Verifying the Behavioral Contracts among Components by means of Semantic Web Techniques

Francisco-Edgar Castillo-Barrera
Engineering Faculty
Universidad Autónoma de San Luis Potosí
Dr. Manuel Nava 8, Zona Universitaria poniente
78290 San Luis Potosí, México
ecastillo@uaslp.mx

Héctor A. Durán-Limón
Department of Information Technologies
Universidad de Guadalajara
Periférico Norte 799, Mdulo L-308
45100, Zapopan, México
hduran@cucea.udg.mx

Carolina Medina-Ramírez
Department of Electrical Engineering
Universidad Autónoma Metropolitana, México
Av San Rafael Atlixco 186, Col.Vicentina
09340 Distrito Federal, México
cmed@xanum.uam.mx

Jose Emilio Labra Gayo
Department of Computer Science
Universidad de Oviedo
C/Valdes Salas s/n 33007-Oviedo, España
labra@uniovi.es

S. Masoud Sadjadi
School of Computing and Information Sciences
Florida International University (FIU)
11200 SW 8th St
Miami, USA
sadjadi@cs.fiu.edu

Abstract

Verification and validation of Component-Based systems are important tasks to do during the whole component lifecycle. In companies, verification techniques for component matching are difficult to integrate into the standard software development process because these can be time-consuming, error-prone, and require specialized expertise. In this paper we describe a semantic web framework for verifying behavioral contracts: invariants, pre- and post-conditions. In addition, we use the CORBA-IDL vocabulary with semantics for this purpose. Our approach relies on a core ontology of software components and SPARQL queries. The ontology captures the concepts, properties, relationships, requirements, and software component functionality. This is encoded using OWL DL, supported by the Pellet reasoner for checking the ontology component consistency. The Resource Description Framework (RDF) triples (describing components content in the form of subject-predicate-object expressions) are queried using SPARQL, in order to complement the matching verification process. We use case example and a prototype (a semantic framework called Chichen Itza) to show the feasibility of our approach.

1. Introduction

Crnkovic and Larsson [11] define Component-Based Software Engineering (CBSE) "as an approach to software development that relies on software reuse". The goal of CBSE is the rapid assembly of complex software systems using pre-fabricated software components. In order to achieve this aim, methods for verifying the matching among components are necessary. Such methods can be basically classified in Formal, Semi-Formal, and Informal methods. Formal methods such as Z [27] or VDM [24] require a mathematical background. For that reason, in practice it is not adopted by industry. Parnas says "paraadoxically, success stories reveal the failure of industry to adopt formal methods as standard procedures; if using these methods was routine, papers describing successful use would not be published" [22][26]. Other problems to adopt formal methods

