Towards a Software Domain Metric based on Semantic Web Techniques

F. Edgar Castillo-Barrera[1], **Héctor G. Pérez-González**[1], **and S. Masoud Sadjadi**[2]
[1]School of Engineering, Universidad Autónoma de San Luis Potosí, San Luis Potosí, México
[2]School of Computing and Information Sciences, Florida International University (FIU), Miami, USA

Abstract

The reuse of software remains a major objective in the software industry. An important task in order to accomplish this goal is to classify the software based on the application domain for which it was done. This action facilitates their possible assembly with other programs based on the same vocabulary and domain. In this paper we describe a software domain metric which is measured based on semantic web techniques. This metric is independent of lines of code, binary and executable code of the software, and the programming language. Our approach is based on a lightweight ontology of CORBAL-IDL language and SPARQL queries. The ontology captures the vocabulary and its relation. This is encoded using OWL DL, supported by the Pellet reasoner to check the ontology component consistency. The populated ontology is queried using SPARQL. These queries look for matching words based on a vocabulary which describes a domain. We use an example and a prototype (a semantic framework called Chichen-Itza) to show the feasibility of our approach.

1. Introduction

The IEEE Standard 610.12-1990, Standard Glossary of Software Engineering, defines a Metric as: "A quantitative measure of the degree to which a system, component, or process possesses a given attribute". This quantitative measure is not always possible to apply based on lines of code. For example, Software Components are sold without source code. Another example is the concept of abstract attributes, for which there are not direct ways of measuring them or to quantify them. In this work we focus on attributes based on "the domain or context" which allows us to determine if a software component or application was done for a specific domain. Information about the domain can be used to determine if it is possible to assemble two software componentes, for example. Crnkovic and Larsson [8] 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 based on its domain are necessary.

In this work, we propose a software metric based on semantic web techniques (Ontologies, Reasoners and Semantic queries) in conjunction with the *Chichen-Itza* framework to mitigate this problem. We propose an approach for measuring such indicators. This approach looks for matching words in *a CORBA-IDL++ file* using and Ontology populated with *words based on a vocabulary for a specific domain*. For each application, artifact or software component it is neccesary to make a file in CORBA-IDL++ and a file with the vocabulary of the domain. CORBA-IDL++ is an extension of CORBA-IDL language which we made it for this purpose. Our method for measuring is able to check matching words in differente languages and it can recognize a word within another.

Our method for measuring, can only be applied if the application, artifact or software components can be described as methods and parameters. Binary, Executable, and Source Code are not required. In this work, we consider the following definition: *"A component is a reusable unit of deployment and composition that is accessed through an interface"*[8]. In practice, we have noted that problems related to interface incompatibility are frequent. In particular, incompatibility with the semantics of operation parameters and interface operations (behavioral contracts [4]). We consider that the use of a semantic matching approach (a software component ontology) could help to detect domain based on the vocabulary of the domain before the component-based system is deployed. The rest of the paper is structured as follows. In Section 2 we present our proposal to measure the vocabulary in a specific domain. In Section 3 we explain the Semantic Web Framework called Chichen Itza and semantic web techniques (Ontologies and SPARQL queries). Section 4 shows an example about our semantic approach in the ATM domain. In Section 5 we draw some concluding remarks. Finally, acknowledgments are given in Section 6.

