Chichen-Itza II: a Semantic Framework for Verification and Learning of Pipes and Filters architectural pattern

Francisco-Edgar Castillo-Barrera[1] and Carolina Medina-Ramirez[2]

[1] Autonomous University of San Luis Potosi, San Luis Potosi SLP 78290, Mexico,
ecastillo@uaslp.mx
[2] Metropolitan Autonomous University, Department of Electrical Engineering,
Mexico City, Mexico
cmed@xanum.uam.mx

Abstract. In software engineering, verification of architectural designs is a task that requires a very high level of abstraction. In companies, verification techniques for architectural designs are difficult to integrate into the standard software development process because it requires specialized expertise in an application domain. This paper describes a semantic framework for verifying models and how it is used for learning of Pipes and Filters Architectural Patterns (PFAP). Elements of this architecture are seen as software components which are assembled among them and each assembling implies a contract. Each contract is verified using semantic web technologies based on the rules of PFAP. We show an example of the verification of Voice over an IP model using an extended and new version of the prototype called Chichen-Itza II to show the feasibility of our approach.

Keywords: Ontology, Software Components, Contracts, Pipes and Filters Architectural pattern, VoIP, Verification, Assembling, SPARQL, Pellet OWL-DL Reasoner

1 Introduction

Len Bass, Paul Clements and Rick Kazman define Software Architecture [4] as: "The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them". Software architecture is important in companies because it helps stakeholders to understand different system characteristics that are affected by the architecture [10]. As a result, the architect and stakeholders can make early design decisions about the system development, deployment and maintenance life. Software architectures are classified in families called architectural patterns, also called design patterns. An architectural pattern defines a lexicon of components and connectors with rules on how they can be assembled. Patterns allow the reuse of designs by giving solutions to frequently occurring problems. System

architectures are described by architecture description languages (ADLs) [18]. The first ADL was Acme, developed by the ABLE group at Carnegie Mellon University, and Dave Wile at USC's Information Sciences Institute [11]. After that a visual tool called AcmeStudio (graphical editor for architectural designs) was developed including Acme ADL. Other frameworks were developed such as Aesop [8], C2 [16], Darwin [12], Jacal [14], and MetaH [19], for improving a better understanding and communication among the stakeholders in the project. It is important to note that they have to understand the architectural pattern used in the system design. Moreover, the *Semantic Web* is an extension of the World Wide which is a set of standards, a set of tools, and people that shares data [2]. *Semantic Technology* is another important concept in Computer Science which its goal is to give semantics to data and it is supported by semantic tools [7] such as (RDF, SPARQL, OWL, and others) that provide semantic information about the meaning of words. Semantic Web Techniques are methods and techniques based on semantic tools which allow us to also manipulate in which allow the verification and learning of PFAP designs by applying semantic techniques (Ontologies and SPARQL queries)formation. In this work, we propose a new extended and modified version of the semantic framework called Chichen-Itza [3]. With this technology, we want to mitigate the problem of verifying architectural designs and improve the learning of this kind of projects based on Pipes and Filters architectural pattern. We propose an approach for verifying the assembling among elements of this architectural pattern using contracts of a required interface that matches with the contract of a provided interface. We consider that the use of a semantic matching approach (a pipe and filter ontology) could help to detect interface incompatibility before the system is deployed. We created a Voice over IP system architectural design based on Pipes and Filters pattern for exemplifying our approach. The rest of the paper is structured as follows. In Section 2 we present some related work. In Section 3 we presents the Chichen-Itza Framework and its features. Section 4 describes the process of verification used by Chichen-Itza and our semantic approach for verifying the matching of Architectural Components. In Section 5 we show the feasibility of our technique by describing an example building a Voice over IP (VoIP) system, based on Pipes and Filters Architectural pattern. Section 6 describes how it is possible to learn about this Architectural pattern and the Architectural design that made it. Finally, in Section 6 we draw some conclusions and future work.