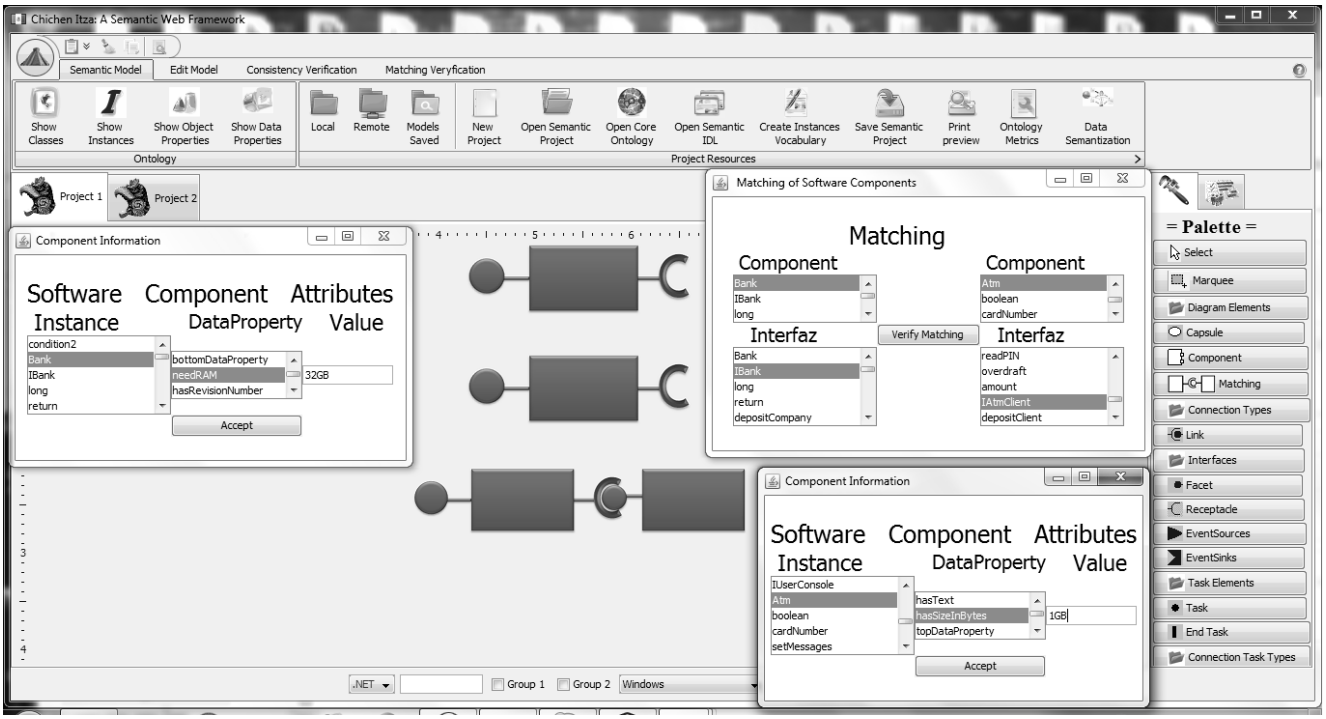

Figure 1. Visual assembled of software components in Chichen Itza framework.

are the time and the cost involved in the analysis of large-scale component-based systems [32]. For the reasons given above, formal methods require new ways to be applied. In this work, we propose a verification technique based on semantic techniques (Ontologies and SPARQL queries) in conjunction with *Chichen-Itza* framework to mitigate this problem. We propose an approach for verifying that *the contract of a required interface* matches with *the contract of a provided interface*. Our approach is able to check contract conformance of syntactic and behavioral aspects [6]. The former invoves verifying the compatibility of the signatures of a provided and a required interface. The latter is in charge of certifying that the values of the parameters are within a valid range and have a proper semantics. In this work, we only consider sequential composition in which the composed components are executed sequentially. We follow an ontology-based approach which involves a formal method but without the complexity of most formal methods. In this work, we consider the following definition: *"A component is a reusable unit of deployment and composition that is accessed through an interface"*[11]. In practice, we have noted problems related to interface incompatibility are frequent. In particular such as incompatibility with the semantics of operation parameters and interface operations (behavioral contracts [6]). We consider that the use of a semantic matching approach (a software component ontology) could help to detect interface incompatibility before the component-based system is deployed.

The rest of the paper is structured as follows. In Section 2 we present some related work. In Section 3 we briefly explain semantic web techniques and its elements (Ontologies and SPARQL queries). Section 4 describes our semantic approach for Verifying the Matching of Software Components. In Section 5 we show the feasibility of our technique by describing an example. In Section 6 we draw some concluding remarks. Finally, acknowledgments are given in Section 7.

2 Related Work

There are several works about techniques for verifying contracts, Brada [8], Mariani and Pezze [19], Tsai and Eric Y.T. Juan [31], Cernuda, Cueva et all [12]. The most closely related work about component contract verification was made by Barnett and Schulte [3]. They propose a method for implementing behavioral interface specifications on the .NET Platform using contracts to check the conformance of an implementation class and they define the AsmL specification language. In contrast with their work, our model is independent of plataform and our semantic specification language is supported by a domain ontology about software components. Another related work was made by Lau

and Ukis [18]. They enrich component's interface with metadata which is used for preventing component's conflicts with the target execution environment. Their work is focused on .NET and J2EE frameworks. The ontologies based on software component and matching are mostly represented by work of Claus Pahl at Dublin City University [21] who made an ontology for software component matching. His ontology is based on DAML+OIL [4] logic languange. Our ontology has an advance in logic and it is based on OWL-DL logic model. We have found other works about software component ontologies [30][20], but they need extra information to be able to verify contracts among components.