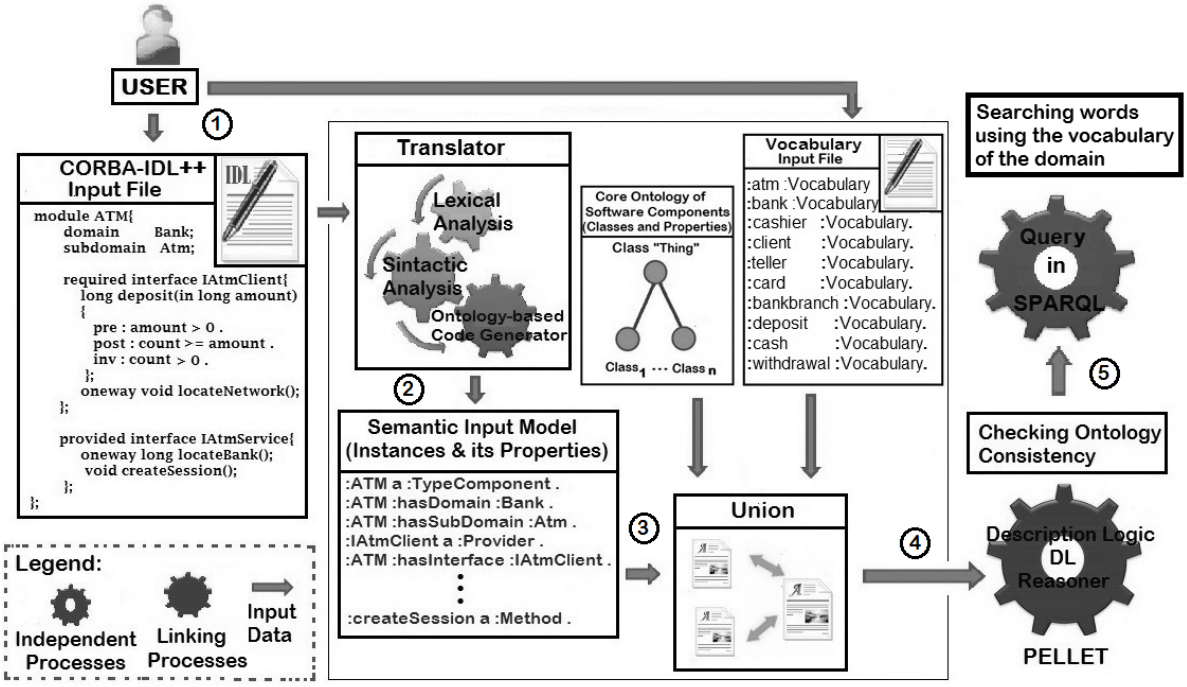

Figure 1. Process to measure the Vocabulary Domain.

2. A Metric based on Semantic Web Techniques

2.1 How to measure?

The measurement process takes place in five steps as shown in Figure 1. In the first step, the user must create an input file written using the language of CORBA-IDL++ (which is an extension of the language of CORBAL-IDL) where it must specify the methods or instructions with its parameters. It is also necessary to have a file with the vocabulary of the domain to check, which must be as complete as possible.

In the second step it is necessary to translate the input file written in CORBA-IDL++ into the language RDF. On having applied the translator, a file with extension n3 is generated.

Phase 3 will need to join the generator file by the translator in n3 with the file the vocabulary of the domain and the ontology of CORBA-IDL developed to the prototype of Chichen-Itza in just one project (which will be called le ontological project).

In phase 4, we have to apply the Pellet Reasoner to the ontological project created in the previous phase.

It is at this stage where the reasoner verifies the consistency of the ontology. If the consistency is right we can ensure that the ontology does not have problems of inconsistencies, so we can apply Semantic queries without problems of computability and decidability [13].

Finally, in phase 5 the user has to apply a query made in SPARQL to search the vocabulary in the ontological project and thus with this to have how many words matched with the language used in the application. With this information, we can apply the semantic metric.

