The Broker Solution Example of Domain Specific Languages Orchestration Framework

Branko Perisic[1], Ana Perisic[2], Marko Lazic[3]

[1] Faculty of Technical Science, University of Novi Sad, Serbia
[1]perisic@uns.ac.rs, [2]arhitektum@windowslive.com,[3]arhitektura@live.com,

Abstract:

Developing Domain Specific Languages faces with several challenging demands concerning problem domain/s and solution domain/s. Considering the recent explosion of Domain Specific Languages development it is obvious that certain problem domains are well covered with usable DSLs while others are in their infancy. Taking into the consideration the inherent complexity of underlining problem domains there are some recent researches targeting the federation or orchestration of several related DSLs in contrast to developing a single one. In this article there is the foundation of DSL Orchestration Framework, based on Broker Architectural Pattern, presented with respect to Architectural Design and Urban Planning Problem Domains.

1 Introduction

A general approach to problems solving, concerning engineering or reengineering something, addresses the following three interdependent domains: *Problem Domain*, *Solution Domain* and *Execution Domain*. A Domain is often understood as a *family* of *systems* exhibiting similar static structure, dynamic behavior and/or external functionality. According to [1] while attempting to established a sustainable general solution the domain experts and software designers are usually faced with demanding decision making process. This process addresses the wide variety of concerns like: exact definition of problem domain, targeted stakeholder gropes, intellectual clarity of fundamental concepts, the elegance and understandability of currently dominating methods, techniques and tools, the structure and behavior of any new tool or methodology that has to be developed, the efficiency and effectiveness of purposed solutions etc. According to [2], Domain Engineering (DE) is a systematic approach that provides a common core architectural framework supporting this challenging decision making process.

Designing something requires the ultimate understand of what a particular stakeholder wants, concerning that a sustainable solution emerges when: the expectations, support and real behavior of the created artifacts are compliant or suitably well aligned. The fundamental question is how one can acquire enough domain knowledge in order to formulate operationally usable abstractions. According to [3], the creation of domain mental model, that relies on empathy with a domain experts, may be a suitable way to gain the mutual understanding among domain experts and software engineers. Empathy with a person is distinct from studying how a person "*uses"* something, and extends to knowing what the person "*wants to accomplish"* regardless of whether he/she has or is aware of the thing that is being designed.

In this article we are focused on three crosscutting domains that emerge from the history of Civil Engineering (CE) that is a potentially unlimited origin of knowledge concerning the relationships established between humans and nature. The role of Architectural Design (AD) and Urban Planning (UP), based upon prescribed principles and doctrines, influenced by the systematic education and technology impacts, opens a completely new paradigm concerning the space and natural environment. Construction Industry (CI), although tightly coupled with the AD and UP domains, exhibits its routine mainly through the transition phase.

2 The Software Engineering Domain Aspects and DSL paradigm

Generally speaking, the history of software development is a history of raising the *level of abstraction* and the *level of reusability.* According to [4], raising the level of abstraction changes the platform on which each layer depends. In our opinion it is the main motivation factor that, combined with the reusability, may aid raising the domain experts efficiency and effectiveness. The recent explosion of Domain Specific Languages development has established a challenging target for novel "*silver bullet*" that drives the majority of current Model Driven Engineering researches. Certain problem domains are well covered with usable DSLs while others are in their infancy.

According to [5], DSLs have been used quite successfully in computer science already to build intuitive user interfaces that are understandable by non-IT experts but, until now they have not been adequately applied to the areas of Architectural Design and Urban Planning. The authors of [5] consider that article as the first step towards supporting the participatory, bottom up, approach to urban planning, formulating the intuitive interfaces (i.e. DSLs) as the key factors to its

success. However, concrete application of this approach including a practical meta-model and a common vocabulary for urban planning DSLs remains a topic for future researches.