2 Related Work

In this research there are several involved areas: Architectural Description Languages (ADL), Semantic Frameworks, Verification of Contracts, Interface Description Languages (IDL) and Learning using Semantic Web Technologies. In the case of frameworks and ADL's, the closest work found was Acme-Armani [1] [11]. Acme-Armani is a graphical framework which defines its own ADL and supports Pipe and Filter Architecture. Acme framework supports annotation of architectural structure with arbitrary lists of properties and embodies the archi-

tectural ontology, providing a semantically extensible language. However, this framework never checks the architectural pattern and assembling using Semantic Web Techniques, and neither focus on learning. Another graphical framework system is Aesop [10], where users can develop environments with pattern-specific architectural design. In this framework it is possible to check that compositions of design elements satisfy the topological constraints of the pattern. But Aesop does not check composition using Semantic Web Technologies and it is not focused on learning. In the field of Semantic Web Technologies we found the work of Knight et al [15] about an Ontology-Based Framework for Bridging Learning Design and Learning Content. In this paper authors emphasize the use of ontologies to capture all learning designs, learning objects, and the relations between them, and show how this use of ontologies can result in tools that increase the level of reusability.

3 Chichen-Itza II: a Semantic Framework

Chichen-Itza II is a semantic framework and graphic tool where systems based on Pipes and Filters architectural pattern can be created and verified. We have used the Jena API and the Java programming language along with the IDE NetBeans 7.0. One of the most important features of this framework is enabling knowledge reuse in Pipes and Filters architectural designs using Semantic web techniques [7]. This framework facilitates the process of learning about the system created based on Pipes and Filters Architectural pattern during its assembling process. Messages are displayed if the user tries to connect invalid components and explain how to correct them. Moreover, in this framework it is possible to study the Pipes and Filters Architectural pattern (Components, Connectors and its Rules of Assembling). Our main contributions are threefold. First, we developed and extended a new version of Chichen-Itza framework that allows us to reuse the knowledge of models created based on Pipes and Filters Architectural pattern. Second, our approach supports the assembling validation based on contracts during the assembling components of the system and third, we provide an environment in which people can learn about Pipes and Filters Architectural patterns and the systems developed using this pattern. The process of verification, within the Chichen-Itza framework, is done at a very high level, using the ontologies information among the components of the Pipes and Filter Architectural design to be assembled. Each component is represented in a graphic way. That information, in the ontology populated and created, is evaluated and after that the reasoner verify if it is correct. In addition, we capture this new knowledge and it is saved in the ontology architectural models repository.

4 Verification process in Chichen-Itza II framework

The process to verify the assembling among Pipes and Filters components is easy for an user who is building the system based on this pattern. He introduces his model into the framework by means of a file or selecting its icon in a palette menu.

Chichen-Itza II transforms its vocabulary (filters and pipes that the user needs for building its system) from a text file into ontology instances. The components instances created with classes defined in the Pipes and Filters Ontology are associated with object and datatype properties. The verification process consists of four steps, showed in Fig. 1. This process is described in the next sections.

