Domain-Specific Software Development Terminology:

Do We All Speak the Same Language?

Arturo J. Sánchez-Ruíz [1], Motoshi Saeki [2], Benoît Langlois [3], Roberto Paiano [4]

[1] University of North Florida, School of Computing
1 UNF Drive, Jacksonville, FL 32246, USA
asanchez@unf.edu

[2] Department of Computer Science, Tokyo Institute of Technology
Ookayama 2-12-1, Meguro-ku, Tokyo 152, Japan
saeki@se.cs.titech.ac.jp

[3] Thales D3S/EPM, Domaine de Corbeville, 91404 Orsay Cedex, France
benoit.langlois@thalesgroup.com

[4] University of Lecce
Via per Arnesano, 73100 Lecce (LE), Italy
Roberto.Paiano@unile.it

Abstract

We are interested in determining whether researchers and practitioners of the Domain-Specific Software Development (DSSD) community mean the same when they refer to core terms commonly used as part of the area's nomenclature. In this position paper we report on an initial exercise we began at the 2006 OOPSLA Workshop on Domain-Specific Modeling which consisted of identifying these core terms, agreeing on their meaning, and building a concept map showing how they relate to each other. We hope this exercise will serve as one of the initial seeds which will help build an evolving and multidimensional body of core definitions researchers and practitioners of our community can use as an authoritative reference.

1. Introduction

It is a tradition of the OOPSLA Workshop on Domain-Specific Modeling, now in its seventh incarnation, to organize small focus groups to discuss topics of relevance in the area, as the second activity of the day.[1] During the 2006 edition of this workshop, the authors of this position paper constituted a group whose topic was "nomenclature." Some of the group members proposed this topic arguing that it was important to identify a set of "core terms," i.e. fundamental concepts which define the platform on which Domain-Specific Modeling is practiced. The proponents also argued they sometimes were under the impression that not always researchers and practitioners meant the same when using some of these terms.

Right at the beginning of our discussion we clearly realized it was not easy to identify what terms should be considered as "core," and once we identified some of them it was not easy to just recite a definition for each of them because they were all tightly connected to each other. One of the group members proposed to use a concept map as a way to put all the concepts on a common canvas. This exercise resulted in a messy map with different kind of arrows and relationships. [2]

We decided to continue the exercise, cooperating remotely across four different countries, and three different continents, and this position paper reports on the current status of our cooperation. In section 2 we described the methodology we followed. Section 3 presents the set of core terms we identified with their associated converging definitions. Section 4 shows the concept map which puts all these terms together. The paper ends with our conclusions, suggestions for future work, and a list of additional references we consulted. [3]

2. Methodology

We approached the exercise as follows.

- **Iteration 0**: agree upon an initial set of core terms. The following terms were identified: Domain, Application, Level of Abstraction, Language, Syntax, Semantics, Model, Meta-Model, Ontology, Domain-Specific Modeling, and Domain-Specific Language.
- **Iteration 1**: each participant creates a document containing a succinct definition for each term, citing examples and authoritative sources when considered appropriate.
- **Iteration 3**: try to get to a set of converging definitions and produce a document with them.
- **Iteration 4**: build a concept map showing these core terms and relationships among them.

The following sections present what we produced after applying this methodology.

3. Core Terms and Their Definitions

For the sake of clarity, we decided to include a "common grounding" which defines a context framing all the terms. We also slightly modified some term names. For instance, instead of using "Domain", we decided to use "Application Domain." We introduced additional related concepts when we considered it convenient. For instance, we added "Meta-Language" as part of the definition of "Language."

---

[1] The first part of the workshop is dedicated to paper presentations, and the third part is dedicated to brief presentations by the focus groups.

[2] It turns out the organizers took some pictures of us working and of our initial concept map, which can be seen at http://www.dsmforum.org/events/DSM06/Pictures/Group_e_small.jpg and http://www.dsmforum.org/events/DSM06/Pictures/Group_reporting.jpg, respectively.