3 The Problem Domain Aspects of DSL paradigm

According to [6], the primary aim of The Architectural Design Process is to guarantee that the architecture artifact is designed in such a way to simultaneously satisfy different representational, functional, aesthetic, and emotional needs of organizations and the people who intend to live or work in it.
The Architectural Design Process has to be well structured to ensure that stakeholders needs are satisfied in predefined priority chain order, thus preventing the case that resulting architecture artifact is the consequence of random collection of unrelated decisions.
The authors of [6] introduce the classic model of the seven steps Architectural Design Process encapsulating the following phases: *Pre-Design* (PD); *Schematic Design* (SD); *Design Development* (DD); *Construction Documents* (CD); *Bidding & Negotiation* (BN); *Construction Observation/Contract Administration* (CO/CA) and *Supplemental Services* (SS).
According to [7] Urban Design is the art of creating and shaping cities, towns and neighborhoods that holistically encompasses the disciplines of urban planning, architecture and landscape architecture in order to manage and transform the interactions of different aspects of urban life into a physical, usable and sustainable form.
Considering [8], urban spaces consist of a complex collection of: buildings, parcels, blocks and neighborhoods interconnected by streets. They are particularly difficult to model because of the large scale, ranging from few to several hundreds of square kilometers, and the underlying structure determined by a very large number of variables needed to describe: the land policies, market behavior, transportation infrastructure, governmental plans and population changes, that are hard to quantify and usually fuzzy in their nature.
Accurately modeling of the structure and the behavior of urban space is essential in spite of the fact that change processes are, from the point of view of information technology propulsion, very slow. On the other hand, considering the building utilization and spatial interactions, there are much more rapid changes that have to be encountered. That's why the authors of [8] state that it is essential for a rational approach to Urban Policy Process Management to rely on dynamic instead of static equilibrium models.

4 The Solution Domain - The DSL Orchestration Framework model

In order to formulate the Broker Based Orchestrator Framework model we have referenced the leading principles and ideas in the field of language driven software engineering and domain specific modeling paradigm. The traditional approach to DSL foundation [9],[10],[11] and establishes well known macro pattern or the methodology framework that guides the developers through the unfriendly world of DSL specification and development. Paul Hudak in [12] describes domain-specific languages (DSLs) as "*small programming languages tailored for a particular application domain*" consistently referring to the families of specific, similar problems suitable for linguistic description. As such, DSLs can be viewed as sets of general, all-encompassing solutions for problem domains. The traditional approaches focuses on Language and Ontology development as the main limiting and frustrating aspects mainly because of the lack of open-minded experts that posses the equal knowledge of problem domain, the domain of language construction and software engineering domain. From our opinion only team based work upon project based learning approach may create an synergic environment for complex problem domains handling.
On the other hand, according to [13], Domain Specific Modeling has been widely applied to provide dedicated tools to domain experts that better support the transformation of model to working product. According to [14], from embedded system design to enterprise architecture modeling, any Domain Expert faces the same situation: *multiplicity of views*; *multiplicity of concerns*; *multiplicity of models* and *heterogeneity of modeling artifacts*.
This raises the importance of standardization and interoperability on the artifact level (as through CityGML or BIM formalisms). The longevity, ease of learning and implementation are the most important characteristic of any paradigm or a particular tool-set. Domain experts do not wish to spend the whole life in order to gain a suitable level of expertise, or to dramatically change the gained daily routine while running for some promising product release. The effort needed to create a DSL with all accompanied tools is now a day significantly lower than ten years ago. It is particularly approved by the recent Language Workbenches solutions appearance like: Enso, Mas, SugarJ, Whole Platform, MPS, Onion, MataEdit+, Spoofax, Rascal, Xtext., that were compared in [15], Accelo [16], or SLEWorks described in [17] and [18], Sirius [15] and Sirius implementation described in [19].
According to [20], the multiple DSLs approach is particularly useful when dealing with diverse domains, as well as in situations where different domain experts need to work on a unified artifact.

The Building Information Management (BIM) paradigm is considered by the authors of [21] and [20] as the challenging while attempting to design an workflow fashion framework enabling the different domain experts to modify or analyze a model during its inception phase in opposition to the serialized approach of moving drawings from one group the next only after their completion.
Our experience with the related work analysis shows that the majority of researchers focus on an narrow domain that is justifiable but may lead to limited level of utilization making the effort hard to justify.
The state of the art of AD, UP and CE exhibits a little documented experiences with DSLs orchestration frameworks, which makes our approach a challenging one. Being conscious of the embedded complexity, in this article, we have suggested the Extensibility and the Orchestration of DSLs as a paradigm for sustainable DSL Framework development
In software engineering Extensibility is a system design principle that takes into the account the future growth of system under development. For the Orchestration of DSLs we suggest The Broker Architecture Pattern represented by an Architecture diagram shown in Figure 1.

