Knowledge Capitalization in a Component-Based Software Factory: a Semantic Viewpoint

Francisco-Edgar Castillo-Barrera[1], Carolina Médina-Ramírez[3], Hector A. Duran-Limon[1]

[1] Universidad de Guadalajara, Zapopan, Jalisco, México
ecastillo@uaslp.mx,hduran@cucea.udg.mx
[2] Department of Electrical Engineering, Universidad Autónoma Metropolitana, Mexico City, México
cmed@xanum.uam.mx

Abstract. Recently, Ontologies have boomed as artifacts to represent domain knowledge and as an important component in specific applications helping decision-making. In software engineering domain, ontologies have been employed to manage software components (classifying, retrieving, matching et all), and this approach has been deemed as an effective way for capturing and using the knowledge of the software components on the retrieving system and matching process, before and after it. This knowledge will be used by new developers gaining time in the development of new projects and in consequence it implies a reducing costs for training. In this paper, we describe a software component ontology for knowledge capitalization. We use an example and a prototype (called Chichen Itza) to show the feasibility of our approach.

Keywords: Knowledge Capitalization, Ontology, Software Components, Semantic Technique, CBSE.

1 Introduction

Personnel rotation or departure from the project usually have a big impact in the time for finishing a project. The time and money required for training new personnel to acquire experience imply to increase the project costs. The new Knowledge obtained for each part of the software component assembled, can be stored in an ontology as artifact. This knowledge can be used by new developers or new members of the project to reduce time. In consequence, the company decreases costs. The software component knowledge (company, author, functional and no-functional, requirements, etc) can be described using the concepts belonging to the ontology. This description can be stored in a XML file and asociated to the binary file of the software component. The Ontology written in OWL-DL can be exchanged among different systems and component-based applications. Besides, the input domain model is classified and saved using a Meta-Ontology of domains called *MetaDomOnto*. When a new input domain model will be introduce in the system, the developer can check if there is a related model or

the same model in the database. In both of these cases, he can found terms or atributes in his vocabulary domain and he can use it in his project. This allows to the company to reduce the costs of the project. Besides, using a reasoner for checking the consistency of the input domain model, brings an economic added value to the company by means of an automatic way for verifying its model. This action is made during the Architectural Design [1] in a high abstraction level. This action prevent to the company to buy software components that do not match.

The rest of the paper is structured as follows. In Section 2 we give the state of the art of capitalization on software component. In Section 3 we briefly explain Component-Based Software Engineering and component matching for assembling purposes. Section 4 describes our semantic approach for Knowledge Capitalization in a Component-Based Software Factory. In Section 5 we show the feasibility of our technique by describing an example and a prototype called Chichen Itza. Finally, in Section 6 we conclude our work.

2 State of the art

The ontologies based on software component and matching is mostly represented by work of Claus Pahl [2] who wrote an ontology for software component matching. Pahl's ontology is oriented for Web Services, our ontology is made for a software factory framework which support Pipe-and-Filter architecture styles. Other researchers such as Yutao, Keqing, Liu and Jingbai Tian [3] have developed ontologies based on a standard (No.19763), which are focus to use a Grid-Oriented Plataform. The most closely related work was made by Nkambou [4]. In this work, the author describes a CBR-Based tool (called CIAO-SI) dedicated to the capitalization of development experience/knowledge within a software development company, using tasks ontologies. Another work about Knowledge Capitalization was made by Rodriguez-Rocha et al. [5]. Her work is mainly focus on the Knowledge Capitalization in the Automotive Industry. She developed an ontology based on the ISO/TS 16949 Standard.

3 Component-Based Software Engineering and Software Components

Crnkovic and Larsson [6] define Component-Based Software Engineering (CBSE) *as an approach to software development that relies on software reuse.* The aim of the CBSE is the rapid assembly of complex software systems using pre-fabricated software components. CBSE combines elements of software architecture, software verification, configuration and deployment. CBSE is a process that emphasizes the Design and Construction of systems using reusable software components. Reuse is a primary concept to software development, as it reduces development effort, time and *cost*. We want to increase the *reuse* and to decrease the *cost* with our proposal. A software component is an existing piece of software which