2.2 What is the Metric?

The most common definition of Metric is: "quantitative measure of degree to which a system, component or process possesses a given attribute". The metric proposed tries to give a quantitative measure of degree to which a system possesses an attribute based on a specific domain or context. The formula that appears below calculates the percentage to which a program belongs in a specific domain:

$$MDom = \frac{\#matching\ words\ input\ vocabulary}{\#identifiers\ (\#methods+\#parameters)} * 100$$

In order to calculate this, we have to know the number of matching words, the number of methods and parameters. We want to point out that the words defined in the input vocabulary file have to be as complete as possible. In table 1 we have calculated the metric for 6 applications, 3 of

Table 1. The results obtained after we have applied the semantic metric

Application	In ATM Domain	Number of Methods	Number of Parameters	Number of Words Vocabulary domain	Matching Words	Semantic Metric	Approved
A	**Yes**	7	5	**15**	8	66	Yes
B	**Yes**	25	46	**15**	60	84	Yes
C	**Yes**	50	92	**15**	121	85	Yes
D	**No**	60	127	**15**	4	2	No
E	**No**	70	140	**15**	1	0.5	No
F	**No**	80	204	**15**	6	2	No

them are in the ATM domain and the others are not. For example in application A:

$MDom = \frac{8}{(7+5)} * 100 = 66$

2.3 Ontologies

An ontology [10] 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 CORBA-IDL language.

2.4 SPARQL Query Language

SPARQL is a query language for the Resource Description Framework (RDF). We have selected it because this is a W3C Recommendation [18]. We use Web Ontology Language (OWL-DL) [19] which extends RDF and RDFS. We selected OWL DL language because we can assure that all conclusions given by the Reasoner are computable and decidable.

3 Chichen Itza: a Semantic Web Framework

Chichen Itza [1] is a Semantic Web Framework which allows the management of semantic models (Ontologies) in memory, verify its consistency (Reasoners) and execute semantic queries in SPARQL language. Chichen Itza consists of a friendly visual editor where the users can edit, save and load their ontologies and queries. This framework was programmed in Java language [11] and is portable to other plataforms. The Chichen Itza framework is shown in 2

3.1 A CORBA-IDL Ontology

A CORBA-IDL Ontology was created for verifying information about the input domain models. This ontology consisted of *20 classes*, *28 Object Properties*, *36 Data Properties* and it was written using *n3 notation* [3] because it is easier to understand than RDF in its XML syntax. 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* [8]. The interface of a method describes the syntactic specification of the method. Interfaces define the methods used in contracts. 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.

3.1.1 Evaluating the 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 [14] to check the consistency of the ontology. The second evaluation applied to the ontology is based on the work of Gómez-Pérez [2] 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.

3.2 Domain Verification based on Vocabulary

Our approach about matching words is based on interfaces as contracts by Szyperski [16]. Interface specifications are contracts between a client of an interface and a

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

provider of an implementation of the interface. A contract states what the client needs to use the interface. It also states what the provider requires to implement to meet the services promised by the interface. 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. Conditions defined for each method have 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 applying 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 to do an automatic evaluation by using the Pellet reasoner [14] 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.

3.3 Extending CORBAL-IDL vocabulary with Semantics

CORBA(Common Object Request Broker Architecture)[17] is a standard created by the Object Management Group (OMG)[7] that enables software components written in different computer languages to work 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 interface written using the concepts and properties defined in the CORBA-IDL ontology. 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 .
:IBank         :hasMethod :withdrawal .
:deposit     a :Method .
:withdrawal  a :Method .
:amout       a :Parameter .
:idClient    a :Parameter .
:deposit       :hasNumParameters  2 .
:withdrawal    :hasNumParameters  3 .
```

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*), object property *hasMethod* and just one data type property (*hasNumParameter*). In particular, the notation *:deposit :hasNumParameters 2* means that the method deposit has exactly 2 parameters.

3.4 The Pellet Reasoner

Pellet [14] is an open-source Java based OWL DL reasoner. In our verification process we use Pellet for checking the consistency of the ontology. We have selected the Pellet reasoner because it gives an explanation when an inconsistency is found. It is also possible to check for restrictions.

3.5 Domain verification using SPARQL queries

For more complex checking we can apply other actions such as: production rules [9]. We decided to explore semantic queries in SPARQL [15] instead of production rules. The second step after the reasoner has checked the ontology consistency is to apply a SPARQL query. We defined specific queries that evaluate matching words in methods and parameters identifiers. Such queries are completely transparent to the user who only needs to provide the file produced in n3 by the translator. We have used Jena API [12] and Java language [6] for programming and NetBeans IDE 7.0 [1]. SPARQL is similar to the database SQL but for ontologies. Besides, we can use variables in the queries, filtering information, and if statements. 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. The query which verifies the *matching words with the name of the methods* is showed below.