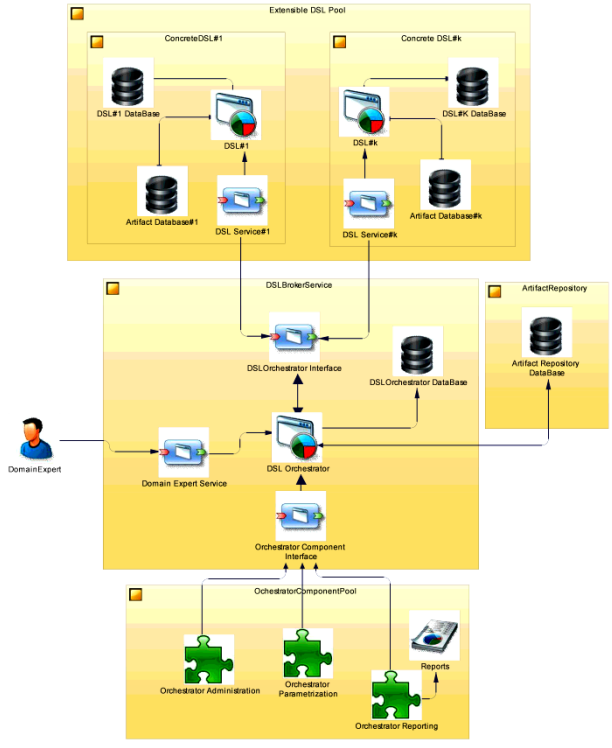

Figure 1. DSL Orchestrator Architecture Model based on Broker Pattern

The DSL Broker Service is used in order to structure independent DSLs (Concrete DSL) that are modeled like autonomous distributed software systems and support the cooperative work of different Domain Experts clustered over shared project artifacts. Concrete DSLs interact with Broker by remote service invocations implemented through the DSL Orchestrator Interface, specified by DSL Orchestrator. A DSL Broker Service component is responsible for coordinating communication, such as: forwarding requests, disseminating created artifacts and handling exceptions.
The extendibility is supported through autonomous implementation of DSL Orchestrator Interface by a DSL instance that joins the Extensible DSL Pool.
DSL Orchestrator Database stores all the relevant information concerning registered DSLs instances and their mediation.
The Orchestrator Component Pool hosts the Orchestrator supporting services dedicated to: Administration, Parameterization and Reporting.
The Orchestrator Component Pool itself is freely extendable based on the Orchestrator Component Interface implementation.

5 Conclusion and Further Research Directions

Considering the Language Workbench researches it is obvious that the creation of small languages today is fare better supported than just several years ago. This does not minimize the disciplined way of DSL development, but makes it possible to develop the family of closely related DSLs that may be orchestrated in order to raise the level of abstraction when orchestrating complex engineering domains. Developing Domain Specific Languages faces with several challenging demands concerning problem domain and solution domain..
Taking into the consideration the underlining problem domain complexity there are some recent researches targeting the federation or orchestration of several related DSLs in contrast to developing a single one.

In this article there is the foundation of Broker Based DSL Orchestrator presented. The extensibility of DSLs is introduced and discussed from community driven development perspective as well as the role of interoperability and synergic approach to multiple DSLs orchestration.

The directions of further researches mainly concentrates on the following:

- the operational verification of stated Broker based Orchestration Framework;
- the development of meta language specification framework that may coordinate different Language Workbenches in order to compare their effectiveness concerning the ease and speed of DSLs development concerning selected abstract levels of Architectural Design and Urban Planning domains; and

Literature

[1] Lapets, A.: Algebraic Semantics of Domain-Specific Languages, (2006) [Online]. Available: http://cs-people.bu.edu/lapets/resource/algsemdsls.pdf (current Jun 2014)

[2] Fawcett, P.: The Architecture Design Notebook, Architectural Press An imprint of Elsevier Linacre House, Oxford, Burlington. (2003)

[3] Young I.: Mental Models: Aligning Design Strategy with Human Behavior. Rosenfeld Media, New York. (2008)