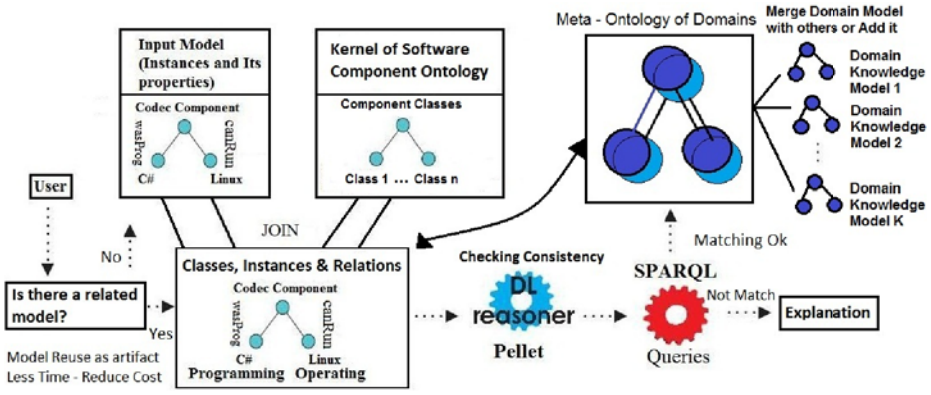

Fig. 1. Semantic Verification and Capitalization Process

can be deployed independently and it is subject to composition by third party. In this work, we consider the following definitions: *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies* [7] and *A component is a reusable unit of deployment and composition that is accessed through an interface* [6]. Our semantic viewpoint of a software component is that it is a software unit consisting of a name, an interface and code, that can be exploted and reused in order to Knowledge capitalization purposes.

4 Knowledge Capitalization in a CBSF: a Semantic Approach

4.1 Semantic Matching and Verification Process

We define matching among components when all contracts among them have satisfied the post-conditions and pre-conditions which gives the functionality expected based on requirements establish in their contracts. Semantic verification is the process which uses an Ontology and Semantic Technologies (SPARQL queries) to guarantee compliance with contractual agreements. The semantics of an operation are described with a contract. An important aspect of Components to consider during the matchig is Identity and State. A component may have an identity and state, represented by objects. Component connection and interaction are based on plugging components When a client requests services of a provider glue code is needed.

4.2 An Ontology for Software Components

An Ontology [8][9][10][11] is defined by Gruber as *a specification of a conceptualization* [8]. In other words, an Ontology defines the terms used to describe

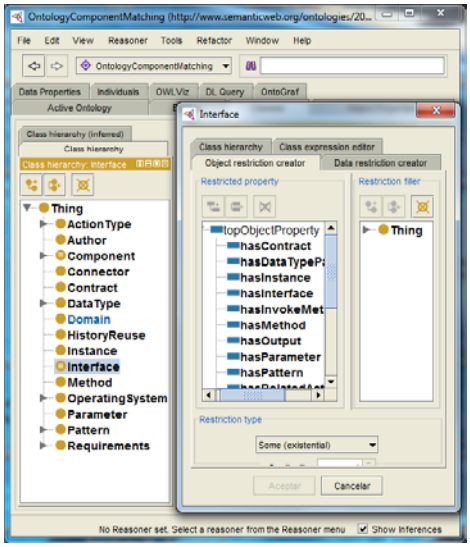

Fig. 2. A Segment of Software Component Ontology in Protégé Editor

and represent an area of knowledge, also is the model (set of concepts) for the meaning of those terms, thus defines the vocabulary and the meaning of that vocabulary, are used by people and applications that need to share domain information. An Ontology is a formal representation of knowledge which allows us to obtain information by means of the checking of its consistency, using Reasoners [12]. A Software Component Ontology was created for capturing and verifying information about the input domain models during the Architectural Design [1]. This ontology consisted of 50 classes, 19 Object Properties, 15 Data Properties. The notation n3 is used by the ontology, because is a valid RDFS and OWL-DL notation. The Ontology use RDFS and OWL-DL language [13]. They are fundamentally based on descriptive logic languages. The Protégé editor [14], was used to visualise the ontology and its RDF graph. OWL-DL and Protégé are the current recommendation of the W3C [15]. The OWL-DL ontologies have the ability of:

a) Automatic reasoning a) To be distributed through many systems
b) Scalability to the needs of the Web
c) Compatibility with web standards for accessibility and internationalization
d) Opening and extensibility

