An Ontology proposal for Semantics Checking on Imposed Restrictions by Application Design in Component-based Software Assembling

Manuel Corona-Perez[1], Edgar Castillo-Barrera[1]

[1]Departamento de Tecnologías de Información
CUCEA – Universidad de Guadalajara
Periférico Norte 799 – Mod. L305
Los Belenes 45100
Zapopan, Jalisco, México
manuelcorona@cucea.udg.mx, ecastillo@uaslp.mx

Paper received on 20/08/10, Accepted on 04/10/10.

Abstract. Component-based Software has demonstrated to be a good solution in Software reuse, but this solution fails in guarantee Semantics imposed by the application design. We propose an Ontoloy-based Semantics model to check restrictions to support changes from one application to another. This semantic checking is be developed as a process during the Software Component assembling design which is called Componization.

Keywords: component-based software, ontology, semantics checking, componization

1. Introduction

Component-based software development (CBSD) [5] [1] has shown to be a good solution in software reuse and this help to build applications not necesary from the scratch. We build a new applications using executable pieces of software previosly developed and used in other applications builded from third parties. This software is known as Components Off The Shelf (COTS) [8]. When we reuse Software Components and we assemble them into an application framework we need to adapt them with the aim to get the software solution that we are building. This process of adaptation is called Composition. But not only the goal is to adapt these components, we need also to validate operational semantics, if we want to guarantee that these operations will be satisfied.

Some restrictions come from intercomponent operations, defined by *contracts*, and other restrictions are defined from the System Architect of the application.

To validate these operational restrictions, we propose an *Ontology-based Semantics.*

An Ontology is an explicit specification of a conceptualization [4], It set up semantics on specific domain knowledge in a formal and all-porpouse way [2]. We need an Ontology to represent components and the operational semantics between components at the user level(System Architect). To represent the meaning and consistency of things we use *Semantics*, and an Ontology is one o the best way to model semantics. Our purpose is an *Ontology-Based Semantics* [6] to model semantic validation on Component-based Software Assembling.

2. Context Semantics Model in Components Assembling

When we use Components in a Software Applications, we must guarantee that some constraints previously imposed for the Application must be fulfilled.

This way we can focus in three different ways where semantics may occur. The first one is *Component to Component Assembling*, the second one is *Component to Framework Assembling*, and the third one is *The Whole application*.

Semantics must occurs between the Components, between the Component and the Framework. In essence we call this, *Referential Semantics Componization*. Furthermore the semantics constraints for the application as a whole are called *Integral Semantics Componization*.

2.1. Referential Semantics Componization

Let´s A and B be two Components that are working together and F an application Framework for which we define the following set of rules:

$$\text{a) } R_A^B = \{r_1, r_2, r_3, ..., r_n\}$$

Where **a)** is the set of semantic restrictions between Components A and B.

$$\text{b) } R_A^F = \{s_1, s_2, s_3, ..., s_n\}$$

Meanwhile **b)** is the set of semantic restrictions between Component A and the Framework of the application.

This means that R_A^B and R_A^F are two restriction Sets imposed between A and B and between the Framework and then Component A, we do not care who is imposing any of the restrictions.

Example:
A= Thermometer Component which process temperatures in Celsius and Kelvin Scales.

B= Control Heat Component whose task is to mantain a specific temperature in an oven.
F= Framework in which a Metals Alloy Application is being implemented.

$$R_A^B = \{PlumbAlloyTemperature, IronAlloyTemperaturte, CopperAlloyTemperature\}$$

R_A^B means the necesary temperatures restrictions of fluid melting metals in a Metals Mix Process to produce a specific Alloy.

2.2. Integral Semantics Componization

Let's P a Metals Alloy Application, A and B the Components before defined. Such as:

$$R_I^P = \{t_1, t_2, t_3, ..., t_n\}$$

Where R_I^P represents semantics restrictions set that the application P must be fulfilled.