```
PREFIX       : <http://www.ejemplo.org/#>
PREFIX  rdf: <http://www.w3.org/1999/02/
                        22-rdf-syntax-ns#>
PREFIX  owl: <http://www.w3.org/2002/07/owl#>
SELECT ?Vocabulary  ?MatchMethod WHERE
 {
   ?Vocabulary  rdf:type  :Vocabulary  .
   ?Interface :hasMethod ?Method .
   BIND( if(regex(str(?Method),
                  str(?Vocabulary), "i"),
```

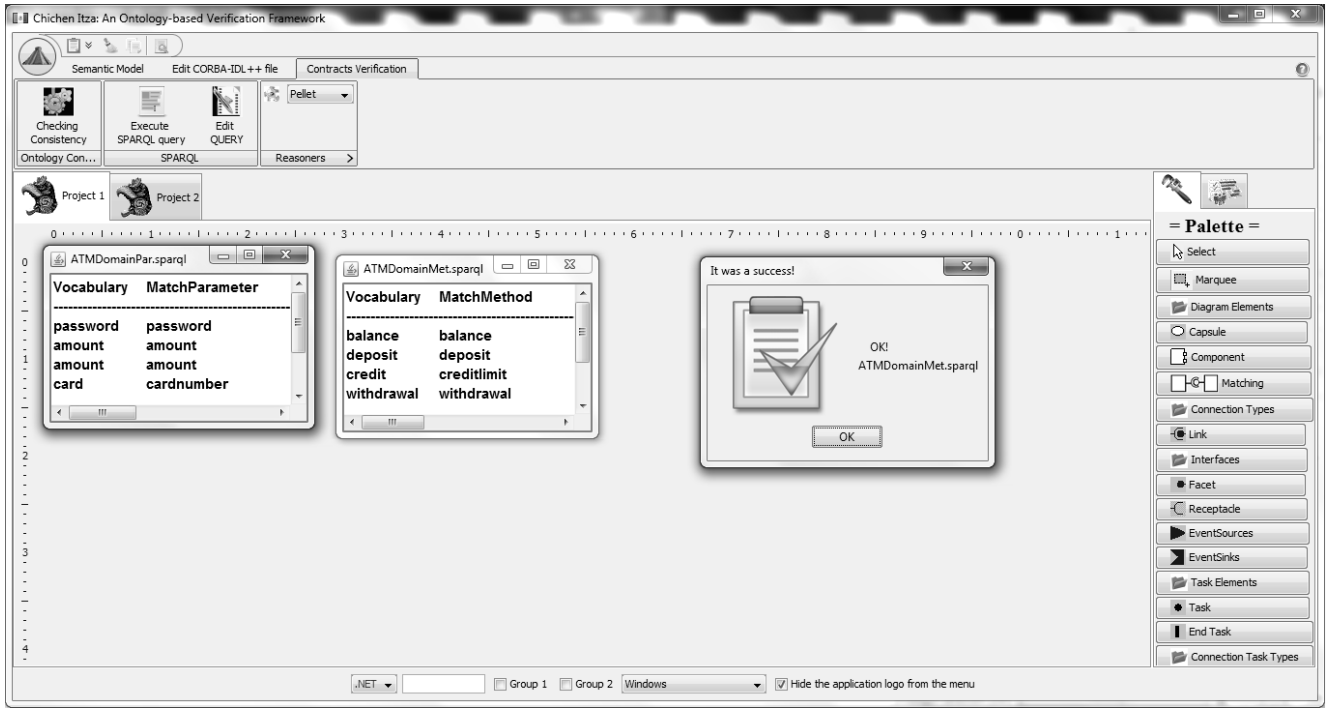

Figure 2. Chichen Itza Framework: two queries in SPARQL were used by looking for matching words