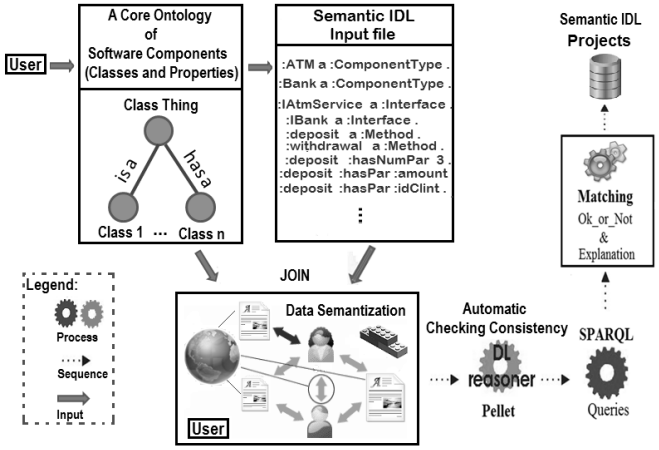

Figure 2. Semantic Verification Process

3. Semantic Web Techniques

3.1 Ontologies

An ontology [15][28] is a knowledge representation which defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relationships used to define extensions to the vocabulary. In our case, the domain area is software components. In particular, the ontology is able to manage all attributes for software components, establishing links between two components that can be connected by means of its interfaces.

3.2 SPARQL Query Language

SPARQL is a query language for the Resource Description Framework (RDF), this is a W3C Recommendation [34]. We use Web Ontology Language (OWL) [35] which extends RDF and RDFS. We use the *disjointWith* property to verify compatibility among the instances created by the user and its properties. We selected OWL DL language because we can assure that all conclusions given by the reasoner are computable and decidability. Example using RDF triples (*Parameter* class and *hasDataTypeParameter* object property) is showed below.

```
:Parameter a owl:Class .
:hasDataTypeParameter rdfs:domain
                      :Parameter .
:hasDataTypeParameter rdfs:range
                      :DataType .
```

4 Chichen Itza: verifying the matching of software contracts

Chichen Itza [1] is a Semantic Framework which it consists of a visual editor of software architectures. See Fig.1. The tool makes use of the library Flamingo and the Ribbon component [16] implemented in Java. The process to verify a matching among components is very easy for the user.

4.1 A Core Ontology for Software Components

A Software Component Ontology was created for capturing and verifying information about the input domain models during the Architectural Design [14]. It was written using **notation 3 or n3** [5] which is similar to RDF in its XML syntax, but more easy to understand. This ontology consisted of *20 classes*, *28 Object Properties*, *36 Data Properties*. The ontology was written using *n3 notation*, it is used by RDFS and OWL DL logic model. The main classes are: *ComponentType*, *Interface*, *Method*, *DataType*, *Parameter*, *ComponentModel*, *PreCondition* and *PostCondition*. The 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 specified by an *interface*, *type declarations*, a *pre-condition*, and *post-condition* [11]. In addition, there are two types of interfaces (provided and required). The interface of a method describes the syntactic specification of the method. Interfaces define the methods used in contracts and composition. 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). The Pre-condition describes the condition of the variables prior to the execution of the method whose behavior is described by the Post-condition.

[1] Chichen Itza is the name of a large pre-Columbian city built by the Maya civilization

