Ontology-driven Translators: The new generation

Francisco-Edgar Castillo-Barrera

Engineering Faculty, Universidad Autónoma de San Luis Potosí, México
ecastillo@uaslp.mx

Abstract. In this paper we describe a proposal for a new generation of translators. This approach is based on a domain ontology of software components for driving the translation process. We use an example and a prototype to show the feasibility of our approach.

Keywords: Domain ontology, Translators, CORBA-IDL, SPARQL, Pellet, Reasoner, Description logic.

1 Introduction

In software ingeniering the research about how the component context can improve their assembling has been studied so long. We dispose of standard like CORBA-IDL. We are interenting in explore the use of a domain ontology not only for guiding the assembling of components but also for enriching the component descriptions.

These enriching descriptions needs a different kind of translators. These ones could be based on a domain ontology of software components for driving the translation. This approach has some advantages like:

- **Semantic associated to the code**, ie in software components the entry file which contains information about declaration of their interfaces and their methods can be enriched with an ontology.
- **New knowledge.** By transforming the translations in an ontology (description logic) reasoners can be applied automatically and possibly generate new knowledge (components where possible may be used).

We consider three kind of translators based on ontologies:

1. Translators that incorporates the source code of the ontology in the translation process.
2. Translators which maintain a direct communication with the domain ontology during the translation proces.
3. Translators which before doing the translation using an ontology based on a domain to verify that the vocabulary used in the code is correct or not.

For example, an English to Spanish translator. This translator receives text in English, uses an ontology to verify English language (eg WordNet).

In this paper we describe the second kind of translators. In our approach, a populated ontology (properties,instances) is generated from a CORBA-IDL+C file by using an ontology-based translator.

The rest of the paper is structured as follows. In Section 2 we present some related work. In Section 3 we describe our approach an Ontology-based Translator for CORBA-IDL+C language. Section 4 describes an example and a prototype to show the feasibility of our approach. Finally, in Section 5 we draw some concluding remarks.

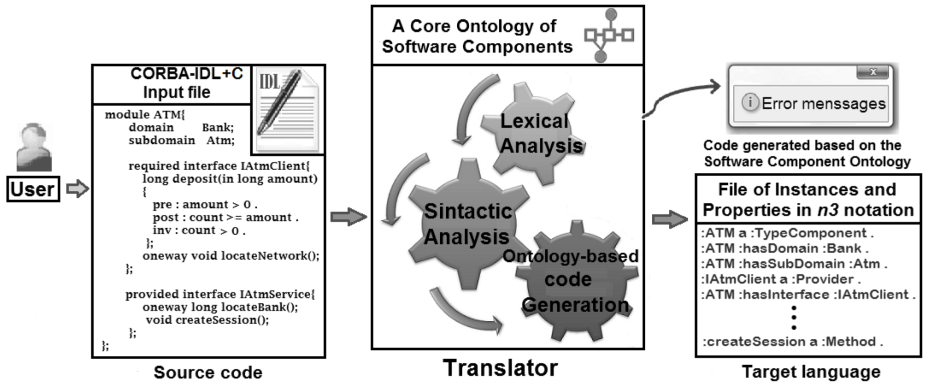
2 Related Work

An ontology for Syntactic and Semantic English-Korean machine was made by Il-Sun Song, Su-Kyung Kim and Ho-Jin Choi [9]. The authors apply two translation modules to achieve their goal: Syntactic and Semantic. The first module transforms the English structure into Korean structure and the second module extracts an exact meaning of a word using ontologies. In contrast, our proposal is based on Programming Languages and we use a domain ontology about Software Components. A closely related work is a Test Specification Code Translator using ontology and XML/XSLT Technologies presented by Lim Lian Tze, Tang Enya Kong and Zaharin Yusoff [11]. They developed and implemented a framework for translating test specification code between platform-specific languages and they use a test domain ontology to translate correctly test keywords. We do not use an ontology to correct word, by contrast, we use a domain ontology for translating CORBA code in an ontology source code in OWL-DL sintax. The most closely related work was made by Alessio Lomuscio, Hongyang Qu and Monika Solanki [7]. This approach consists(semi-)automatically compile and verify contract-regulated service compositions.