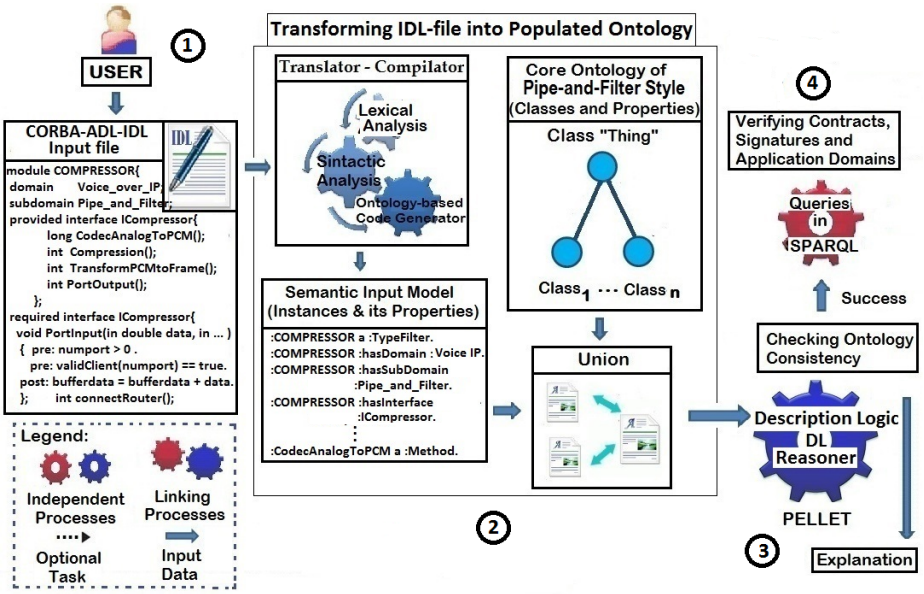

Fig. 1. Verification process model in Chichen-Itza II framework

4.1 Step 1: Creating the CORBA-ADL-IDL Input file

CORBA(Common Object Request Broker Architecture) is a standard created by the Object Management Group (OMG)[5] that enables software written in different programming languages to work among themselves by means of their interfaces. These interfaces are described using the Interface Definition Language (IDL). For each component of the model to verify it is necessary to define a file IDL with its interfaces written using the concepts, rules and properties defined by the Pipes and Filters architectural pattern. It works at the same time as an ADL (architecture description languages). For example, a Compressor component in a Voice over IP system has an ADL-IDL file, and in this file the words *domain*, *subdomain*, *provided*, *interface* and *required* are the keywords of the CORBA IDL-ADL language. An example of this file is showed below.

```
module COMPRESSOR{
       domain        Voice_over_IP;
```

```
        subdomain    Pipe_and_Filter_Architectural_pattern;

         provided interface ICompressor{
             long CodecAnalogToPCM();
             int  Compression();
             int  TransformPCMtoFrame();
             int PortOutput();
         };
         required interface ICompressor{
             void PortInput(in double data, in numport )
             {  pre: numport > 0 .
                pre: validClient(numport) == true.
                post: bufferdata = bufferdata + data.
             };
             int connectRouter();
        };
};
```

For each CORBA-ADL-IDL file created it has to be loaded in the framework and transform into an n3 ontology. This is possible in Chichen-Itza because it has a Compiler-Translator and the user only has to select in the main menu the option "ADL-IDL Translator to n3".

4.2 Step 2: Transforming Input file into a Populated Ontology

Ontologies are the key for Semantic Web goals [2]. An Ontology defines the basic terms used to describe and represent an area of knowledge (in this case a PFAP), as well as the rules for combining terms and relations used to define extensions to the vocabulary. Thus, ontologies define the vocabulary and the meaning of that vocabulary. More specifically, an ontology is a formal representation of knowledge with semantic content which allows the companies and organizations understand and share information [13]. We propose an ontology called **OntoCorePipeFilter** which has the minimum concepts (Pipe, Filter, Port, Datasource, SinkData) which are required to build a PFAP Models. **OntoCorePipeFilter** ontology was created for capturing and verifying information about the input architectural models during the design of the system. This ontology consists of 19 classes, 10 Object Properties and 1 Datatype Property. For each component of the input architectural model an instance is created. The notation **n3** is used by the ontology, because is a valid RDFS and OWL-DL notation and language. The Ontology is showed in Figure 2 using the Chichen-Itza framework. The ontology files obtained in the previous step have to be join in a single project. This project is created by selecting the menu option "Join Ontologies in a Project" and in the same project the **OntoCorePipeFilter** ontology has to be included.