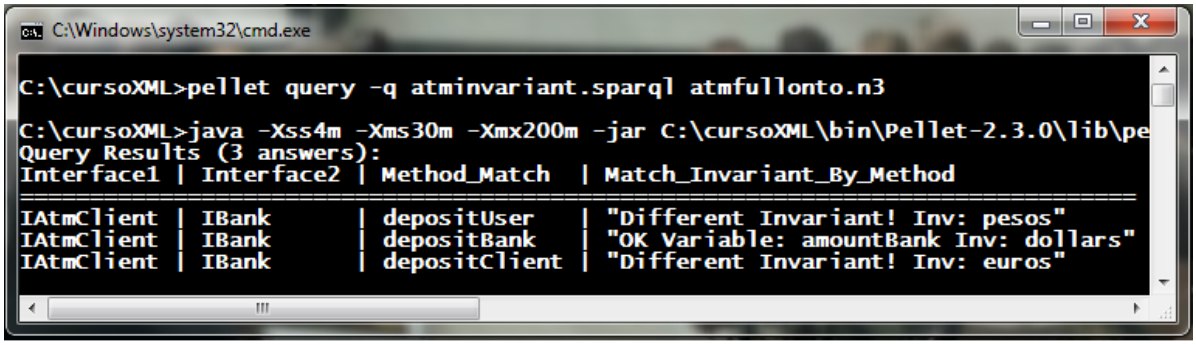

Figure 3. Currency invariant verification using a SPARQL query

4.1.1 Evaluating the core ontology created

The ontology developed has been evaluated in an informal and formal way. Regarding the former, the ontology was evaluated by the developers using the *Pellet* reasoner [23] to check the consistency of the ontology. The second evaluation applied to the ontology is based on the work of Gómez-Pérez [4] who establishes five criteria (*consistency*, *completeness*, *conciseness*, *expandability* and *sensitiveness*). The number of concepts and their relations among them, allow us to check the ontology consistency with less steps than other kind of ontologies.

4.2 Verification of Contract Matching

Our definition about matching is based on interfaces as contracts by Szyperski [29]. Interface specifications are contracts between a client of an interface and a provider of an implementation of the interface. A contract states what the client needs to do to use the interface. It also states what the provider requires to implement to meet the services promised by the interface. We define *Contract Matching* among components when we say there is a componet interface match when the provided interface of a component satisfies the requirements of the required interface of another component. Such a match is validated for sytactic and functional semantic aspects. In the first case, it is checked whether the provided interface includes at least the same list of methods defined in the required interface. We follow a structural approach whereby the names of the interface operations can be different but the types of the parameters and the order of the paramenters must be compliant. In the case of functional semantic it is validated using SPARQL queries about Invariants, Pre- and PostCondition of methods. See Figure 4. Conditions defined for each method has to be matched with the same variable, logic operator and value. We verify restrictions and assumptions at construction time, in a completely static manner, prior to the testing stages. Semantic verification is the process which uses Semantic Web Techniques (Ontologies and SPARQL queries) to guarantee compliance with contractual agreements. The semantics of an operation are described in an interface (contract). The only task for the user before to apply our model is to define the vocabulary of his domain and semantics. He introduces his model into the framework by means of a file or by the menus that allows us to do an automatic evaluation by using the Pellet reasoner [23] which checks inconsistencies. 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.

4.3 Using CORBAL-IDL vocabulary with Semantics

CORBA(Common Object Request Broker Architecture)[33] is a standard created by the Object Management Group (OMG)[10] that enables software components written in different computer languages for working among them by means of their interfaces. These interfaces are described using the *Interface Definition Language (IDL)*. In our semantic model, we need to receive the component interfaces written using the concepts and properties defined in the software component ontology and Bradas affirm that ”developing CORBA components is rather tedious by today’s standards due to its IDL-first approach” [8]. For the reasons above, we have decided to use the keywords of the CORBA-IDL with elements of the ontology and supported with Chichen Itza framework. For example, *ComponentType*, *Interface*, *Method*, *Parameter* and *hasNumParameters* are keywords. Part of the semantic ATM-IDL vocabulary. It is showed below.

```
:Atm        a :ComponentType .
:Bank       a :ComponentType .
:IAtmClient a :Interface .
:IAtmClient   :hasMethod :deposit .
:IBank      a :Interface .
```

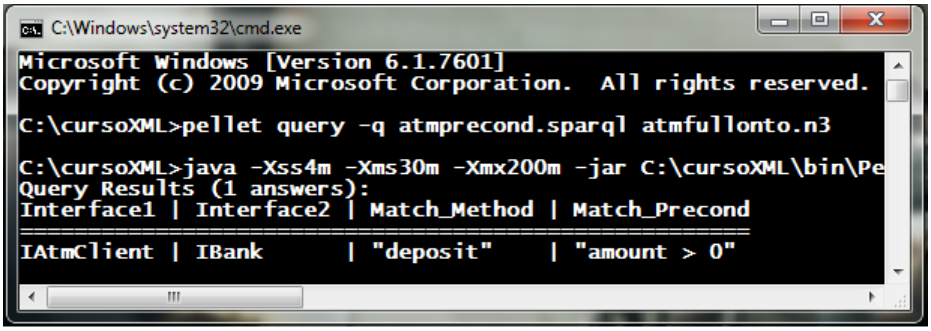

Figure 4. Matching the precondition about the amount

```
:IBank          :hasMethod :withdrawal .
:deposit     a :Method .
:withdrawal  a :Method .
:amout       a :Parameter .
:idClient    a :Parameter .
:deposit        :hasNumParameters  2 .
:withdrawal     :hasNumParameters  3 .
:deposit        :hasPrecond  :condition1 .
:deposit        :hasPostcond :condition3 .
                      :
```