3 An Ontology-based Translator for CORBA-IDL+C language

Six steps were required to build the Ontology-based Translator which generates a populated Ontology of Software Componentes in *OWL-DL* language and using *n3* notation. They consist on:

1. Building an Ontology in the Domain of Software Components
2. Defining the structure of the file in CORBA-IDL++
3. Extending the keywords of CORBA-IDL language
4. Implementing the Lexical Analyzer for a CORBA-IDL++ file
5. Implementing the Syntactic Analyzer
6. Generating the Target Code by Syntax-directed Translator


CORBA-IDL+C Input file
User
Source code
A Core Ontology of Software Components
Lexical Analysis
Sintactic Analysis
Ontology-based code Generation
Translator
Error menssages
Code generated based on the Software Component Ontology
File of Instances and Properties in *n3* notation
Target language

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

    required interface IAtmClient{
       long deposit(in long amount)
       {
         pre : amount > 0 .
         post : count >= amount .
         inv : count > 0 .
        };
       oneway void locateNetwork();
    };

    provided interface IAtmService{
       oneway long locateBank();
        void createSession();
     };
};
```

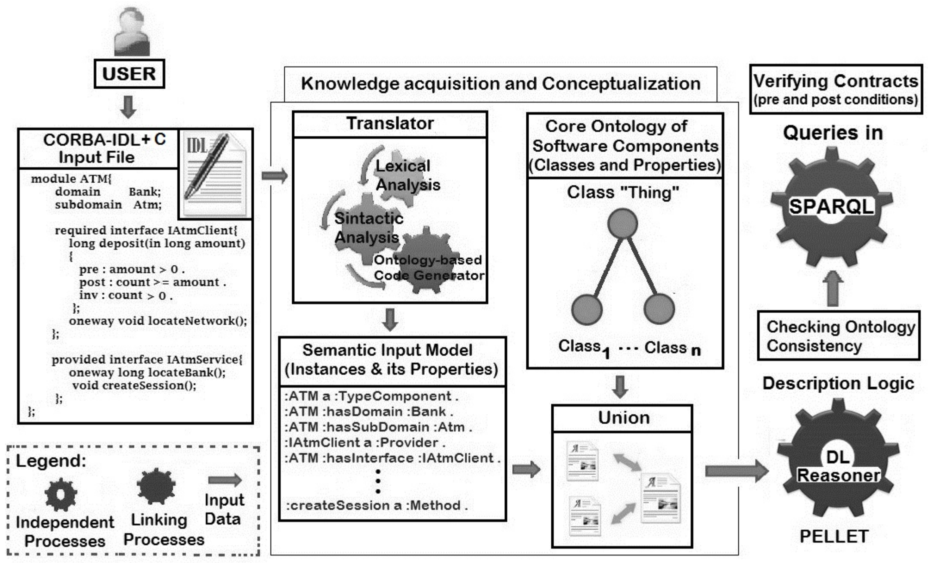
```
:ATM a :TypeComponent .
:ATM :hasDomain :Bank .
:ATM :hasSubDomain :Atm .
:IAtmClient a :Provider .
:ATM :hasInterface :IAtmClient .
        ⋮
:createSession a :Method .
```

Fig. 1. The Stages of the Ontology-based Translator

In this paper, we will describe only the step one. An ontology [4][10] 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. The Ontology built in this work was in the domain of software components. Ontology classes and subclasses definition were written using **notation 3 or n3** [2] which is similar to RDF in its XML syntax, but more easy to understand. When we define a new vocabulary we have to define new classes, it means what type of thing something is, we write *a owl:Class*. The statements each consist of a subject, verb and object ending with a period. In **n3** we can write RDF triples in that way. The Ontology built is showed below.