[3] Numerous references are also presented as footnotes.

Finally, closely related terms are defined in a group. What follows is the list of all the terms and our definitions.

3.1. Common Grounding

A common ground to all the terms is defined by the existence of problems and the need to solve them. Problems vary in their level of complexity, but all these terms appear in the context of finding approaches to solve them, regardless of their complexity. The complexity of a problem can be characterized, among other traits, by the following: number of objects, relationships among these objects, and computational requirements (e.g. amount of memory space and processing speed.)

Examples of very complex problems often appear in the call for approaches to "grand challenges" [4] in various branches of science and technology. Examples of medium complexity problems include large-scale E-Commerce applications and on-line Multiplayer Gaming. Examples of low complexity problems include the management of information in small organizations.

3.2. Software Application

A software application is often an important component of an approach to solve a problem. In other words, a software application is (usually) a fundamental part of the solution to a problem. Therefore, we acknowledge the existence of problems whose solutions cannot be expressed just as software applications (i.e. it takes more than a software application to solve some problems.)

3.3. Application Domain

Intuitively, the application domain frames the problem at hand. More precisely, the application domain is characterized by the relevant objects (a.k.a. concepts) and their relationships. The relevance is relative to the problem at hand, and the decision of whether or not certain objects and relationships are relevant is made by the team who analyzes the domain in the context of a problem which needs a solution (a.k.a. "Analysts.")

Domains can be quite large, for instance "Medical," "Legal." They can be focused, for instance the domain defined by a personal collection of music CDs and movie DVDs. They can also be arranged in some hierarchical structure. For instance, the domain associated with the problem of finding immunizations for rapidly changing human viruses would be a sub-domain of "Medical."

3.4. Abstraction and Levels of Abstractions

When analyzing a domain, in the context of building approaches to solve a given problem, analysts practice a cognitive process known as abstraction, which consists of concentrating on the "essence" of the domain, ignoring elements that are considered to be superfluous.

Abstraction is practiced when analysts need to decide which objects and relationships are relevant, and when they document and specify such objects and relationships. This definition introduces a de facto bitonality: the superfluous versus the essential. When this process is applied iteratively (or some would say recursively or fractally) the result is a lattice of layers, each of which is referred to as a level of abstraction.

---

[4] Just do a google search on "Grand Challenges" to see specific examples.

This technique is commonly used in computing, and a popular example is the set of layers that cover all the way from the hardware of a computer to a programming language such as Java: towards the bottom of the layering we might find elements such as gates and circuits, towards the top we find the application programming interface (API,) somewhere in-between these two we might find the Java Virtual Machine (JVM.) The decomposition of a domain into layers includes the definition of transformations (mappings) which relate elements from different layers.

3.5. Language, Alphabet, Vocabulary, Syntax, Semantics, and Meta-Language

A language allows analysts to characterize domains, problems within these, and solutions to such problems. A language is defined by its alphabet, vocabulary, syntax, and semantics.

The alphabet is the set of symbols that are used to build elements in the vocabulary. The vocabulary is the set of allowed "words". [5] Sometimes the vocabulary is built by the application of "lexical rules," that is to say those which define how words are constructed from elements of the alphabet.

The syntax is the set of rules that define all possible "phrases" in the language (i.e. the set of syntactically correct phrases,) which are built from words in its vocabulary. Finally, the semantics is the set of definitions that establish the "meaning" of all possible syntactically correct phrases in the language.

Notice that rules and definitions associated with lexical, syntactical, and semantic elements need to be expressed with the help of alternative languages, these languages are called "meta-languages" because they are used to define other languages.

As an example let us consider a programming language, say Java. [6] Its alphabet is Unicode. [7] An example of a lexical rule would be a regular expression which define how identifiers are built, therefore the language of regular expressions is a meta-language used to define the lexical structure of the language Java. The language of regular expressions [8] has, in turn an alphabet, vocabulary, syntax, and semantics. The language used to define these elements would be a meta-meta-language. This process is not infinite. It stops at the meta-meta level, with the Zermelo-Fraenkel Set Theory, [9] which gives meaning to regular expression phrases.