We proposed an ontology called *Kernel Ontology* which has the minimum concepts contained in the software components ontologies analyzed for this work. Kernel Ontology is built by means of classes and relations among concepts. These concepts and classes correspond to the specification of an abstract data type and a set of methods that operate on that abstract data type. Each method is spec-

ified by an interface, type declarations, a pre-condition, and post-condition [6]. In addition, there is an explicit class about *Contracts* and two types of interfaces (*provided* and *required*). The interface of a method describes the syntactic specification of the method. The typing information describes the types of input and output or both parameters and internal (local) variables. All of the above is represented in our ontology (class Type, class Parameter, etc.). The most important part to consider in our ontology are the Conditions (Pre and Post). Because we have to verify that those conditions are met. The Pre-condition describes the condition of the variables prior to the execution of the method whose behavior is described by the Post-condition. The Ontology is showed in Fig.2.

4.3 Using The Pellet Reasoner

Pellet [12] is an open-source Java based OWL-DL reasoner. In our verification process we use Pellet for checking the consistency of the ontology and classify the taxonomy. Pellet gives explanation when an inconsistency was detected. Restrictions can be expressed into an ontology. For instance, the following code verify that one component has at least 1 interface.

```
:Component rdfs:subClassOf
    [ a owl:Restriction ;
      owl:onProperty :hasInterface ;
      owl:cardinality 1 ].
```

An interesting property of the ontology used in this work is a blank node. It is a node in an RDF graph representing a resource without URI or literal. We used it as variable. If we put the same blank node, the result for this node has to be the same. In our example below, _:c1, _:c2, _:c12, _:i1 and _:i2 are blank nodes (variables). The example shows How to know about the contract between two components and their interfaces.

```
_:c1 :hasContract  _:c12 .
_:c2 :hasContract  _:c12 .
_:c1 :hasInterface _:i1 .
_:c2 :hasInterface _:i2 .
```

A difference with Logic Programming Paradigm, we can check types using ontologies. Besides, in the matching process subtypes can be accepted as parameters. See code below.

```
:Int a owl:Class .
:ShortInt rdfs:subClassOf :Int .
```

The disjointWith property allow to verify restrictions in the input model. For example a component made in .Net can not run in the Linux operating system. Defining disjoint properties is also possible [16].

```
PREFIX : <http://ejemplo.org#>
SELECT DISTINCT ?comp1 ?comp2 ?instancecomp1 ?instancecomp2 ?method1 ?method2
                ?parmet1 ?parmet2
WHERE
{
 ?comp1         :hasContract          ?contract .      # Is there contract?
 ?comp2         :hasContract          ?contract .      # Is it the same contract?
 ?contract      :useInterface         ?interface .     # What interface is used?
 ?comp1         :hasInstance          ?instancecomp1 . # is there instance for this component?
 ?comp2         :hasInstance          ?instancecomp2 . # is there instance for this component?
 ?instancecomp1 :invokeMethod         ?method1 .       # call to method
 ?instancecomp2 :invokeMethod         ?method2 .       # call to method
 ?method1       :hasNumParameters     ?numpar .        # Number of parameters method 1
 ?method2       :hasNumParameters     ?numpar .        # Number of‘parameters method 2
 ?method1       :hasParameter         ?parmet1 .       # parameter method 1
 ?method2       :hasParameter         ?parmet2 .       # parameter method 2
 ?parmet1       :hasDataTypeParameter ?type .          # type parameter method 1
 ?parmet2       :hasDataTypeParameter ?type .          # type parameter method 2
 ?parmet1       :linkParameter        ?parmet2 .       # Verifying consistency in parameters

 FILTER( ?comp1 != ?comp2 && ?instancecomp1 != ?instancecomp2 ) # Avoiding duplicated results
}
```

Fig. 3. Query to detect Contracts in SPARQL

```
:Linux rdfs:subClassOf :OperatingSystem ;
     owl:disjointWith :Windows .
```

All properties defined in the Ontology and blank nodes are checked by the reasoner (Pellet) during the consistency verification process.

