Reinaldo de Souza Junior

A Domain Specific Language for OntoUML

Vitória - ES, Brasil

2013

Reinaldo de Souza Junior

A Domain Specific Language for OntoUML

Monografia apresentada ao curso de Ciência da Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Departamento de Ciência da Computação

Supervisor: Prof. Dr. João Paulo Andrade Almeida

Vitória - ES, Brasil

2013

Reinaldo de Souza Junior

A Domain Specific Language for OntoUML

Monografia apresentada ao curso de Ciência da Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Trabalho aprovado. Vitória - ES, Brasil, ?? de dezembro de 2013:

Prof. Dr. João Paulo Andrade Almeida
Orientador

Professor
Convidado 1

Professor
Convidado 2

Vitória - ES, Brasil
2013

Acknowledgements

Abstract

Essa monografia provê uma sintaxe textual para a criação de modelos conceituais baseados numa linguagem de modelagem filosoficamente e cognitivamente bem-fundada, chamada OntoUML, bem como um editor para o uso desta linguagem. Primeiramente, revisamos as restrições de integridade contidas no metamodelo da linguagem OntoUML e as adequamos de modo a ser compatível com o ferramental atual. Em seguida, propomos uma sintaxe concreta textual (uma linguagem específica de domínio, DSL) para representar os conceitos existentes no metamodelo. Na sequência, desenvolvemos um editor textual para que usuários possam criar e manipular modelos OntoUML por meio da DSL proposta. Objetivo é criar uma linguagem textual que represente os conceitos da linguagem OntoUML e fornecer a ferramenta necessária para seu uso. Portanto, nesse trabalho propomos uma sintaxe concreta para a linguagem OntoUML e provemos uma ferramenta que possibilita a edição, validação sintática e transformação de modelos.

Palavras-chaves: OntoUML. linguagem específica de domínio. Eclipse. Xtext.

Abstract

This work provides both a textual sintax to creating conceptual models based on a philosophically and cognitively well-founded modeling language, named OntoUML, and an editor to the language itself. First, we review the integrity constraints contained in the language's metamodel and we suit them to be compatible with the current tooling. Then, we propose a textual concrete syntax (a domain specific language, DSL) to represent the concepts existing in the metamodel. In the sequel, we design a textual editor based on the Eclipse platform to allow users to create and manipulate OntoUML models using the proposed DSL. The goal is to create a textual language that represents the concepts of the OntoUML language and provide a tool needed for its use. Therefore, in this work we propose a concrete sintax to the OntoUML language and provide a tool which support the editing, syntax validation and transformation of models.

keywords: OntoUML. domain specific language. Eclipse. Xtext.

Contents

1 INTRODUCTION 7
1.1 Background 8
1.2 Motivation and Goals 9
1.3 Approach and Structure 9

2 MODELING INFRASTRUCTURE 11
2.1 Previous infrastructure 11
2.2 Our infrastructure 13

3 THE UFO REFERENCE METAMODEL 14
3.1 The Eclipse Modeling Framework 14
3.2 Unified Foundational Ontology on Ecore 15

4 TEXTUAL CONCRETE SYNTAX 16
4.1 Xtext 16
4.2 Requisites 16
4.3 Grammar 16
4.4 Syntactical constraints with CompleteOCL 16

5 TEXTUAL EDITOR 17
5.1 Implementation 17

6 TRANSFORMATIONS 18

Conclusions 19

Bibliography 20

1 Introduction

Information is an abundant, ubiquitous and precious resource. Most of the information available to us is in computer-based forms, such as files and databases. Thus, the task of computer scientists is to develop theories, tools and techniques for managing this information and making it useful. To use information, one needs to represent it, capturing its meaning and inherent structure. Such representations are important for communicating information between people, but also for building information systems that manage and exploit this information in the performance of useful tasks (MYLOPOULOS, 1998). A particular important activity to arrive at such representations is conceptual modeling, which is defined as "the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication. Moreover, it supports structuring and inferential facilities that are psychologically grounded. After all, the descriptions that arise from conceptual modeling activities are intended to be used by humans, not machines" (MYLOPOULOS, 1992).