In the code above we would like to emphazise that there are some instaces of clases (*Atm* and *Bank*), some classes (*ComponentType*, *Parameter*, *Interface* and *Method*), some object properties (*hasMethod*, *hasPrecond* and *hasPostcond*) and just one data type property (*hasNumParameter*). In particular, the notation *:deposit :hasNumParameters 2* means that the method deposit has exactly 2 parameters.

4.4 Using The Pellet Reasoner

Pellet [23] is an open-source Java based OWL DL reasoner. In our verification process we use Pellet for checking the consistency of the ontology. Pellet gives an explanation when an inconsistency is detected. Restrictions can be expressed into an ontology. For instance, the following code states that one component has at least 1 interface.

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

In contrast with the 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 allows for verifying restrictions in the input model (Semantic CORBA-IDL file). For example, we could establish that a component made in *.Net* can not run in the *Linux* operating system and the *EJB* component model is not compatible with the *MS COM* model. Defining *disjointWith* properties is also possible [1].

```
:Linux rdfs:subClassOf :OperatingSystem ;
     owl:disjointWith :Windows .
:EJB rdfs:subClassOf :ComponentModel ;
      owl:disjointWith :MS_COM .
```

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

4.5 Behavioral Contract verification using SPARQL

At this moment the complete verification is not possible using only the reasoner. For more complex checking we can apply anothers actions such as: production rules [13]. We decided to explore semantic queries in SPARQL [25] insted of production rules. The second step after the reasoner have checked the ontology consistency is to apply a SPARQL query. We defined specific queries that evaluate and verify the contract information of the components. Such queries are completely transparent to the user who only needs to provide the contracts of the components. We have used Jena API [17] and Java language [9] for programming and NetBeans IDE 7.0 [2]. SPARQL is similar to the database SQL but for ontologies. Besides, we can use variables in the queries, constraints, filtering information, logic operators, if statements and more. Lines are linked by variables which begin with a question mark. The same name of variable implies 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. Part of the query which verifies the *Precondition matching* is showed below.

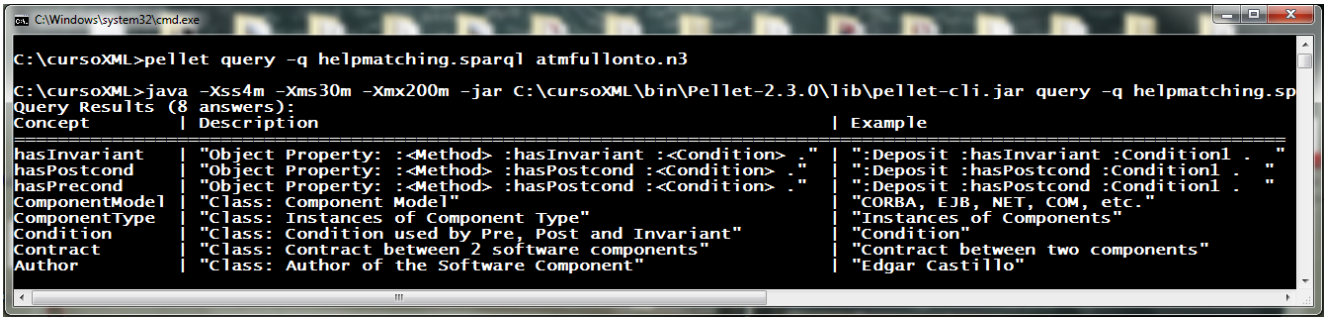

Figure 5. Semantic IDL help using the ontology

```
PREFIX     : <http://www.ejemplo.org/#>
PREFIX  owl: <http://www.w3.org/2002/07/owl#>
PREFIX  xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?Interface1 ?Interface2
                ?Match_Method ?Match_Precond
{
?Interface1  :typeInterface         :required ;
             :hasMethod             ?Method1 .
?Method1     :hasParameter          ?par1 ;
             :hasMethodName         ?name1 ;
             :hasNumParameters      ?numpar1 .
?par1        :hasIndexOrder         ?pos1 ;
             :hasDataTypeParameter  ?partype1 .
?Method1     :hasPrecond            ?precond1 .
?precond1    :hasVariable           ?var1 .
?precond1    :hasOperator           ?opr1 .
?opr1        :symbolOperator        ?oprname1 .
?precond1    :hasNumber             ?num1 .
                    :
} order by ?Match_Method
```

An additional benefit of using ontologies and SPARQL queries has been the extra information (metadata) to offer support for writing the IDL file. See Figure 5.