Example:

P = Metals Alloy Application

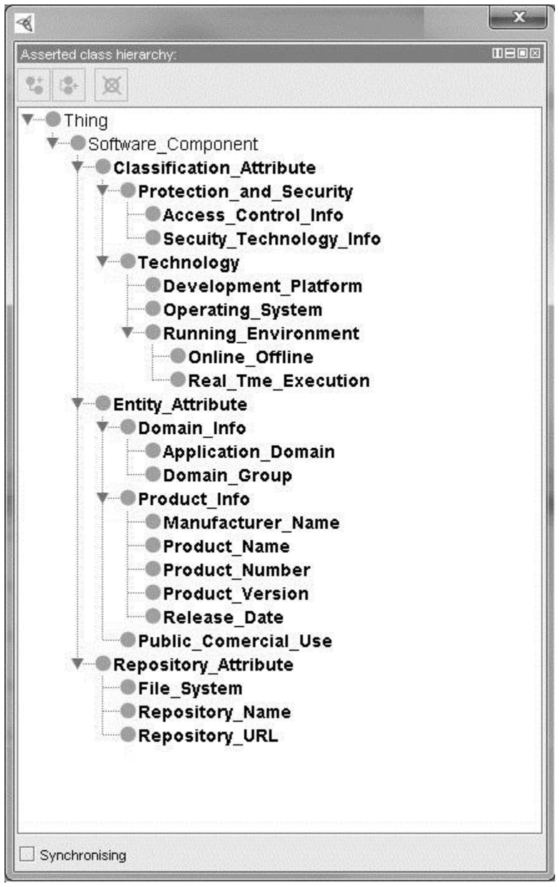
Then:

$$R_I^P = \{MetalsAlloyTemperature, SequenceOfMixingMetals, IronThemperatureInAlloy, \\ PlumbThemperatureInAlloy, CopperThemperatureInAlloy\}$$

Which ones represents restrictions that the application impose during the Metals Alloy Process.

3. Proposed Ontology for the Context Semantics Model

An Ontology shows to be a good way to represents Semantics. In this case, we are working on building an Ontology that implements the Context Semantics Model. The first step is to describe an Ontology for a Software Component similar to the one proposed by Dong, Liu, He, He [2] in Figure 1. Using this Component Ontology we can relate one component to another and also among Components and the Framework.



```
Thing
  Software_Component
    Classification_Attribute
      Protection_and_Security
        Access_Control_Info
        Secuity_Technology_Info
      Technology
        Development_Platform
        Operating_System
        Running_Environment
          Online_Offline
          Real_Tme_Execution
    Entity_Attribute
      Domain_Info
        Application_Domain
        Domain_Group
      Product_Info
        Manufacturer_Name
        Product_Name
        Product_Number
        Product_Version
        Release_Date
      Public_Comercial_Use
    Repository_Attribute
      File_System
      Repository_Name
      Repository_URL
```

Figure 1: A Software Component Ontology

Using the component methods we can implement the Context Semantics model and generate a complete semantic verification process. We can also consider the semantic context for the specific application where the Software Components are being used.

The aim in this task must be mapping each relation restriction to one relation operation. This should be done by following the pre-conditions and post-conditions imposed by *the interfaces*. Also we need to map the methods in the Method class and all its parameters in the same Ontology. Each element belonging to the relation restricions set in the Context Semantics Model must have a relation between the Method and the Parameters where Components are interacting.

Our proposed contribution as a solution in this work will define relation restrictions that were explained in the Context Semantics Model. By Building relations on the generated Ontology, we establish the operational semantics among methods that each component has defined in its interfaces. We use another Ontology that describe this kind of relations and here it shows where we are going to apply our semantic model.This other Ontology description was done by Linhalis, de Mattos, de Abreu. [3] Figure 2 and is used only to show where our Semantic model will be implemented.