As clearly stated by CARRARETTO: (Is it bad to quote someone else's introduction on what should be my own introduction?)

> As any pragmatic subject, conceptual modeling can profit from tool support to complete its purpose. Since conceptual models are concrete artifacts, they must exist in the physical world. Therefore, in order to obtain such artifacts, users primarily need a tool for creating and editing the constructs of a modeling language. Besides editing, there is a variety of manipulations that can be done with a model (henceforth, model manipulation) such as analysis of its content, syntactic and semantic validation, interchange between conceptual modeling tools, persistence in different formats, generation of textual documentation, transformations to different modeling languages, etc.
>
> Although it is possible to create conceptual models using a pen and a piece of paper (or, perhaps, cave walls and red ochre), users desire the usability and flexibility provided by computer-based tools. However, even those types of tools can be built in an ad hoc manner. For instance, many of the model manipulation activities can be done in different tools that do not share any implementation aspect and, as a consequence, lack integration and reusability. Such situation impacts both users and developers. On one hand, users cannot perform their modeling activities easily, since there is no integration between tools. On the other hand, developers perform extra tasks every time a new tool to support some modeling activity has to be developed, because there is no reusability.

We address these issues in this work by providing an organized infrastructure for conceptual modeling that integrates many modeling activities, such as model editing and manipulation, and promotes reusability of modeling components. More specifically, we build on top of Carraretto's work (CARRARETTO, 1990) to give tool support for a well-founded conceptual modeling language, namely OntoUML (GUIZZARDI, 2005), providing:

- a reference metamodel to be the core of model manipulation;
- a textual language to describe OntoUML models;
- a textual editor for constructing models, and also
- syntax validation by means of OCL syntactical constraints.

Thus, we provide a modeling infrastructure for OntoUML.

1.1 Background

Abstractions of a given portion of reality are constructed in terms of concepts, i.e., abstract representations of certain aspects of entities that exist in that domain. *Conceptualization* is the set of concepts used to articulate abstractions of state of affairs in a given domain. The abstraction of a given portion of reality articulated according to a domain conceptualization is termed here a *domain abstraction*. Conceptualizations and domain abstractions are abstract entities that only exist in the mind of the user or a community of users of a language. In order to be documented, communicated and analyzed, these entities must be captured in terms of some concrete artifact. The representation of a conceptual model is named here a *model*. Moreover, in order to represent a model, a *modeling language* is necessary. The relation between conceptualization, domain abstraction, modeling language and models is depicted in figure 1 below. (CARRARETTO, 1990) and (GUIZZARDI, 2005).

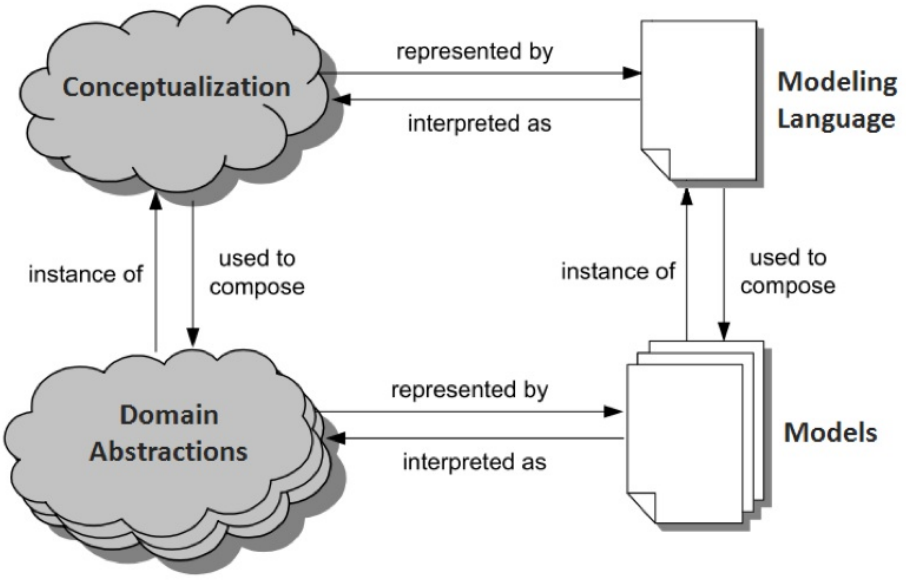

Figure 1 – Relation between conceptualization, domain abstraction, modeling language and models

According to Harel e Rumpe (2000, p. 3), "A *language* consists of a syntactic notation (syntax) which is a possibly infinite set of elements that can be used in the communication, together with their meaning (semantics)."

