Building Domain Specific Languages upon the Object Constraint Language (OCL)

Extended abstract

Antonio Dominguez

antonio@ultra.sig.uvigo.es

Universidade de Vigo

Abstract

The paper aim is to demonstrate how to extend OCL, both with new data structures and operations, in order to use it as a domain specific specification language. This data structures and operations are domain specific, and identified as a result of a domain analysis process. In conjunction with appropriate transformation techniques it will be possible to generate code directly from system specifications.

1. Introduction

Software Information Systems Engineering has usually three phases: system specification, system design and system implementation. The first phase is probably the most important and also the most difficult. In this initial stage, software analysts must understand the business process and produce a system specification. However it is only possible to verify the specification correctness when the final system is built and was up and running for a long time.

Most approaches on improving software quality, reducing software production and operation costs, and time to market, are centered on various software reuse techniques. This implies moving to a dual software engineering process, the first of that is called domain engineering, and the second one is the traditional software systems engineering process. Domain Engineering[1] has three main phases: domain analysis[2], [3], domain design and domain implementation .

Although with the dual software development approach it is possible to achieve improvements in software engineering, it is difficult to achieve these improvements in some domains [1], mainly because each reusable software component must be reused several times in different systems in order to return the initial investment.

Clearly using a specification language, common both to software engineers and system users, can improve the software engineering process. This paper describes an approach for using OCL as a base to obtain domain specific languages, that can be used to build system specifications. And those specifications maybe transformed in final

---

[1] A domain is seen as a set of related systems[4], [5], [6], [7]

systems.

2. Domain Specific Specification Languages.

Generic specification languages traditionally are seen difficult, and definitely not user oriented. Specification languages common to software engineers and systems users must be easy and useful. A good approach to obtain specification languages more user friendly is restricting its generality to a very known domain[8].

In many cases those languages are seen as a lateral product of the domain analysis, and used to ease the process of documenting the domain components, and domain structure. Although the most of the domain analysis techniques refer in same way to this kind of languages, there is no a known and formalized process to obtain DSL in a given domain.

The approach of considering this languages as specification languages, that can be used in the given domain to build specifications of new information systems can be improved if specifications written with these languages can be transformed in several steps into final code, using transformational approaches[9], [10], [11], [12].

3. The Object Constraint Language as a specification language.

The Object Constraint Language is a formal language that has been developed as a business modeling language within the IBM Insurance division, and adopted as part of UML(Unified Modeling Language)[13], [14], [15]. Useful on writing system constraints, can be used associated with UML models or any other modeling language. It is not a programming language, so it is not possible to write program logic or flow control[16]. OCL expressions, when evaluated simply return a value. So it can be used as a query language, to write queries that return the values satisfying the constraint specified by the query.

A constraint is basically an abstract representation of the possible values a data element or system function can take, can manipulate, or may return. Constrains refer only to a relevant aspect of the component being described, so it is possible to use a set of constraints to describe each component. Using a constraint language is an important step in order to formalize the system specification process??.

OCL can be used as a base to implement domain specific languages, if some of his limitations are removed, and additional funcionalities are added. As a query language [2] OCL has several limitations. OCL includes as primitive operations union, intersection and difference. Selection can be expressed in OCL. But it has problems when dealing with cartesian product, projection, quotient and join. Definitely it is no a good language for data manipulation, but it can be easily extended to do so and to express domain functionality, as showed in the nest section.

---

[2]Query languages, proposed initially by Codd are those that can simulate tuple calculus or the equivalent relational algebra or domain calculus. Predicate calculus languages are those languages where queries describe a desired set of tuples by spifying a constraint they may satisfy.

4. Extending OCL for Business Domain Modeling

A domain model is the result of a domain analysis process. Obtaining in this process a domain specific specification language is an additional effort that can bring us many useful advantages, that can be improved in case of having a transformer, that were able to generate systems automatically.