5 Automated Teller Machine: example

ATM is a machine at a bank branch or other location which enables customers to perform basic banking activities. The component model used for describing the ATM was written using UML 2 notation [7], and is shown in figure 6. The vocabulary of the input model is created by the user who selects classes and relation among concepts and he creates his instances. In this case the input model (semantic IDL file) only has the information of 5 software components and we can create its instances and relations among them using the Chichen Itza's menus.

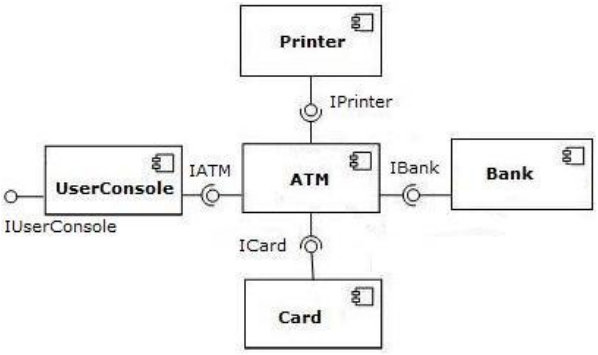

Figure 6. UML ATM Component-based system

6. Conclusions

In this paper we have presented and described a semantic technique for checking the Matching of Software Components. In comparision with other formal methods, this semantic technique, based on logic (ontology), a reasoner and a set of SPARQL queries offers an easy way to check matching among components and verifying the model proposed. This model can be extended and enriched with more concepts that rely on architectural design and non-functional requirements (QoS). The Ontology was expressed in a logic-based language (OWL DL), enabling detailed, sound, meaningful distinctions to be made among the contracts expressed as classes, properties and relations. The OWL DL ontology proposed is checked with the Pellet reasoner. Because it has a finite complexity. The use of a core domain ontology permits us to search for specific component information using intelligent techniques like SPARQL queries. Extending the ontology with no functional properties (Quality of Services attributes), Design Patterns and object properties (*hasInvoke*, *hasResponse*, etc.) for dynamic behaviour are key points for our future work.

7 Acknowledgments

This material is partly based upon work supported by the National Science Foundation under Grant No. OISE-0730065. I would like to express my deepest thanks to Comision Mexico-Estados Unidos para el Intercambio Educativo y Cultural (COMEXUS) through the Fulbright Programme, for gave me a grant to support my research in Miami, Florida without which this work would not have been possible, and Florida International University (FIU), School of Computing and Information Sciences. In particular, I would like to thank to Dr. S. Masoud Sadjadi and his team for their fruitful discussions and support for my research.

References

[1] F. E. Antoniou Grigoris and V. H. Frank. Introduction to semantic web ontology languages. 2005.

[2] E. Armstrong, J. Ball, S. Bodoff, D. B. Carson, I. Evans, K. Ganfield, D. Green, K. Haase, E. Jendrock, J. Jullion-ceccarelli, and G. Wielenga. The j2ee ^{TM}(tm) 1.4 tutorial for netbeans ^{TM}(tm) ide 4.1 for sun java system application server platform edition 8.1.

[3] M. Barnett and W. Schulte. Contracts, components, and their runtime verification on the .net platform. *J. Systems and Software, Special Issue on Component-Based Software Engineering*, 2002.

[4] S. Bechhofer, C. A. Goble, and I. Horrocks. Daml+oil is not enough. In *SWWS*, pages 151–159, 2001.

[5] T. Berners-Lee, D. Connolly, and S. Hawke. Semantic web tutorial using n3. In *Twelfth International World Wide Web Conference*, 2003.

[6] A. Beugnard, J. Jézéquel, N. Plouzeau, and D. Watkins. Making components contract aware. *Computer*, 32(7):38–45, 1999.