An example of a syntactic rule would be the portion of Java's grammar used to define control structures. If the well-known notation referred to as the "Backus-Naur Form" (BNF) is used to define the grammar, then BNF would be the meta-language used to define the syntactic structure of the language. In this case, we find the theory of formal languages at the meta-meta level. [10]

There are various approaches used to define the semantics of programming languages, [11] here we briefly mention just two. In the denotational approach, the semantic of a phrase in the language is defined by a function that maps such phrase to chosen mathematical structure. In this case, said mathematical structure plus set theory would be the meta-meta-language used to define the semantics of the programming language. In the operational approach, the semantics of a phrase is defined by a

---

[5] Synonyms of vocabulary include "set of lexemes", and "lexicon."
[6] See http://java.sun.com/docs/books/tutorial/
[7] J. Gosling et alia: "The Java Language Specification", Third Edition. Addison-Wesley, 2005. See http://java.sun.com/docs/books/jls/
[8] See http://mathworld.wolfram.com/RegularExpression.html
[9] See http://plato.stanford.edu/entries/set-theory/ZF.html
[10] J. E. Hopcroft, and J. D. Ullman: "Introduction to Automata Theory, Languages, and Computation". Addison-Wesley, 1979.
[11] C. A. Gunter: "Semantics of Programming Languages – Structures and Techniques". MIT Press, 1992.

translation of such phrase to a program in an abstract machine. In this case, the language used to define the abstract machine and the translations constitute the meta-language used to define the semantics of the programming language. The meta-meta-language would be what gives meaning to these translations.

3.6. Model and Meta-Model

The process of identifying (or, equivalently eliciting,) documenting, and specifying the objects and their relationships, that are relevant in the context of a given problem, is known as modeling the domain (a.k.a. Domain Modeling.) The result of this process is a model of the domain (a.k.a. Application Domain.)

Abstraction is used when building a domain model, and sometimes various layers are defined. A language is needed to define the model. The meta-language that is used to define the semantics of this language is referred to as a meta-model (i.e. the model used to define the meaning of another model.) The language that is used to define the semantics of the meta-language would therefore be the meta-meta-model. Typical meta-meta-languages (and therefore associated with the corresponding meta-meta-Models) are mathematical theories such as the Zermelo-Fraenkel Set Theory, and First-Order Logics,[12] to name just two that are very popular.

3.7. Ontology

In the context of philosophy, ontology is a science that studies concepts such as being and existence. [13] In the context of Computing, an ontology is a specification of the knowledge about a domain. A definition of ontology (in the latter sense) that has been extensively used is due to Tom Gruber: "An ontology is an explicit specification of a conceptualization." [14] The term "conceptualization" is used with the meaning given in the paper by Genesereth and Nilsson and paraphrased by Gruber as: "… objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold them". [15]

Given these definitions, it seems plausible to equate an ontology with a model of a domain of application. In other words building an ontology for a domain of application is one approach to model such domain. Also, since a model can be used to give semantics to another model, an ontology can play the role of a meta-model as well.

A popular characterization of ontologies, discussed in a paper by Deborah McGuiness, [16] is presented as a linear spectrum based on the formality used in defining an ontology. For instance, on the "less-

---

[12] J. Barwise, and J. Etchemendy: "Language, Proof, and Logic". Center for the Study of Language and Information (CSLI). See http://cslipublications.stanford.edu/site/157586374X.html

[13] For an introduction, see http://plato.stanford.edu/entries/logic-ontology/

[14] Thomas R. Gruber: "A Translation Approach to Portable Ontology Specifications". Knowledge Acquisition, 5(2):199-220, 1993.

[15] M. R. Genesereth, and N. J. Nilsson: "Logical Foundations of Artificial Intelligence". Morgan Kauffman Publishers, 1987.