```
:ComponentType   a owl:Class .
:Interface       a owl:Class .
:Provider  rdfs:subClassOf :Interface .
:Required  rdfs:subClassOf :Interface .
:Provider   owl:disjointWith :Required .
:Required   owl:disjointWith :Provider .
:Method          a owl:Class .
:DataType        a owl:Class .
:Parameter       a owl:Class .
:OperatingSystem a owl:Class .
:ComponentModel  a owl:Class .
:Requirements    a owl:Class .
       :FunctionalRequirements
             rdfs:subClassOf :Requirements .
       :NonFunctionalRequirements_QoS
             rdfs:subClassOf :Requirements .
:PreCondition  rdfs:subClassOf :Condition .
:PostCondition rdfs:subClassOf :Condition .
```

List 1. The main classes of the software component ontology


USER
CORBA-IDL+C Input File
IDL
Knowledge acquisition and Conceptualization
Translator
Lexical Analysis
Sintactic Analysis
Ontology-based Code Generator
Semantic Input Model (Instances & its Properties)
Core Ontology of Software Components (Classes and Properties)
Class "Thing"
Class1 ··· Class n
Union
Verifying Contracts (pre and post conditions)
Queries in
SPARQL
Checking Ontology Consistency
Description Logic
DL Reasoner
PELLET
Legend:
Independent Processes
Linking Processes
Input Data

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

    required interface IAtmClient{
      long deposit(in long amount)
      {
        pre : amount > 0 .
        post : count >= amount .
        inv : count > 0 .
       };
      oneway void locateNetwork();
    };

    provided interface IAtmService{
      oneway long locateBank();
       void createSession();
    };
};
```

```
:ATM a :TypeComponent .
:ATM :hasDomain :Bank .
:ATM :hasSubDomain :Atm .
:IAtmClient a :Provider .
:ATM :hasInterface :IAtmClient .
         ⋮
:createSession a :Method .
```

Fig. 2. Ontology Verification Process based on a CORBA-IDL+C Translator

The mainly concepts used in our logic model are methods, contracts, and interfaces. Interfaces define the methods used in contracts and composition. 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. Some classes are showed in List 1. The Ontology is built by means of classes and relations among concepts. Each method is specified by an *interface*, *type declarations*, a *pre-condition*, and *post-condition* [3]. In addition, there are 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, Post and Inv). The Pre-condition describes the condition of the variables prior to the execution of the method whose behavior is described by the Post-condition. Invariant are values which has to be hold during the hole process.

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 during the whole ontology life cycle and they used the **Pellet** reasoner [3] to check the consistency of the ontology. This group supervised the releases mainly by asking the defined competency questions and checking whether the ontology could answer them. The second evaluation applied to the ontology is based on the work of Gómez-Pérez [1] who establishes five criteria (*consistency*, *completeness*, *conciseness*, *expandability* and *sensitiveness*).

4 Verifying contracts between ATM and Bank components

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. The component model used for describing the ATM system was made in Chichen Itza Framework using its graphical interface of software components, and is shown in figure 3: One example in the design phase using the ATM example [6]. In this case the input model (semantic IDL file) only has the information of 2 software components and we can create its instances and relations among them using the Chichen Itza's menus. We complement the verification process performing queries in SPARQL.

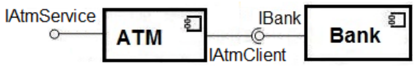
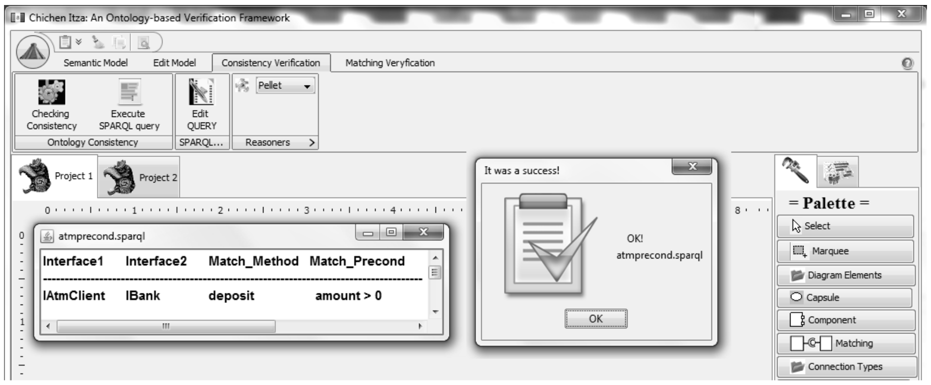


Fig. 3. ATM and BANK Components in UML

A prototype of the framework involves a visual editor. See Fig.4. The tool makes use of the library Flamingo and the Ribbon component [5] implemented in Java. The process to verify a matching among components is very easy for the user.



atmprecond.sparql

```
Interface1   Interface2   Match_Method   Match_Precond
------------------------------------------------------
IAtmClient   IBank        deposit        amount > 0
```

It was a success!

