Middle-East Journal of Scientific Research 19 (7): 908-912, 2014
ISSN 1990-9233

DOI: 10.5829/idosi.mejsr.2014.19.7.1469

An Adaptive Language Approach for Domain Analysis

D. Kerana Hanirex

Department of CSE,
Bharath University, Chennai-73, India

Abstract: Domain-specific languages (DSLs) are computer languages intended for problem solving in a specific domain. Ontology is a formal representation of a set of concepts from a particular domain and the relations between them. The proposed system aims at developing a system which provides multi programming paradigm as currently most of the programming languages are providing only single programming paradigm. So mostly software developers need to mix and match different paradigms, typically lead to impedance mismatch. The proposed system is developed using integration of ontology paradigm in a programming language which is already having multiple programming techniques like functional, object oriented, concurrency. The basic approach used in integration is metaprogramming, which will craft and a process of languages for creating, modifying, adapting, adjusting and otherwise transforming other program.

Key words: Domain-Specific Language · Domain analysis · Ontology

INTRODUCTION

Programming languages are used for human-computer interaction. Programming language can be divided into general-purpose languages (GPLs) and domain-specific languages (DSLs). GPLs, such as Java, C and C#, are designed to solve problems from any problem area. When developing new software, a decision must be made as to which type of programming language will be used: GPL or DSL. Reasons for using a DSL are as follows: easier programming, re-use of semantics and the easier verification and programmability for end-users. However, using a DSL also has its disadvantages, such as high development costs. The key is to answer the question of when to develop a DSL,the simplest answer to this question is: a DSL should be developed whenever it is necessary to solve a problem that belongs to a problem family and when we expect that in the future more problems from the same problem family will appear. DSL development consists of the following phases: decision, analysis, design, implementation, testing, deployment and maintenance.

While the implementation phase has attracted a lot of researchers, some of the DSL development phases are less known and are not as closely examined (e.g. analysis, design). Various methodologies for domain analysis have been developed. Examples of such methodologies include: DSSA (Domain Specific Software Architectures), FODA (Feature-Oriented Domain Analysis) and ODM (Organization Domain Modeling). Formal methodologies are not used due to complexity and the domain analysis is done informally. Even if the domain analysis is done with a formal methodology, there are not any clear guidelines on how the output from domain analysis can be used in a language design process. The outputs of domain analysis consist of domain-specific terminology, concepts, commonalities and variabilities. Variabilities would have been entries in the design of DSL, while terminology and concepts should be reflected in the DSL constructs and commonalities could be incorporated into the DSL execution environment. Although it is known where the outputs of the domain analysis should be used, there is a need for clear instructions on how to make good use of the information, which are retrieved during the analysis phase, in the design stage of the DSL.

we propose that domain analysis (classic domain analysis (CDA)) that can be performed with the use of existing techniques from other fields of computer science. A particularly suitable one is the use of ontologies. An ontology provides the vocabulary of a specialized domain. This vocabulary represents the domain objects, concepts and other entities. Ontologies

Corresponding Author: D. Kerana Hanirex, Department of CSE Bharath University, Chennai-73, India.

in the CDA have already been used. We propose that an ontology replace the CDA. They also investigated how ontologies contribute to the design of the language. Ontologies in connection with DSL are also used by other authors. The proposed solution of the first problem, the use of ontologies, has a significant effect on the second problem, related to CDA.

Software can be developed for a variety of purposes, the three most common being to meet specific needs of a specific client/business ie the case with custom software, to meet a perceived need of some set of potential users ie the the case with commercial and open source software, or for personal use for example a scientist may write software to automate a mundane task. Software environments are designed with a single programming paradigm, such as ontologies, functions, objects, or concurrency. So the solutions developed using this type of software environments will limit the representation and efficiency. So mixing and matching languages, platforms and paradigms is the typical technique used to improve the solution. Cross-mapping multiple paradigms and platforms will produce an impedance mismatch that increases a solution's complexity.

The term ontology has become popular in several fields of Informatics like Artificial Intelligence, Agent systems, Database or Web Technology. The term ontology in Computer Sciences ontology stands for a formal explicit specification of a shared conceptualization or it can also be defined aso the subject of existence or as an description of the concepts and relationships that can exist for an agent or a community of agents.