[16] Deborah L. McGuiness: "Ontologies Come of Age". In Dieter Fensel, Jim Hendler, Henry Lieberman, and Wolfgang Wahlster, editors. The Semantic Web: Why, What, and How, MIT Press, 2001. In this paper, the author notes: "This spectrum arose out of a conversation in preparation for an ontology panel at AAAI '99. The panelists (Lehman, McGuinness, Ushold, and Welty), chosen because of their years of experience in ontologies found that they encountered many forms of specifications that different people termed ontologies. McGuinness refined the picture to the one included here."

formal" side of the spectrum one finds a catalog. On the "more-formal" side of the spectrum one finds ontologies based on mathematical logic.

A more recent characterization, derived from discussions that have taken place in the "Ontolog Community of Practice – ONTOLOG," has identified various semantic and pragmatic dimensions. One of the co-authors of this paper attended ONTOLOG's "Ontology Summit 2007," which was held at the National Institute for Standards and Technology (NIST) in April of 2007. The main goal of this meeting was to build a common framework to characterize a wide variety of ontologies. [17]

Notice that regardless of the approach used to build an ontology, a language must be used to define it, and such language must have all the elements that we have previously defined.

Let us consider, as an example, the case of Wine Making as the application domain. [18] This can be considered as a sub-domain of Manufacturing Processes. [19] The PSL [20] ontology, developed by NIST is an ISO standard (ISO-18629) which defines fundamental manufacturing process concepts and their relationships, structured as a lattice of First-Order Logic theories. One language that is used to describe PSL is the Knowledge Interchange Format (KIF). [21] Suppose we have developed WMVL, the "Wine-Making Visual Language," which analysts of this domain can use to build models of their manufacturing processes. Consider now the problem of efficiently allocating certain resources needed for wine manufacturing. Analysts can build models for this problem using WMVL whose semantics can be defined using PSL (ontology as a meta-model), whose language is KIF (meta-language), whose semantics is given by First-Order Logic (meta-meta-model and meta-meta-language).

3.8. Domain-Specific Modeling (DSM)

DSM is the process of building a model for a specific domain. The following phrase, taken from our DSM forum succinctly captures the relevance of this term: "Domain-Specific Modeling raises the level of abstraction beyond programming by specifying the solution directly using domain concepts." [22] From the perspective of the end-user, the level of abstraction defined by a programming language exposes too many details which are not relevant to said user. It is therefore too low. By empowering the user with the ability to build solutions using concepts that are germane to the domain, DSM effectively raises the level of abstraction in the sense that it builds a substrate (i.e. the domain-specific model) which is closer to the way end-users conceptualize solutions in their domain.

DSM is therefore a specialization of the general concept of modeling. It is thus natural to use terms such as modeling, models, meta-models, languages, and ontologies in the context of practicing DSM.

3.9. Domain-Specific Language (DSL), and Domain-Specific Software Development (DSSD)

In connection with DSL and DSSD one finds the language, translators, and environments associated with the model built for the specific domain. End-users, as opposed to software developers, utilize domain-specific languages and environments to effectively build solutions to problems in their domain.

---

[17] See http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologySummit2007 and also http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologySummit2007_FrameworksForConsideration/DimensionsMap
[18] For an example, see http://www.winebusiness.com/html/MonthlyArticle.cfm?dataid=3563
[19] John A. Schey: "Introduction to Manufacturing Processes", McGraw-Hill, 1999.
[20] See http://www.mel.nist.gov/psl/
[21] See http://www-ksl.stanford.edu/knowledge-sharing/kif/
[22] See http://www.dsmforum.org/

The underlying languages, translators, and environments generate the code that implements the actual software solution.

If we consider, once again, the case of Wine Manufacturing, the hypothetical WMGL would be an example of a DSL. For this DSL to have practical use in the hands of end-users, translators and generators would be needed in order to map WMGL specifications to executable software (i.e. written in a programming language such as Java, C, C++, C#, et cetera.) The meta-models/languages used to give semantics to WMGL and the translators/generators would also be part of the whole process which in an instance of Domain-Specific Software Development (DSSD.)