4.4 Complementing the Verification with Semantic Queries

For more complex checking we can apply anothers actions such as: production rules [17]. We decided to explore semantic queries in SPARQL. The second step after the reasoner have checked the ontology consistency is to apply a SPARQL query. We decide to define a specific query which evaluates and verifies certain information on the input model. Of course, all this process is transparent, for the user. We have used Jena API [18] and Java language [19] for programming that and NetBeans IDE 7.0 [20]. SPARQL is the version of SQL for ontologies. But, we can use variables in the queries, constraints, filtering information, logic operators, if statements and more. In Fig.3 we explain each line by means of questions. Lines are linking by variables which begin with a question mark. The same name of variable imply the same value to look for in the query. The Jena API allowed us to use SPARQL queries in our framework programmed in Java language.

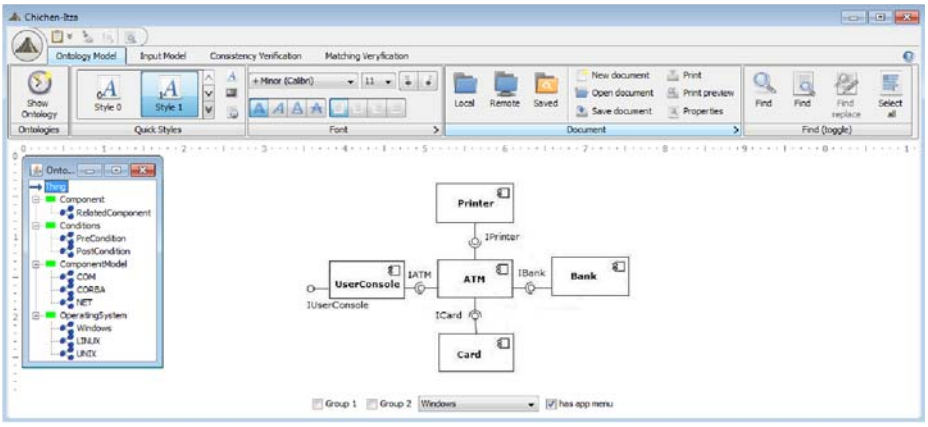

Fig. 4. UML2 ATM component system, in Chichen Itza Framework

4.5 Matching Software Components

This process, within the Chichen Itza framework, is done at very high level, using the ontologies information between one component and the second component to be matched. Each component is represented in a graphic way. That information is evaluated and after the system decides if is possible matching or not the components. In Addition, we capture the new knowledge in this new component, called "Capsule". In our framework, a Capsule has a graphical representation which is stored as a new component with its own characteristics.

5 ATM verification using Chichen Itza Component-Based Framework

5.1 Chichen Itza: a Component-Based Software Factory Framework

Chichen Itza is a software factory framework which focuses on maximising the level of reuse in two dimensions: architectural design and software components. A special focus is paid on pipe-and-filter architecture styles. The aim of this framework is allow to develop component-based applications from scratch using a friendly interface and a graphical arquitectural description language. Chichen Itza is a component-based software factory framework which focuses on maximising the level of reuse in two dimensions: architecture design and software components. Our main contributions are twofold. First, we define a framework that allows for reusing a single component-based architecture design for different component platforms. Second, our approach supports the automation of component composition in multiple component platforms. A prototype of the framework involves a visual editor of software architectures. See Fig.4. The tool makes use of the library Flamingo and the Ribbon component [21] implemented

Component	Interfaces	Methods
ATM	IATM	createSession(), locateBank(int CardNo, string Password)
Printer	IPrinter	printReceipt()
UserConsole	IUserConsole	readPIN(int CardNo, string Password), setMenu(), setMessages()
Bank	IBank	Consortium(), Withdrawal(int CardNo, string Password, int Amount)
Card	ICard	readCard(), ejectCard()

Table 1. Summary of ATM Component-Base System

in Java. The process to verify a matching among components is very easy for the user. He introduces his model into the framework by means of a file or by the menus. Chichen Itza transforms his vocabulary from a text file into an ontology instances and its relations. The instances are created from classes defined in the Software Component Ontology. See Fig.1.

In our approach we used an Automated Teller Machine (ATM) example. ATM is a machine at a bank branch or other location which enables customers to perform basic banking activities without humans (checking one's balance, withdrawing or transferring funds) even when the bank is closed. The component model used for describe the ATM system was made in Chichen Itza Framework using its graphical interface of software components, and is shown in figure 4: One example in the design phase using ATM example [22]. The input model is created by the user who selects classes and relation among concepts and he create instances. In this case the input model only has 5 software components and we can create its instances and relations among them using the Chichen Itza's menus. In Table 1 Methods, Interfaces and Components are shown.