Using OCL as a base for a formal domain specific language is based in the two following assumptions:

1. As constraint language can be easily extended with new constraint clauses. The new clauses are identified in the domain analysis process. Examples of this kind of clauses can be:

   - Temporal constraints, and or real time constraints.
   - Order constraints and arquitechtonic constraints.
   - Data flow constraints, and so on.

2. It is possible to implement automatic generation of code, using transformation techniques. So using appropriate transformations, different system versions can be obtained by changing system specifications and/or system infrastructure, simple by doing a new (very low costly)code generation process.

In doing the first it is necessary to consider business domains as having certain abstract structure. First there are domain data elements, second there are domain actors, internal and external of the domain, a finally there are domain activities or funcionalities. There is not the objective of languages being discussed there to deal with domain data of with interactions between activities. The main objective is to deal with the description of domain activities or funcionalities. In doing that, it is necessary to include in the language clauses useful to describe easily this funcionalities, to build an appropriate transformation mechanism.

5. Conclusions

Obtaining as products of domain analysis a Domain Specific Specification Language, and an appropriate set of transformers will allow automatic code generation of systems from systems specifications, users and systems engineers can communicate using common tools. If necessary different versions of the same system can be easily obtained, thus facilitating the construction of families of systems.

References

[1] **M. Simos**. 1987 (oct). The Domain-Oriented Software Life Cycle: Towards an Extended Process Model for Reusability, from *Proceedings of the Workshop on Software Reuse.* Boulder, CO.

[2] **G. Arango and R. Prieto-Diaz**. 1991. Domain Analysis and Software Systems Modeling IEEE Computer Society Press. Chap. Domain Analysis Concepts and Research Directions, pages 9-

33.

[3] **G. Arango**. 1993. Domain Analysis Methods, from *Software Reusability.*, ed. W. Shaefer and R. Prieto-Diaz and M. Matsumoto Ellis Horwood.

[4] **Yellamraju V. Srinivas**. 1991. Domain Analysis and Software Systems Modeling IEEE Computer Society Press. Chap. Algebraic specification fo Domains, pages 90-119.

[5] **M. Natori and A. Kagaya and S. Honiden**. 1996 (apr). Reuse of Design Processes Based on Domain Analysis, from *Proceedings of the Fourth International Conference on Software Reuse.* Pages 31-40.

[6] 1996. *Organization Domain Modeling (ODM) Guidebook, Version 2.0.* Software Technology for Adaptable, Reliable Systems (STARS), Technical Report STARS-VC-A025/001/00.

[7] **M. Simos**. 1997. Lateral Domains: Beyond Product-Line Thinking, from *Proceedings of the Eight Workshop on Institutionalizing Software Reuse*

[8] **Valeri N. Agafonov**. 1997. Reuse of General Specification Notions and Specification Languages, from *Proceedings of the Eight Workshop on Institutionalizing Software Reuse.*

[9] **J. do Prado Leite and M. Sant'Anna and F. de Freitas**. 1994. Draco-PUC: a Technology Assembly for Domain Oriented Software Development, from *Proceedings of the Third International Conference on Software Reuse.* Pages 94-100.

[10] **James Neighbors**. 1981. *Software Construction Using Componets.* PhD thesis. University of California at Irvine.

[11] **J. Neighbors**. 1984 (sep). The Draco Approach to Constructing Software from Reusable Components. *IEEE Transactions on Software Engineering*, **10**(5), pages 564-573.

[12] **J. Neighbors**. 1991. Domain Analysis and Software Systems Modeling IEEE Computer Society Press. Chap. DRACO: A Method for Engineering Reusable Software Systems, pages 34-52.

[13] **UML**. 1997. *UML Summary.* Technical report. Rational Software Corporation.

[14] **UML**. 1997. *UML Notation Guide.* Technical report. Rational Software Corporation.

[15] **UML**. 1997. *UML Semantics.* Technical report. Rational Software Corporation.

[16] **UML**. 1997. *Object Constraint Language Specification v1.1.* Technical report. Rational Software Corporation.