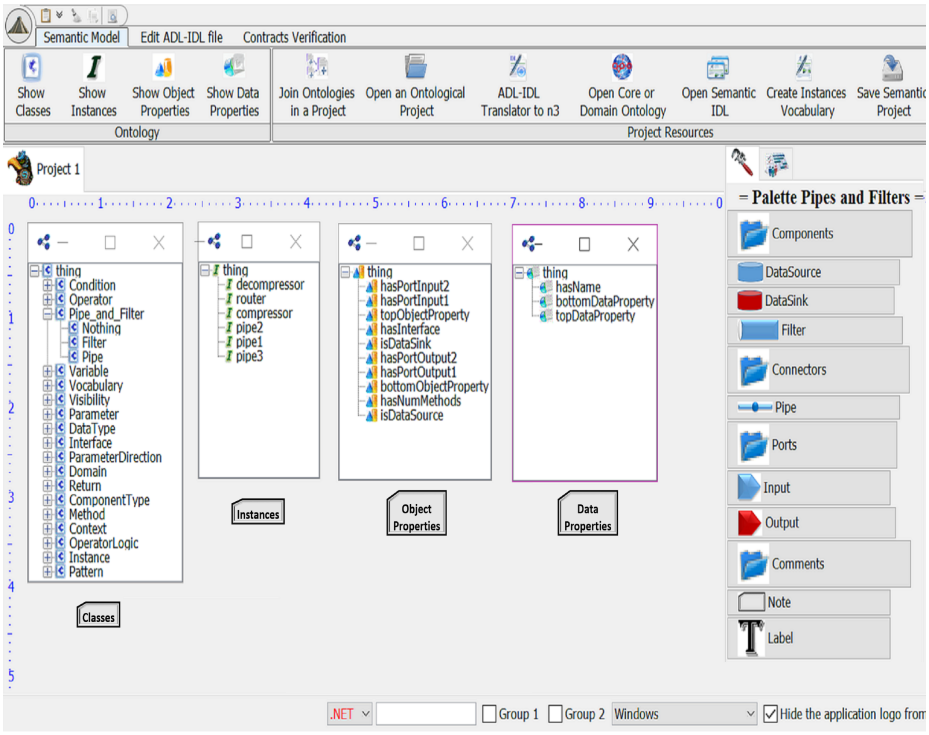

Fig. 2. Ontology classes, instances, object and datatype properties

4.3 Step 3: Ontology verification using The Pellet Reasoner

All instances created, properties (object and datatype) established among instances, and blank nodes in the Ontology are checked by the Pellet reasoner during the consistency verification process. A reasoner is a program which its main task is checking the ontology consistency. It verifies if the ontology contains contradictory facts, axioms or wrong properties among concepts. Besides, new knowledge can be inferred after applied it. The most popular reasoners are Cerebra, FACT++, KAON2, Pellet, Racer, Ontobroker and OWLIM. Pellet [17] is an open-source Java based OWL-DL reasoner. In our verification process we use Pellet for checking the consistency of the ontology created. We select the Pellet reasoner, because it gave an explanation when an inconsistency was detected. Restrictions can be expressed into an ontology. For instance, the following code verifies that one Pipe component has at least 2 ports.

```
:Pipe rdfs:subClassOf
    [ a owl:Restriction ;
      owl:onProperty :hasPorts ;
      owl:cardinality 2 ].
```

Elements of the architectural pattern are subclasses of the **Pipe_and_Filter** class. With this information the reasoner can validate members from different architectural patterns. The property *hasPortInput1* allows the reasoner to validate the connection betwen Filter and Pipe.

```
:Pipe_and_Filter          a owl:Class .
:Pipe           rdfs:subClassOf :Pipe_and_Filter .
:hasPortInput1 a owl:ObjectProperty ;
            rdfs:domain   :Filter ;
            rdfs:range     :Pipe .
```

The *disjointWith* property allows to verify restrictions based on rules of the architectural pattern. For example, a Pipe component is not a Filter component and these two classes are different and are another example based on the rules of architectural pattern. It is defining constraints for assembling Pipes with Tubes only. Defining disjoint classes are also possible.

```
 :Pipe rdfs:subClassOf   :Pipe_and_Filter ;
          owl:disjointWith :Filter .
```