The syntactic notation can be divided in two parts: the *abstract syntax* and the *concrete syntax*. The abstract syntax is responsible for defining the elements of the language and the rules for relating them in valid expressions. The concrete syntax is responsible for defining a concrete representation of each element and relation previously defined so they can be manipulated to create valid expressions. (HAREL; RUMPE, 2000)

A semantic definition for a language, or simply a *semantics*, consists of two parts: a *semantic domain*, and a *semantic mapping* from the syntax to the semantic domain.

Similarly to Guizzardi's definition of *conceptualization*, Harel and Rumpe define the *semantic domain* as the specification of "...the very concepts that exist in the universe of discourse. It is an abstraction of reality, describing the important aspects of the systems that we are interested in developing. It is also a prerequisite for comparing different semantic defitions."

Although, the semantic domain is defined for describing the meaning of a notation, the definition

of the semantic domain is normally independent of the notation. This allows to "reuse" the semantic domain for other notations. (HAREL; RUMPE, 2000)

A *foundational ontology* is a representation of theories that describe knowledge about reality in a way, which is independent of language, of particular states of affairs (states of the world), and epistemic states of knowledgeable agents. In other words, a foundational ontology is a domain-independent philosophically and cognitively well-founded system of real-world categories. It deals with formal aspects of objects irrespective of their particular nature, including concepts from theory of parts, theory of wholes, types and instantiation, identity, dependence, unity. Furthermore, a foundational ontology can be used to provide real-world semantics for general conceptual modeling languages, and to constrain the possible interpretations of their modeling primitives. (CARRARETTO, 1990)

Now, we are able to contextualize this work in terms of the concepts previously presented. In this work, we provide a full implementation of a modeling language, containing both its abstract syntax (metamodel) and concrete syntax (textual DSL). Despite having different abstract and concrete syntaxes, the proposed language is compatible with the OntoUML language because they share the same semantic domain (conceptualization): a unified foundational ontology (UFO) defined in (GUIZZARDI, 2005).

1.2 Motivation and Goals

Although tool support for OntoUML conceptual models has already been developed in (CARRARETTO, 1990), where a tool for building and validating the syntax of OntoUML conceptual models were provided - as well as a *reference metamodel* for OntoUML, in our work we address some problems related to the author's approach. Basically, those problems are related to his metamodel implementation.

First, it uses a fully-compliant UML 2.0 metamodel implementation as a foundation. Because of this, the metamodel is too polluted with UML constraints which are not strictly necessary to create well-founded conceptual models. Second, its generated plugins have dependencies which were deprecated on recent versions of Eclipse, which constrains the adoption of the tool as well as its integration with more recent tools based on Eclipse (Xtext in our case). And third, the OCL syntactical constraints are defined in terms of annotations in the same metamodel. It makes hard to visualize, manipulate and reuse the constraints.

Consequently, one goal is to develop a metamodel that: is independent of UML 2.0; is compatible with the latest versions of the Eclipse Modeling package; and has the metamodel specification and syntactical constraints separated but still integrated.

Another goal is to propose a textual concrete syntax for the OntoUML language and provide tooling to manipulate models specified in this language.

1.3 Approach and Structure

In order to achieve our goals, we use the tools encompassed in the Eclipse Modeling Project. The Eclipse Modeling Project focuses on the evolution and promotion of model-based development technologies within the Eclipse community by providing a unified set of modeling frameworks, tooling, and standards implementations [built on top of the Eclipse software development environment][1].

Another important aspect is that Eclipse has an extensible plug-in system that allows us to provide new functionalities on top of the runtime system. Moreover, Eclipse is free and open source software.