6 Conclusions

Knowledge Capitalization in a Component-Based Software Factory (CBSF) is possible using ontologies in every part where components are used. Ontologies are usually expressed in a logic-based language (Description-Logic), enabling detailed, sound, meaningful distinctions to be made among the classes, properties and relations. Ontologies give more expressive meaning, but maintains computability. The use of an ontology permits us to search an specific component information using intelligent techniques like production rules in comparison with a classic SQL query. The queries on the ontology are simple and easy to do for all users whereas a classic SQL query in a database requires computational knowledge. In this paper we have presented and described an ontology for Classifying, Searching and Matching Software Components in a Component-Based Software Factory Framework.

References

1. Eden, A., Kazman, R.: Architecture, design, implementation. In: proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society (2003) 149–159
2. Pahl, C.: An ontology for software component matching. Volume 9. Springer-Verlag, Berlin, Heidelberg (2007) 169–178
3. Ma, Y., He, K., Liu, W., Tian, J.: A grid-oriented platform for software component repository based on domain ontology. Volume 0. IEEE Computer Society, Los Alamitos, CA, USA (2007) 628–635
4. Nkambou, R.: Capitalizing software development skills using cbr: the ciao-si system. In: IEA/AIE'2004: Proceedings of the 17th international conference on Innovations in applied artificial intelligence. Springer Springer Verlag Inc (2004) 483–491
5. Rodriguez-Rocha, B.D., Castillo-Barrera, F.E., Lopez-Padilla, H.: Knowledge capitalization in the automotive industry using an ontology based on the iso/ts 16949 standard. Volume 0. IEEE Computer Society, Los Alamitos, CA, USA (sep. 2009) 100–106
6. Crnkovic, I., Larsson, M.: Building reliable component-based software systems. Artech House computing library, Norwood, MA (2002)
7. Szyperski, C., Gruntz, D.: Component software: Beyond object-oriented programming. Addison-Wesley (2002)
8. Gruber, T.: Toward principles for the design of ontologies used for knowledge sharing. (1995) 907–928
9. Berners-Lee T.Hendler J., a.L.O.: The semantic web. (2001)
10. Fox, M.S.: The tove project towards a common-sense model of the enterprise. In: IEA/AIE '92: Proceedings of the 5th international conference on Industrial and engineering applications of artificial intelligence and expert systems. Springer-Verlag, London, UK (1992) 25–34
11. Staab S., Studer R., S.H., Sure, Y.: Knowledge processes and ontologies. Volume 16. (Jan-Feb 2001) 26–34
12. Parsia, B., Sirin, E.: Pellet: An owl dl reasoner. In: In Proceedings of the International Workshop on Description Logics. (2004)
13. W3C: Owl web ontology language (1994)
14. Eriksson, H.: Document management using protege. In: 10th Intl. Protege Conference, Budapest,Hungary (2007)
15. W3C: http://www.w3.org/consortium/. (1994)
16. Antoniou Grigoris, F.E., Frank, V.H.: Introduction to semantic web ontology languages. (2005)
17. del Río, A.C., Gayo, J.E.L., Lovelle, J.M.C.: A model for integrating knowledge into component-based software development. KM - SOCO (2001) 26–29
18. Jena: Jena a semantic web framework for java. (2000)
19. Clarke, P.J., Babich, D., King, T.M., Kibria, B.M.G.: Model checking and abstraction. ACM Transactions on Programming Languages and Systems **16** (1994) 1512–1542
20. Armstrong, E., Ball, J., Bodoff, S., Carson, D.B., Evans, I., Ganfield, K., Green, D., Haase, K., Jendrock, E., Jullion-ceccarelli, J., Wielenga, G.: The j2ee ^{TM}(tm) 1.4 tutorial for netbeans ^{TM}(tm) ide 4.1 for sun java system application server platform edition 8.1
21. Java.net: Flamingo. http://java.net/projects/flamingo/ (2010)

22. kiu Lau, K., Wang, Z.: A survey of software component models. Technical report, in Software Engineering and Advanced Applications. 2005. 31 st EUROMICRO Conference: IEEE Computer Socity (2005)