Specific Languages: An Annotated Bibliography, SIGPLAN Notices journal , volume 35 number 6, pages 26-36, 2000,
8. Webber, D. L. and Gomaa, H., "Modeling variability in software product lines with variation point model", Science of Computer Programming, Vol. 53, pp. 305-331, 2004.
9. WordNet – a lexical database for the English language, available at: http://wordnet.princeton.edu.