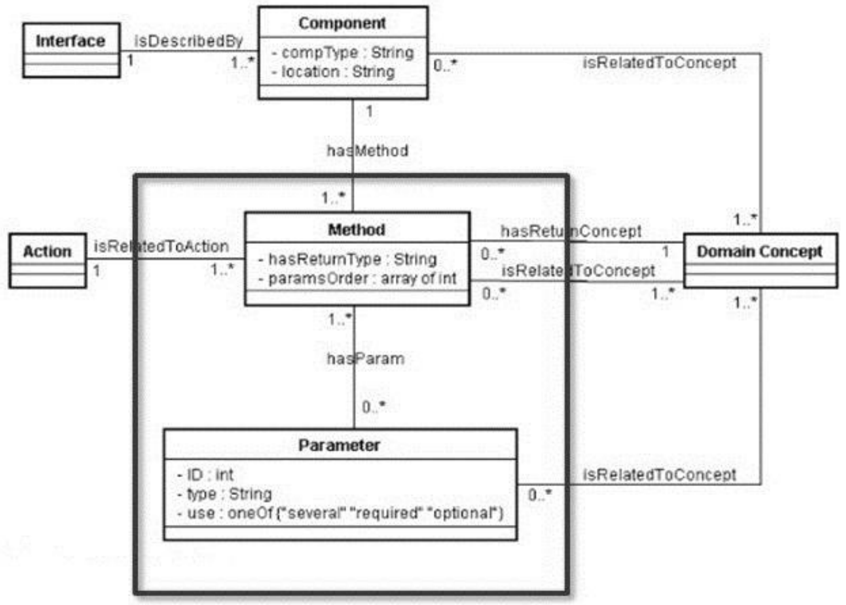


Figure 2: Context Semantics Model is implemented among the Components using Method class and the Parameters class relations

4. Conclusion and Future work

We are building the Context Semantics Model and some Ontology classes in **Protégé** [7] and at the same time we have the abstraction of the Model and the solution domain in a sort to implement relation restriction sets on the semantic context of the application. Using Software component not only requieres sintax and semantics over the components plugins, we are adding a context details because to use components, we need to guarantee additional restrintions when we are using

from one aplication to another. Our work under development is to get a complete Ontology to represent our model doing the relations between the classes and to implement tools to do the Model transformations in the Component-based Software Assembling.
We are developing this work with three main advantages in mind and these are:

- To automate the semantics checks, generating a *glue code* that helps in the implementation of the application.
- To guarantee consistency on the execution of the application, lowering the number of exception errors.
- To get a better performance and realibility of the whole application *per sé*.

Under this perspective, we need to build an environment to run test cases to probe our model. In future papers we will show this results.

References

1. Clemens Szyperski, Dominik Gruntz, Stephan Murer. *Component Software, Beyond Object-Oriented Programming*. Second Edition, Addison-Wesley, 2002.
2. Dang Song, Wudong Liu, Yangfan He and Keqing He. *Ontology Application in Software Component Registry to Achieve Semantic Interoperability*. Wuhan University, China, 2005.
3. Flavia Linhalis, Renata Pontin de Mattos Fortes, Dilvan de Abreu Moreira. *OntoMap: an ontology-based architecture to perform the semantic mapping between an interlingua and software components*. Science Computing and Mathematics Institute (ICMC), University of SÃ£o Paulo (USP), 2009.
4. Thomas R. Gruber. *A Translation Approach to Portable Ontology Specification*. Stanford University, USA, 1993.
5. Ivica Crnkovic, Magnus Larsson. *Building Reliable Component-BasedSoftware Systems*. Editors, Artech House), 2002.
6. Mihai Ciocoiu, Dana S Nau. *Ontology-Based Semantics*. University of Maryland, USA, 1999.
7. Stanford University, University Manchester. *Protege, A free open source Ontology Editor*. http://protege.stanford.edu/, 2006.
8. Xiong Xie, Weishi Zhang. *Research on Software Component Adaptation Based on Semantic Specification*. Department of Computer Science and Technology, Dalian Maritime University, Dalian, China, 2007.