```
            ?Method, " ")  AS ?MatchMethod)
   FILTER (  ?MatchMethod != " "  )
}
```

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

4 Example: Automated Teller Machine

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 [5], and is shown in figure 3. The vocabulary of the input file is created by the user or expert in the domain of ATM. He selects which words are used in that domain. In our example, for each component is neccesary to create an input file written in CORBA-IDL++.

```
:bank                   a :Vocabulary .
:Atm                    a :Vocabulary .
:cashier                a :Vocabulary .
:client                 a :Vocabulary .
:bankbranch             a :Vocabulary .
:teller                 a :Vocabulary .
:card                   a :Vocabulary .
:money                  a :Vocabulary .
```

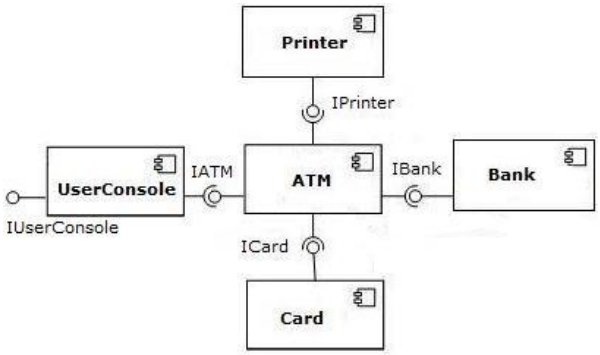

Figure 3. UML ATM Component-based system

```
:amount                 a :Vocabulary .
:password               a :Vocabulary .
:balance                a :Vocabulary .
:deposit                a :Vocabulary .
:withdrawal             a :Vocabulary .
:credit                 a :Vocabulary .

 module ATM{
        domain     Bank;
        subdomain  Atm;
```

```
        provided interface IAtmService{
            oneway void locateBank();
                   long createSession();
                   long balance();
                   long creditLimit();
        };
        required interface IAtmClient{
          long deposit(in short amount,
                       in short numclient);
          void withdrawal(in short cardnumber,
                          in char password,
                          in short amount);
          void locateNetwork();
        };
};
```

5. Conclusions

In this paper we have presented and described a software metric for measuring the percentage that belongs to an application of a specific domain based on a vocabulary used in that domain. In comparision with other metrics, this metric tries to measure an abstract attribute based on the vocabulary of a specific domain. This measure is based on an Ontology, a Reasoner, and a set of SPARQL queries which allow us an easy way to check matching words. This model can be extended and enriched with more attributes that rely on semantics. The Ontology was expressed in a logic-based language (OWL DL). Using this language we can assure the query will not have problems of computability and decidability. The OWL DL ontology proposed is checked with the Pellet reasoner. The use of a domain ontology allows us to search for specific words using intelligent techniques such as SPARQL queries. Extending the ontology with no functional properties (Quality of Services attributes), Design Patterns and Object properties (*hasInvoke*, *hasResponse*, etc.) for measuring the behaviour are key points for our future work.

6 Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. OISE-0730065.

References

[1] E. Armstrong, J. Ball, S. Bodoff, D. B. Carson, I. Evans, K. Ganfield, D. Green, K. Haase, E. Jendrock, J. Jullionceccarelli, 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.

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

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

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

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

[6] 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.

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

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

[9] 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.

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

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

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

[13] J. Z. Pan and E. Thomas. Approximating owl-dl ontologies. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1434. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

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

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

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

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

[18] W3C. http://www.w3.org/consortium/. 1994.

[19] W3C. Owl web ontology language, 1994.