[1] http://www.eclipse.org/modeling/

TODO: Talk about the structure, describing what each chapter contains.

2 Modeling Infrastructure

In this chapter, we present the general aspects of our modeling infrastructure by describing its elements, the technologies in use and their relations.

First, we assess the alleged suitability to reuse of the reference metamodel proposed by (CARRARETTO, 1990). Then we document the problems faced during this assessment and identify some properties that are desirable to a metamodel indended for this level of reuse. Finally, we propose our new infrastructure.

In addition, we provide general details about the tools used to implement the infrastructure.

2.1 Previous infrastructure

** make my point and include excerpts from my first frustration[1]

** stress the properties in primeontouml that made it easy and suitable for reuse.

Before our work, tool support for OntoUML was already provided in both (BENEVIDES, 1990) and (CARRARETTO, 1990). Each work provide an editor for OntoUML, built on top of Eclipse software development environment.

In this context, an editor is a software that allows the user to manipulate the concrete syntax of the language by relating the elements that compose it according to the abstract syntax specification, also called the metamodel of the language.

That said, an editor is responsible for providing to the user an interface to manipulate instances of the metamodel (also called, models) by means of a concrete syntax.

In (CARRARETTO, 1990), a fully-compliant UML 2.0 metamodel is proposed to address the problems existing in (BENEVIDES, 1990) and also "... be used as a reference metamodel for any model manipulation purposes (including abstract syntax validation).". However, Carrareto's work also carries its own issues with respect to reuse.

** Should we mention that we tried to create a concrete syntax for this reference metamodel and describe how we did it, and THEN talk about the problems we found?

First, the reference metamodel is based on the Eclipse implementation of the UML Superstructure (included in EMF 2.5) metamodel. That is, the metamodel was built in terms of concepts from the metamodel of UML.

For example, every association in UML has properties such navigability and ownership which are defined by a combination of three meta-properties (ownedEnd, memberEnd and navigableOwnedEnd). "A navigable end of an association means that it is intended to be traversed at runtime from objects participating in links at the other end(s) of the association to objects participating in links at the navigable end. This is independent of ownership of the end." (OMG, 2006b) Such concepts are closely related to implementation details and have no relevance in the field of strictly conceptual models. This fact also imposes some restrictions to the concrete syntax due to the way the Xtext grammar definition relates to the language metamodel, as can be seen in the following example:

[1] https://github.com/juniorz/ontouml_editor/

```
role Parent : Person { }
role Offspring : Person { }
relator Registration { }

mediation M1 {
  property m11 (Registration);  //memberEnd
  property m12 (Offspring);     //memberEnd
  memberEnd m11, m12;
}

mediation M2 {
  property m21 (Registration);  //memberEnd
  property m22[1,2] (Parent);   //memberEnd
  memberEnd m21, m22;
}
```

Both mediations (*M1* and *M2*) are simply associations between the roles and their relator but, due the presence of the extra ownedEnd and memberEnd meta atributes (inherited from the UML metamodel) and the fact that the syntactical constraints were written in terms of these same attributes, forced the mediation syntax to hold a notation excessively verbose.

By specifying the concepts of the unified foundational ontology without referring to UML concepts, we were able to represent the same concept more clearly:

```
RelatorUniversal Registration {
  mediates Offspring[1..1];
  mediates Parent[1..2];
}
```

Second, it uses a technology to support the usage of OCL constraints which requires the constraints to be embedded into the metamodel by the means of EAnnotations. Then, these EAnnotations are used by EMF to enhance the model validator.

Once again, due the way the Xtext editor delegates validation to the EMF model and the lack of features in the OCLInEcore implementation, the generated error messages were not as clean and useful as desired, to the extent of misleading the user regarding the cause of the error.

** Include a screenshot

In addition to that, the OCL plugin in use is deprecated in Eclipse versions Helios and later and the provided workaround[2] compromises the installation, making impossible the integration of the tool with other components from the Eclipse Modeling Project.

We also wanted to use custom validation messages for OCL and that was not possible with the Indigo Release (or older).