4.4 Step 4: Semantic verification based on Contracts using SPARQL queries

Semantic verification is the process which uses an Ontology and Semantic Technologies (SPARQL queries) to guarantee the correct construction of the system with specific connections and outputs. The semantics of assembling the elements of the architecture are described with object properties. An important aspect of the Pipes and Filters elements to consider during the assembling is the Input and Output connections by means of ports. A pipe element has one output and one input port. The connection among elements are based on the output of one using as input in other. The second step after the reasoner has checked the ontology project consistency is to apply a SPARQL query for validating the correct connection. In our case, we have defined a query which describes the Voice Over IP system connections. The query in SPARQL is showed below:

```
PREFIX : <http://www.ejemplo.org/#>
SELECT ?voiceAnalog ?voiceData ?voicePCM ?voiceAnalog2
WHERE
{  :datasource   :hasOutputPort ?voiceAnalog.
   :pipe1            :hasInputPort  ?voiceAnalog.
   :pipe1            :hasOutputPort ?voiceData.
   :compressor   :hasInputPort  ?voiceData.
   :compressor   :hasOutputPort ?voicePCM.
   :pipe2             :hasInputPort  ?voicePCM.
   :pipe2             :hasOutputPort ?voicePCM.
   :router            :hasInputPort  ?voicePCM.
   :router            :hasOutputPort ?voicePCM.
   :pipe3              :hasInputPort  ?voicePCM.
   :pipe3              :hasOutputPort ?voicePCM.
   :decompressor :hasInputPort  ?voicePCM.
   :decompressor:hasOutputPort ?voiceAnalog2.
```

```
   :pipe4              :hasInputPort  ?voiceAnalog2.
   :pipe4              :hasOutputPort ?voiceAnalog2.
   :datasink         :hasInputPort  ?voiceAnalog2.
}
```

Of course, all this process is transparent to the user. He does not need to know anything about ontologies, reasoners or SPARQL queries, only the manager of the ontology system has to know about that. We could think that SPARQL is the version of SQL for ontologies. Besides, we can use variables in the queries, constraints, filtering information, logic operators, if statements and more. Each N3-triple (each line after) is linked by variables which begin with a question mark. For example *?voiceAnalog* and *?voiceData* are variables. The same name of a variable imply the same value to look for in the query. We can execute and edit queries in Chichen-Itza framework because the Jena API allows us to use SPARQL queries in our framework programmed in Java language. The last step when the ontology project has been verified, consists of saving it in the repository of ontology projects. It is important to note that this repository increases the reuse of these ontology projects and decreases the time in the development of future models based on the same architectural pattern. This being an economic benefit to companies. In our example, the code included in the core ontology is using *N3*-triples notation. In the code above, there are mainly two properties: *hasOutputPort* and *hasInputPort*. First, the triple is formed by a component instance from the architectural pattern (in this case called *:datasource*), second the name of the property *hasOutputPort*, third a variable and finishes with a period.

5 Building a Voice over IP (VoIP) system, based on Pipes and Filters Architecture pattern

Voice over Internet Protocol (VoIP), is a technology for the delivery of voice calls using a broadband Internet connection. This system can be assembled in modules that follow the Pipes and Filters configuration [20]. The basic elements of this system are: a Codec Analog to PCM conversion, a Compression algorithm and Converter PCM to Frame, a Modem or Router to connect with Internet Network, a Decompression algorithm, Converter Frame to PCM and finally a Codec PCM to Analog conversion [6]. Each element is a software component which is assembled using contracts [9]. In Pipes and Filters Architectural patterns these components are represented by a Filter which connect them with the next component by means of a Pipe. In addition, the Speaker data source and the Listener data sink components are connected at the ends. The users can click on any images of the system assembled and the pop-up window is open to explain the characteristics of it. This configuration and pop-up windows opened are shown in Figure 3.