4. A Concept Map

Novak and Cañas define concept maps as: "graphical tools for organizing and representing knowledge. They include concepts, usually enclosed in circles or boxes of some type, and relationships between concepts indicated by a connecting line linking two concepts. Words on the line, referred to as linking words or linking phrases, specify the relationship between the two concepts." [23] The concept map presented in Figure 1 was built using CmapTools. [24]

5. Conclusions and Future Work

In this paper we have presented the result of an exercise we initiated during the 2006 OOPSLA Workshop on Domain-Specific Modeling which consisted of identifying core terms used by our community of researchers and practitioners with the goal of determining if we meant the same when referring to them. We requested each participant to submit succinct definitions for these terms, and then tried to find converging definitions. Finally, we built a concept map connecting these terms.

We believe it was instructive to frame DSM in the context of DSSD, the latter being an approach to software development, which in turn deals with building software solutions for problems within a general domain. From this perspective DSSD is seen as an activity which focuses on specific domains. Modeling has therefore central importance in DSSD, and this is clearly depicted by the concept map.

A pattern suggested by the concept map is that "meta-X" as a qualifier is naturally used to name entities that are used to define the semantics of X. It also suggests that "Model" and the "Language" used to express it can sometimes be used interchangeably when such use does not introduce artificial ambiguities.

We found that concept maps can be used as a clean approach to visually organize a set of concepts with multiple relationships among them. The use of CmapTools enables the multi-layered definition of concepts and relationships annotated with hyperlinks. For instance, one can define a concept map which uses concepts and relationships such that the clicking on a concept takes the user to the concept map for such concept. This leads to a natural extension and generalization of the exercise presented here, namely the construction (via concept maps) of a layered set of core terms for the DSSD community which can dynamically grow and serve as an authoritative reference to its researchers and practitioners. We hope our results will be part of an initial seed that will be used to build such a body of knowledge.

---

[23] J. D. Novak and A. J. Cañas: "The Theory Underlying Concept Maps and How to Construct Them". Technical Report (IHMC CmapTools 2006-01), Institute for Human and Machine Cognition, 2006. Available from http://cmap.ihmc.us/Publications/ResearchPapers/TheoryCmaps/TheoryUnderlyingConceptMaps.htm

[24] See http://cmap.ihmc.us/

Additional References

[1] Clements, P., Northrop, L. *Software Product Lines, Practices and Patterns*, Addison-Wesley, 2002.

[2] Czarnecki, K., and Eisenecker, U.W. *Generative Programming*, Addison-Wesley, 2000.

[3] Evans, E. *Domain-Driven Design, Tackling Complexity in the Heart of Software*, Addison-Wesley, 2004.

[4] Fowler, M. *Language Workbenches: The Killer-App for Domain Specific Languages?*, http://www.martinfowler.com/articles/languageWorkbench.html

[5] Greenfield, J., Short, K., Cook, S., and Kent, S. *Software Factories, Assembling applications with Patterns, Models, Framework, and Tools*, Wiley, 2004.

[6] Langlois, B., Barata, J., Exertier, D. *Improving MDD Productivity with Software Factories*, OOPSLA 2005, First International Workshop on Software Factories.

[7] Langlois, B., Exertier, D., Devda, G. *Towards Families of QVT DSL and Tool*, Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling.

[8] Stahl, T., Völter, M., Bettin, J., Haase, A., Helsen, S. *Model-Driven Software Development*, Wiley, 2006.

[9] Sánchez-Ruíz, A. J., Hansen, G. *Translation Patterns to Specify Processes in the PSL Ontology*. Proceedings of the 5th OOPSLA Workshop on Domain-Specific Modeling.

[10] Paiano, R., Pandurino, A., Guido, A. L. *Conceptual Design of Web Application Families: The BWW Approach*. Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling.

[11] Saeki, M., Kaiya, H. *On Relationships Among Models, Meta Models, and Ontologies*. Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling.