We conclude that, as a modeling language, the reference metamodel is not a true representation of the unified foundational ontology (the conceptualization it represents) because it is also polluted with concepts from the UML 2.0 metamodel used as its foundation. Such fact compromises its suitability to reuse, as stated by the (CARRARETTO, 1990) in his work.

[2] http://wiki.eclipse.org/MDT/OCL/FAQ#How_do_I_workaround_org.eclipse.emf.ocl_deprecation_in_Helios_and_later.3F

2.2 Our infrastructure

** Outline how our infrastructure is organized and how it relates with previous works. ** Mention briefly what piece of tech is involved in every part of the infrastructure.

Essentialy, draw a diagram with: 1. Abstract syntax (metamodel + OCL constraints) 2. Concrete syntax (Xtext Grammar) 3. Editor (Xtext Editor) 4. Reference OntoUML and transformations

3 The UFO reference metamodel

In this chapter, we define the abstract syntax of our language by means of a metamodel. First, we present the technology used in its construction, then we explain the metamodel itself. Finally, we discuss the differences between our metamodel and the former metamodel defined in a previous work.

3.1 The Eclipse Modeling Framework

Eclipse is an open source software project, the purpose of which is to provide a highly integrated tool platform. The work in Eclipse includes a core project, which includes a generic framework for tool integration, and a Java development environment built using it. Other projects extend the core framework to support specific kinds of tools and development environments.

The Eclipse Modeling Project is the focal point for the evolution and promotion of model-based development technologies at Eclipse. At its core is the Eclipse Modeling Framework (EMF) (BUDINSKY et al., 2009) which provides the basic framework for modeling.

The EMF exploits the facilities provided by Eclipse to make it possible to relate modeling concepts directly to their implementations, thereby bringing to Eclipse — and Java developers in general — the benefits of modeling with a low cost of entry. EMF is a framework and code generation facility that allows to define a model in a specific format (i.e., Java interfaces, UML diagram, or XML Schema) and then generate any of the others and also the corresponding implementation classes in an automated fashion.

Another key contribution of EMF to the Eclipse's modeling landscape is the Ecore metamodeling language: the standard language for defining metamodels in the Eclipse project. A simplified subset of Ecore is shown in figure 2.

An *EClass* is used to represent a modeled class: it has a name, zero or more attributes, and zero or more references. *EAttribute* is used to represent a modeled attribute. Attributes have a name and a type. *EReference* is used to represent one end of an association between classes. It has a name, a boolean flag to indicate if it represents containment, and a reference (target) type, which is another class. And, finally, *EDataType* is used to represent the type of an attribute. A data type can be a primitive type like *int* or *float* or an object type like *java.util.Date*.

The most important benefit of EMF, as with modeling in general, is the boost in productivity that results from automatic code generation. From a single Ecore model it is possible to generate Java

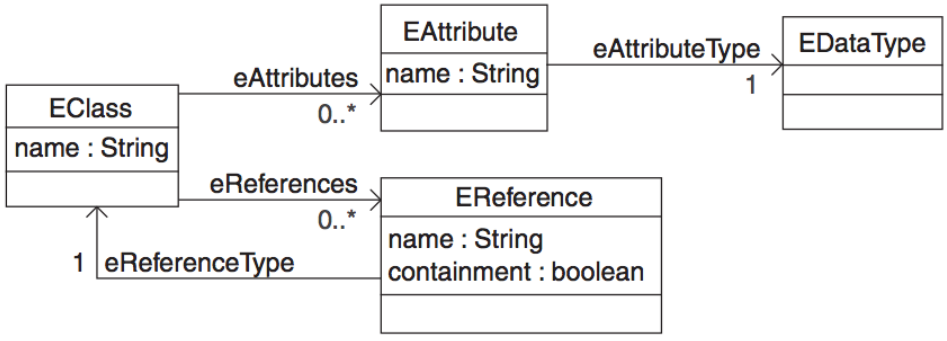

Figure 2 – A simplified subset of the Ecore metamodel (BUDINSKY et al., 2009)

classes and interfaces for every EMF metaclass.