[7] M. Bjerkander and C. Kobryn. Architecting systems with uml 2.0. *Software, IEEE*, 20(4):57–61, 2003.

[8] P. Brada. The cosi component model: Reviving the black-box nature of components. *Component-Based Software Engineering*, pages 318–333, 2008.

[9] P. J. Clarke, D. Babich, T. M. King, and B. M. G. Kibria. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16:1512–1542, 1994.

[10] O. CORBA and I. Specification. Object management group, 1999.

[11] I. Crnkovic and M. Larsson. Building reliable component-based software systems. Artech House computing library, Norwood, MA, 2002.

[12] A. del Río, J. Gayo, and J. Lovelle. Verificación y validación mediante un modelo de componentes. In *Actas del Simposio Iberoamericano de Sistemas de Información e Ingeniería del Software en la Sociedad del Conocimiento (SISOFT-2001), Bogotá (Colombia)*, pages 29–31.

[13] A. C. del Río, J. E. L. Gayo, and J. M. C. Lovelle. A model for integrating knowledge into component-based software development. *KM - SOCO*, pages 26–29, 2001.

[14] A. Eden and R. Kazman. Architecture, design, implementation. In *proceedings of the 25th International Conference on Software Engineering*, pages 149–159. IEEE Computer Society, 2003.

[15] T. Gruber. Toward principles for the design of ontologies used for knowledge sharing. pages 907–928. 1995.

[16] Java.net. Flamingo. http://java.net/projects/flamingo/, 2010.

[17] Jena. Jena a semantic web framework for java. 2000.

[18] K. Lau and V. Ukis. Deployment contracts for software components. *Preprint*, 36, 2006.

[19] L. Mariani and M. Pezze. A technique for verifying component-based software3. *Electronic Notes in Theoretical Computer Science*, 116:17–30, 2005.

[20] X. Nianfang, Y. Xiaohui, and L. Xinke. Software components description based on ontology. In *Proceedings of the 2010 Second International Conference on Computer Modeling and Simulation - Volume 04*, ICCMS '10, pages 423–426, Washington, DC, USA, 2010. IEEE Computer Society.

[21] C. Pahl. An ontology for software component matching. volume 9, pages 169–178. Springer-Verlag, Berlin, Heidelberg, 2007.

[22] D. Parnas. Really rethinking 'formal methods'. *Computer*, 43(1):28 –34, Jan. 2010.

[23] B. Parsia and E. Sirin. Pellet: An owl dl reasoner. In *In Proceedings of the International Workshop on Description Logics*, 2004.

[24] J. P. Paul, P. and J. I. Siddiqui. *Formal Methods State of the Art and New Directions*. Springer, Springer London Dordrecht Heidelberg New York, 2009.

[25] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *The Semantic Web-ISWC 2006*, pages 30–43, 2006.

[26] G. Reed. Exploiting formal methods in the real world: a case study of an academic spin-off company. In *Proceedings of Modelling Software System Structures in a fastly moving scenario*, 2000.

[27] J. Spivey. *Understanding Z: a specification language and its formal semantics*, volume 3. Cambridge Univ Pr, 1988.

[28] S. H. Staab S., Studer R. and Y. Sure. Knowledge processes and ontologies. volume 16, pages 26–34. Jan-Feb 2001.

[29] C. Szyperski, D. Gruntz, and S. Murer. *Component software: beyond object-oriented programming*. Addison-Wesley Professional, 2002.

[30] A. Talevski, P. Wongthongtham, and S. Komchaliaw. Towards a software component ontology. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '08, pages 503–507, New York, NY, USA, 2008. ACM.

[31] J. Tsai and E. Juan. Compositional approach for modeling and verification of component-based software systems. In *Proceedings of the 2000 Monterey Workshop on Modeling Software System Structures in a Fast Moving Scenario*, pages 13–16. Citeseer.

[32] J. J. P. Tsai and E. Y. T. Juan. Compositional approach for modeling and verification of component-based software systems. In *Proceedings of Modelling Software System Structures in a fastly moving scenario*, 2000.

[33] S. Vinoski. Distributed object computing with corba. *C++ Report*, 5(6):32–38, 1993.

[34] W3C. http://www.w3.org/consortium/. 1994.
[35] W3C. Owl web ontology language, 1994.