OK!
atmprecond.sparql

OK

Fig. 4. Matching the Precondition about the amount variable using a SPARQL query

Component	Interfaces	Methods
ATM	IAtmService	createSession(), locateBank()
	IAtmClient	deposit(in long Amount, in long NumClient)
Bank	IBank	void Withdrawal(in long CardNo, in string Password, in long Amount) void Deposit(in long CardNo, in float Amount)

Table 1. Interfaces and Methods of ATM and Bank components

4.1 Using The Pellet Reasoner

Pellet [8] is an open-source Java based OWL DL reasoner. In our verification process we use Pellet for checking the consistency of the ontology. Ontology consistency is defined as a set of conditions that must hold for every ontology. Pellet gives an explanation when an inconsistency is detected.

4.2 Code Generated by the Translator

Part of the code generated by the Translator using the ATM and BANK IDL files 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 .
:amount     a :Parameter .
:numclient  a :Parameter .
:deposit      :hasNumParameters  2 .
:deposit      :hasParameter :amount .
:amount       :hasIndexOrder 1 .
:deposit      :hasPrecond   :condition1 .
        :
```

5 Conclusions

In this paper we have presented and described an Ontology-based Translator-Compiler for generating a populated ontology source code (instances and properties) based on an Domain Ontology of Software Components.

A formal verification based on a Reasoner (Pellet) can be applied at the target code generated in an automatic way, without expertise. In addition we can extract information and knowledge using SPARQL queries. This code can be classified in a certain domain increasing the reuse and compatibility of the component with others. Besides, the verification of the contracts can be done.

The Ontology used was represented in a logic-based language (OWL DL). The OWL DL ontology proposed is checked with the Pellet reasoner and it has a finite complexity (it has not problems of decidibility).

The main contribution of this work is to generate a populated Ontology of properties and instances from a CORBA-IDL+C file by using an Ontology-based Translator which can be used for verifying contracts among components in a formal way. In our example, we have verified the matching among software components using reasoners (formal method based on a Description Logic Reasoner) for verifying the matching of the software components based on Contracts, a software components ontology, interfaces (pre-conditions, post-conditions and invariants), SPARQL queries.

References

1. Bechhofer, S., Goble, C.A., Horrocks, I.: Daml+oil is not enough. In: SWWS. pp. 151–159 (2001), http://www.informatik.uni-trier.de/ ley/db/conf/semweb/swws2001.htmlBechhoferGH01
2. Berners-Lee, T., Connolly, D., Hawke, S.: Semantic web tutorial using n3. In: Twelfth International World Wide Web Conference (2003)
3. Crnkovic, I., Larsson, M.: Building reliable component-based software systems. Artech House computing library, Norwood, MA (2002)
4. Gruber, T.: Toward principles for the design of ontologies used for knowledge sharing. pp. 907–928 (1995)
5. Java.net: Flamingo. http://java.net/projects/flamingo/ (2010)
6. kiu Lau, K., Wang, Z.: A survey of software component models. Tech. rep., in Software Engineering and Advanced Applications. 2005. 31 st EUROMICRO Conference: IEEE Computer Socity (2005)
7. Lomuscio, A., Qu, H., Solanki, M.: Towards verifying contract regulated service composition. In: Web Services, 2008. ICWS'08. IEEE International Conference on. pp. 254–261. IEEE (2008)
8. Parsia, B., Sirin, E.: Pellet: An owl dl reasoner. In: In Proceedings of the International Workshop on Description Logics (2004)
9. Seo, E., Song, I.S., Kim, S.K., Choi, H.J.: Syntactic and semantic english-korean machine translation using ontology. In: Proceedings of the 11th international conference on Advanced Communication Technology - Volume 3. pp. 2129–2132. ICACT'09, IEEE Press, Piscataway, NJ, USA (2009), http://dl.acm.org/citation.cfm?id=1701655.1701781
10. Staab S., Studer R., S.H., Sure, Y.: Knowledge processes and ontologies. vol. 16, pp. 26–34 (Jan-Feb 2001)
11. Tze, L., Kong, T., Yusoff, Z.: A test specification code translator using ontology and xml/xslt technologies