Ontologies for the Software Engineering domain is applicable to software development project which is not just concerned with a specific application.One principal goal of ontology in software engineering domain is to extend the idea of reuse from the implementation to the modelling level. I.e. instead of building systems from ready-made components which are "plugged together" like hardware modules, ontologies are reusable model components from which particular implementations can be derived for specific platforms, according to specific interfaces and constraints of application development projects. One of the most common tasks in software design is to create an object model of the software application. In designing domain specific Software, the designer has option of using knowledge about the domain, in addition to user requirements and principles design. The method leads to a specific architecture of the software component with models in two orthogonal dimensions. One dimension represents the categories originating in the domain ontology and the other dimension represents the functional concerns that originate from user requirements. For the development of ontological based domain model one has to determine the domain, choose an ontology for the domain, address specific user requirements and finally has to construct the application model by configuring the objects that originate from ontological categories with aspects that originate from specific user requirements. The selected ontology should contain the minimal set of concepts that completely covers the domain.

Over the past few years, people who surf the World Wide Web (WWW)got quite used to possibility to reuse content from other sites within the website they are currently browsing. This allows Web users to manage their photos, contacts, or personal diaries in dedicated websites specialized for these media types. However, how these different websites exchange data contradicts the design principles of the WWW in various ways, ranging from a lack of standardized protocols to bypassing important features of the Web architecture: Web applications often maintain a state where they should not, often do not support content negotiation, or even work against caching mechanisms, e.g., by modifying resources on HTTP GET requests. In contrast, the linked-data community builds upon Semantic-Web standards like resource description framework (RDF) and SPARQL protocol and RDF query language (SPARQL) to achieve a Web of Data that is completely based on standards and capable of even more advanced interactions between independent Web-based information systems. Unfortunately, it turned out that the creation of actual software processing linked data is not trivial and requires the software developer to understand quite some of the Web standards that are involved. While such knowledge can be expected from members of the Semantic-Web and linked-data communities, it should not be required for the average software engineer writing software for the WWW.

DSL development with the presented framework is easier and thus cheaper because the framework is able to execute a large part of the transformation independently of the DSL engineer. The involvement of the engineer is required in the steps where the framework has to "understand" the meanings of the concepts for which the work is being done. Another advantage of the framework is the ability to quickly test and verify different solutions; developing different grammars. Of course, the framework does require some knowledge before it can be effectively used. Familiarization with ontologies ensures a much

easier understanding of the framework, the developed framework is appropriate for use in education as well as industry. Students will find it particularly useful when they study the construction of grammars, as the framework autonomously accomplishes several steps and leads them to the correct path. Industrial use would be the primary goal, as it would be leveraged to speed up the process of DSL development as well as lowering the costs. Ontologies have been used by other authors in the DSL field]. A survey of literature has not given any reference where research was aimed at the development of DSLs from ontologies. Also, our framework cannot be compared to various tools for DSL creation (e.g., EMFText, Xtext, MetaEdit+, GME), where DSL is created from a language model (meta-model). All these tools require that domain concepts and relationships among them are already known. Hence, these tools do not support a domain analysis phase, which is usually done *adhoc*.

Previous Research: Many domain-specific languages, that try to bring feasible alternatives for existing solutions while simplifying programming work, have been stared in [1]. In this work, we present an experiment, which was carried out to compare such a domain-specific language with a comparable application library. The experiment was conducted with 10 programmers, who have answered a questionnaire on both implementation approaches. The questionnaire is more than 100 pages long. For a domain-specific language and the application library, the same problem domain has been used-construction of graphical user interfaces. In terms of a domain-specific language, XAML has been used and C# Forms for the application library. A cognitive dimension framework has been used for a comparison between XAML and C# Forms.

The requirement for Rational framework for metaprogramming as consistent programming and metaprogramming languages, familiar metaprogramming construct, familiar code, Compile-time objects as first-class citizens, Closed form is proposed in [2]. Metaprogramming manipulates symbols representing various complex operations rather than plain data elements. A more powerful type of metaprogramming involves extending existing languages or creating new ones. Everyday metaprogramming involves on-the-fly code production. A language that is build with metaprogramming feature should have capability to manipulate functions in the same way as that of data.