As part of the EMF, the *EMF Validation Framework* provides model validation capabilities by supporting the evaluation of invariants and constraints via a validator that is invoked at important moments by an application or at a user's discretion. (STEINBERG, 2014)

Prior to EMF version FOO, these invariants and constraints had to be defined via operations and annotations, respectively. (BUDINSKY et al., 2009, Chapter 18) Then they would be defined as methods during the code generation step and finally be manually implemented in Java.

The usage of *Java Emitter Templates* (JET) made possible to integrate the MDT OCL and the EMF to directly obtain the Java code for validation, without any post-generation custom code (DAMUS, 2007). However, this approach was deprecated with the deprecation of EMF version RAH.

The version BAR of EMF[1] introduced, the concept of a Validation Delegate, in order to delegate evaluation of invariants and constraints to an external engine based on a well-defined interface.(STEINBERG, 2014) This validation delegate became the standard for extending EMF metamodel validation capabilities.

OCLInEcore uses the EMF delegate mechanisms to extend Ecore models with the OCL engine.

CompleteOCL allows to extend the metamodel with a external OCL document. We used (If I'm not mistaken) this CompleteOCLEObjectValidator.

In this work, we use the version 4.2 of the Eclipse platform (codename Juno) and EMF at its version FOO.

3.2 Unified Foundational Ontology on Ecore

** aka PrimeontoUML - we need a better name for this, something like Carraretto's reference metamodel. ** What about UFO (reference) metamodel? foundational metamodel, maybe? ** That is because OntoUML is a UML 2.0 Profile that incorporates the foundations captured in the UFO.

We created the UFO reference metamodel in Ecore

[1] part of the Eclipse Helios release

4 Textual Concrete Syntax

4.1 Xtext

4.2 Requisites

4.3 Grammar

4.4 Syntactical constraints with CompleteOCL

The Object Constraint Language (OCL)(OMG, 2006a), is a formal language used to describe expressions on UML models.

These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model. Note that when the OCL expressions are evaluated, they do not have side effects (i.e., their evaluation cannot alter the state of the corresponding executing system).

5 Textual Editor

5.1 Implementation

6 Transformations

Conclusions

Bibliography

BENEVIDES, A. B. *A Model-based Graphical Editor for Supporting the Creation, Verification and Validation of OntoUML Conceptual Models*. Monografia (M.Sc. thesis) — Centro Tecnológico, Universidade Federal do Espírito Santo, Vitória, Brazil, 1990.

BUDINSKY, F. et al. *EMF: Eclipse Modeling Framework*. 2nd revised. ed. Boston: Addison-Wesley Professional, 2009. ISBN 9780321331885.

CARRARETTO, R. *A Modeling Infrastructure for OntoUML*. 91 f. Monografia (Graduação) — Centro Tecnológico, Universidade Federal do Espírito Santo, Vitória, Brazil, 1990.

DAMUS, C. W. *Implementing Model Integrity in EMF with MDT OCL*. 2007. Disponível em: <http://www.eclipse.org/articles/article.php?file=Article-EMF-Codegen-with-OCL/index.html>. Acesso em: 2014-02-14.

GUIZZARDI, G. *Ontological Foundations for Structural Conceptual Models*. Enschede, The Netherlands: University of Twente, 2005.

HAREL, D.; RUMPE, B. *Modeling Languages: Syntax, Semantics and All That Stuff Part I: The Basic Stuff*. [S.l.], 2000.

MYLOPOULOS, J. Conceptual modelling and telos. In: LOUCOPOULOS, P.; ZICARI, R. (Ed.). *Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development*. New York: Wiley, 1992.

MYLOPOULOS, J. Information modeling in the time of the revolution. *Information Systems*, v. 23, p. 3–4, June 1998.

OMG. *Object Constraint Language*. 2006. Version 2.0.

OMG. *Unified Modeling Language: Infrastructure Specification*. 2006. Version 2.0.

STEINBERG, D. *EMF/New and Noteworthy/Helios*. 2014. Disponível em: <https://wiki.eclipse.org/EMF/New_and_Noteworthy/Helios#Support_for_Validation_Delegates>. Acesso em: 2014-02-14.