[4] Bryant, B. R., Gray, J., Mernik, M., Clarke, P. J., France, R. B., Karsai, G.: Challenges and Directions in Formalizing the Semantics of Modeling Languages. ComSIS Vol. 8, No. 2, 225-253. (2011)

[5] Kramer, M., Ludlow D., Khan, Z.: Domain-specific languages for agile urban policy modelling, Proceedings 27th European Conference on Modeling and Simulation, May 2013, Norway. (2013)

[6] Werner, S., Long, P.: Cognition Meets Le Corbusier – Cognitive Principles of Architectural Design. In: Freksa, C. et al. (eds.): Spatial Cognition III, Springer-Verlag Berlin Heidelberg, 112–126. (2003)

[7] Patsucha, S., Glodney, B., Carol, C., Nameth, J., Sampson, S.: Urban Design Principles, Gensler, City of Los Angeles, (2011).[Online].Available:http://www.urbandesignla.com /resources/docs/DesigningAHealthyLA/hi/DesigningAHealthyLA.pdf (current March 2014)

[8] Simmonds, D., Waddell, P., Wegener, M.: Equilibrium versus dynamics in urban modeling, Enviroment and Planning B: Planning and Design, Vol 40, No. 6. (2011).

[9] Mernik M., Heering J., Sloane A. M.: When and how to develop domain-specific languages. ACM Computing Surveys, Vol. 37, No. 4, 316–344. (2005)

[10] Sprinkle J., Mernik M., Tolvanen J. P., Spinellis, D.: Guest Editors' Introduction,What Kinds of Nails Need a Domain-Specific Hammer? IEEE Software, Vol. 26, No. 4, 15-18. (2009)

[11] Kosar T., Oliveira N., Mernik M., Varanda Pereira M. J., Črepinšek, M., da Cruz D., Henriques P. R.: Comparing General-Purpose and Domain-Specific Languages: An Empirical Study. Computer Science and Information Systems, Vol. 7, No. 2, 247-264. (2010)

[12] Hudak, P.: Modular domain specific languages and tools. Proceedings: Fifth International Conference on Software Reuse, IEEE Computer Society Press. pp. 134–142. (1998).

[13] Völter M., Visser., E.: Language extension and composition with language workbenches., In: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. ACM. 2010, pp. 301-304. (2010)

[14] Guychard, C., Guerin, S., Koudri, A., Beugnard, A., Dagnat, F.: Conceptual interoperability through Models Federation, Semantic Information Federation Community Workshop. (2013)

[15] Sirius Documentation, [Online]. Available: http://www.eclipse.org/sirius/doc/ (current April 2014)

[16] Acceleo. [Online]. Available: http://www.eclipse.org/acceleo/, (current April 2014).

[17] Dejanović I., Perišić, B., Milosavljević G.: MoRP Meta-metamodel: Towards a Foundation of SLEWorks Language Workbench, The 2nd International Conference on Information Society Technology (ICIST 2012), Kopaonik, Serbia, pp. 36-40. (2012)

[18] Dejanović, I., Milosavljević, G., Perišić, B., Vasiljević, I., Filipović, M.: Explicite Support for Languages and Mograms in the SLEWorks Language Workbench, The 3nd International Conference on Information Society Technology (ICIST 2013), Kopaonik, Serbia, pp. 167-172. (2013)

[19] Vujović, V., Maksimović, M., Perišić, B., Milošević V.: A Graphical Editor for RESTfull Sensor Web Networks Modeling, SACI2014, IEEE 9th International Symposium on Applied Computational Intelligence and Informatics, Temisovar, Romania, (2014)

[20] Northrop, L., Clements, P., Bachmann, F., Bergey, J., Chastek, G., Cohen, S., Donohoe, P., O'Brien, L.: A framework for software product line practice, version 5.0. (2007) [Online]. Available: http://www.sei.cmu.edu/productlines/frame_report/ (current Jun 2014)

[21] Kosar T., Oliveira N., Mernik M., Varanda Pereira M. J., Črepinšek, M., da Cruz D., Henriques P. R.: Comparing General-Purpose and Domain-Specific Languages: An Empirical Study. Computer Science and Information Systems, Vol. 7, No. 2, 247-264. (2010)