Modeling spaces is proposed in [3] which will help software practitioner to understand modeling.

A mechanism to bridge both metamodelling and ontologies in order to identify ways in which they can be made compatible and linked in such a way as to benefit both communities and create a contribution to a coherent underpinning theory for software engineering is dicussed in [4].

Semantic Web will enable machines to comprehend semantic documents and data is described in [5]. Shared understanding is achieved by exchanging ontologies. The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users. The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation [6].

Hypotheses: For developing the DSLs of the following are the hypotheses,

Decision
Analysis
Design
Implementation
Deployment

MATERIALS AND METHODS

DSL development is not a simple sequential process of (positive) decision followed by domain analysis, followed by DSL design and so on [7].

Hypotheses Testing

DECISION

The decision phase corresponds to the “when”-part of DSL development. Deciding in favor of a new DSL is usually not easy. The investment in DSL development (including deployment) has to pay for itself by more economical software development and/or maintenance later on. In practice, short-term considerations and lack of expertise may easily cause indefinite postponement of the decision. To aid in the decision process, we identify a number of decision patterns. These are common situations that potential developers find themselves in that might motivate the use of DSLs [8].

Analysis: In the analysis phase of DSL development, the problem domain is identified and domain knowledge is gathered. This requires input from domain experts and/or the availability of documents or code from which domain knowledge can be obtained. Most of the time, domain analysis is done informally, but sometimes domain analysis methodologies such as DARE (Domain Analysis and Reuse Environment), DSSA (Domain-Specific Software Architectures), FODA (Feature-Oriented Domain Analysis), or ODM (Organization Domain Modeling) are used [9-13].

Design: Approaches to DSL design can be characterised along two orthogonal dimensions: the relationship between the DSL and existing languages and the formal nature of the design description.

The easiest way to design a DSL is to base it on an existing language. We identify three patterns of design based on an existing language. First, we can *piggyback* domain-specific features on part of an existing language. A related approach restricts the existing language to provide a *specialisation* targeted at the problem domain. Both of these approaches are often used where a notation is already widely known.

Once the relationship to existing languages has been determined, a DSL designer must turn to specifying the design before implementation. We distinguish between *informal* and *formal* designs. In an informal design the specification is usually in some form of natural language probably including a set of illustrative DSL programs [14-18].

Implementation: When an (executable) DSL is designed, the most suitable implementation approach should be chosen. The implementation patterns we have identified. First, it should be noted that interpretation and compilation are as relevant for DSLs as for GPLs, even though the special character of DSLs often makes them amenable to other, more efficient, implementation methods, such as preprocessing and embedding. Development cost is not directly related to implementation method, however, especially if a language development system or toolkit is used to generate the implementation. DSL compilers are often called application generators. An alternative to the traditional approach to the implementation of DSLs is by embedding. In the embedding approach, a DSL is implemented by extending an existing GPL (the host language) by defining specific abstract data types and operators. A problem in a domain then can be described with these new constructs [19].

Therefore, the new language has all the power of the host language, but an application engineer can become a programmer without learning too much of it.

The Results of Hypotheses Testing: In this section of paper we present analysis the results of research hypotheses. Evaluation of the developed system is done with the following features against three programming languages/framework. The features are compatibility with java and dotnet, lightweight, concurrency with stm and ontology. Compatibility with java and dotnet is defined as the capability of existing and performing operation with java and dotnet. A lightweight programming language is one that is designed to have very small memory footprint, is easy to implement and/or has minimal syntax and features.

CONCLUSION

DSLs will never be a solution to all software engineering problems, but their application is currently limited by a lack of reliable knowledge available to DSL developers. To help remedy this situation, we distinguished five phases of DSL development and identified patterns in each phase, except deployment. Multiparadigm programming feature is accomplished by using homogeneous metaprogramming on a single platform/language. Thus it become an alternative of adding new platform to the environment as the embedding ontologies through a homogeneous DSL in a host language to support the features of the ontology paradigm. Complexities of multiplatform development are avoided [20-22].