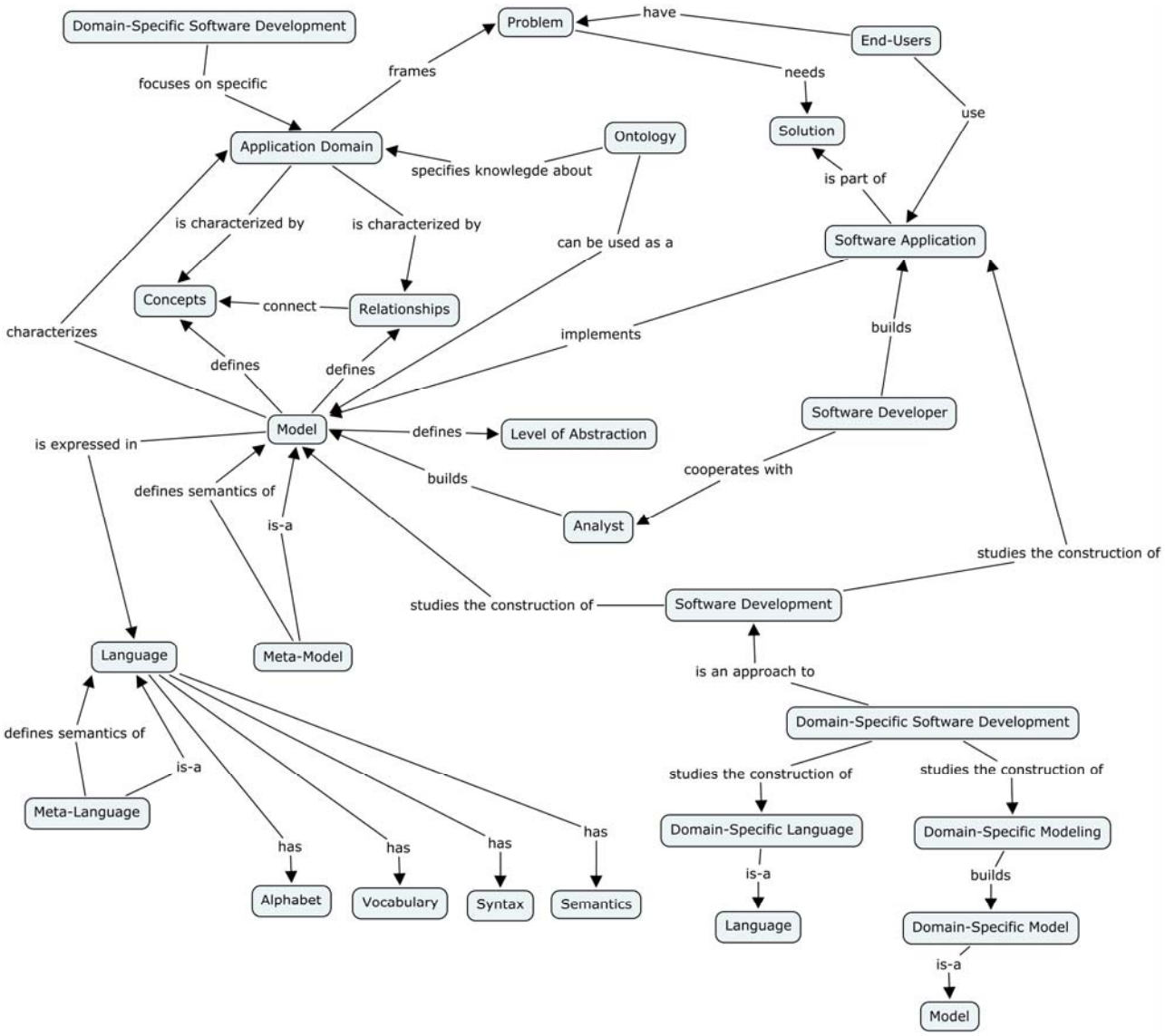

Figure 1: Our Concept Map