REFERENCES

1. Kosar, T., N. Oliveira, M. Mernik, V.J.M. Pereira, M.C. repins¡ek, C.D. Da and and R.P. Henriques, 2010. Comparing general-purpose and domainspecific languages: An empirical study, Comp. Sci. Inf. Syst., 7(2): 247-264.
2. Spinellis, D., 2008. Rational Metaprogramming, IEEE Softw., 25(1): 78-79.
3. Djuric, D., D. Gasevic and V. Devedzic, 2006. The Tao of modeling spaces, J. Object Technol., 5(8): 125-147.
4. Henderson-Sellers, B., 2011. Bridging metamodels and ontologies in software engineering, Int. J. Sys., Sof., 84: 301-313.

5. Berners-Lee, T., J. Hendler and O. Lassila, 2001. The semantic web, Sci.Amer., 284(5): 28-37.
6. Mernik, M., D. Hrncic, B.R. Bryant and F. Javed, 2011. Applications of grammatical inference in software engineering: domain specific language development. In: Martin-Vide, C. (ed.): Scientific applications of language methods 2, Imperial College Press, London, pp: 421-457.
7. Ceh, I., M. Crepinšek, T. Kosar and M. Mernik, 2011. Ontology Driven Development of Domain- Specific Languages. Computer Science and Information Systems, 8(2): 317-342.
8. Kosar, T., P.E. Martinez Lopez, P.A. Barrientos and M. Mernik, 2008. A preliminary study on various implementation approaches of domain-specific language. Information and Software Technology 5, 50(5): 390-405.
9. Studer, R., R. Benjamins and D. Fensel, 1998. Knowledge engineering: Principles and methods". Data & Knowledge engineering, 25(1): 168-198.
10. Staab, S. and R. Studer, 2009. Handbook on Ontologies. Springer Verlag.
11. Lacy, L.W. and O.W.L., 2005. Representing Information Using the Web Ontology Language. Trafford Publishing.
12. Hebeler, J., M. Fisher, R. Blace and Perez-Lopez, 2009. A.: Semantic Web Programming. Wiley Publishing.
13. Tairas, R., M. Mernik and J. Gray, 2009. Using Ontologies in the Domain Analysis of Domain-Specific Languages. In: Models in Software Engineering. LNCS, Springer, 5421: 332-342.
14. Aho, A.I., M.S. Lam, R. Sethi and J.D. Ullman, 2007. Compilers: Principles, Techniques and Tools. Addison Wesley.
15. Sprinkle, M. Mernik, J.P. Tolvanen and D. Spinellis, 2009. What kinds of nails need a domain-specific hammer, IEEE Softw., 26(4): 15-18.
16. Kumaravel, B. Anatha Barathi, 2013. Personalized image search using query expansion, Middle-East Journal of Scientific Research, ISSN: 1990-9233, 15(12): 1736-1739.
17. Kumaravel, A. and R. Udayakumar, 2013. Web Portal Visits Patterns Predicted by Intuitionistic Fuzzy Approach, Indian Journal of Science and Technology, ISSN: 0974-6846, 6(5S): 4549-4553.
18. Kumaravel, A. and K. Rangarajan, 2013. Algorithm for Automation Specification for Exploring Dynamic Labyrinths, Indian Journal of Science and Technology, ISSN: 0974-6846, 6(5S): 4554-4559.
19. Kumaravel, A. and Oinam Nickson Meetei, 2013. An Application of Non-uniform Cellular Automata for Efficient Crytography, India n Journal of Science and Technology, ISSN: 0974-6846, 6(5S): 4560-4566.
20. Pattanayak, Monalisa. and P.L. Nayak, 2013. Green Synthesis of Gold Nanoparticles Using Elettaria cardamomum (ELAICHI) Aqueous Extract World Journal of Nano Science & Technology, 2(1): 01-05.
21. Chahataray, Rajashree. and P.L. Nayak, 2013. Synthesis and Characterization of Conducting Polymers Multi Walled Carbon Nanotube-Chitosan Composites Coupled with Poly (P-Aminophenol) World Journal of Nano Science & Technology, 2(1): 18-25.
22. Parida, Umesh Kumar, S.K. Biswal, P.L. Nayak and B.K. Bindhani, 2013. Gold Nano Particles for Biomedical Applications World Journal of Nano Science & Technology, 2(1): 47-57.