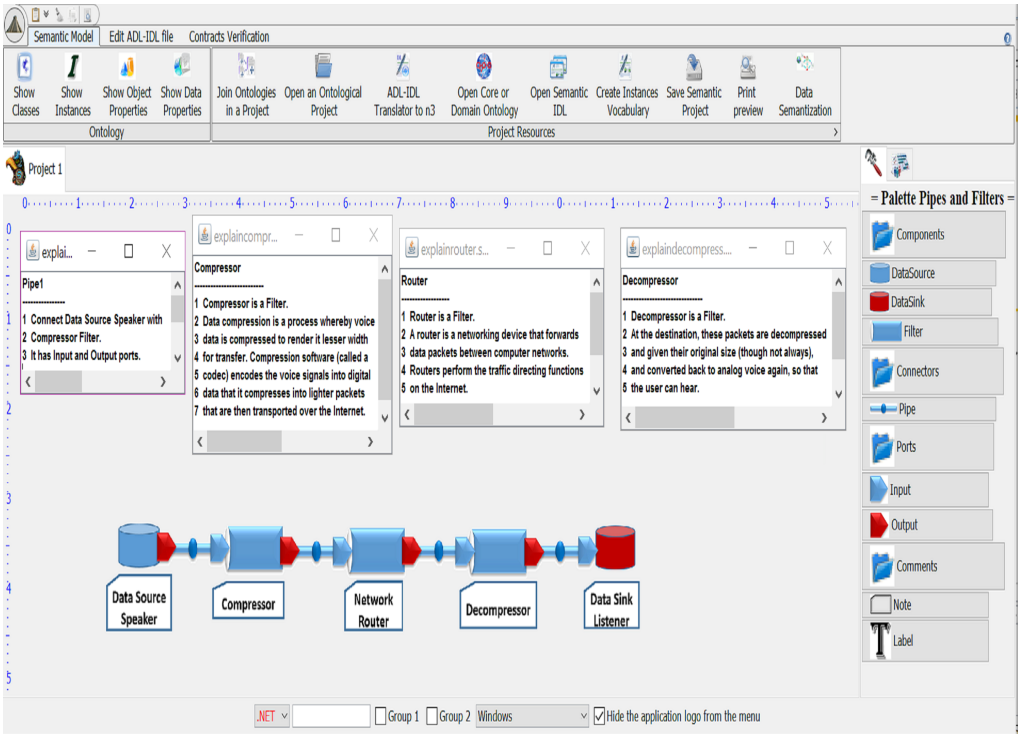

Fig. 3. Pipes and Filters Architecture for Voice over IP

6 Learning in Chichen-Itza II framework

The information of components from the architectural design is saved in a repostory. It contains the populated and annoted ontology with information about the system created. During the assembling process it is possible to know the information from each component which has been assembled by only giving a click over it. The next code shows the corresponding SPARQL query executed by retrieving the information from the component.

```
:router   a :Filter ;
    rdfs:label "Filter router" ;
    rdfs:comment "1  Router is a Filter.";
    rdfs:comment "2  A router is a networking device that forwards";
    rdfs:comment "3  data packets between computer networks.";
    rdfs:comment "4  Routers perform the traffic directing functions";
     rdfs:comment "5  on the Internet.".

PREFIX       : <http://www.ejemplo.org/#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
SELECT ?Router WHERE
{
  :router rdfs:comment ?Router .
}ORDER BY ?Explain
```

Information about each component, represented as an instance in the ontology, can be annoted by the "comment" tag.

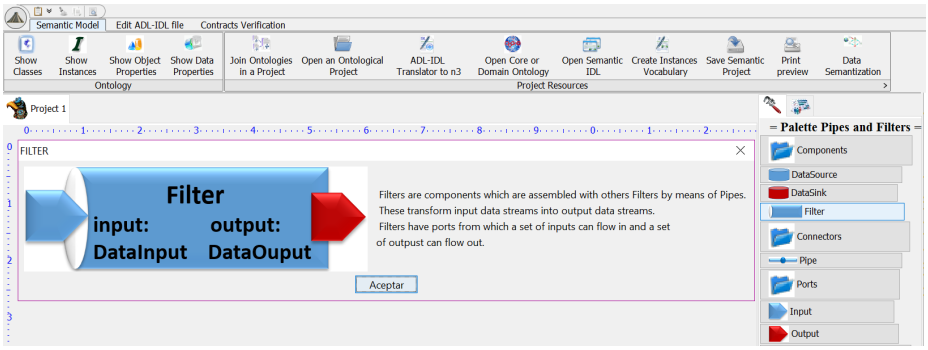

Fig. 4. Learning Filter components from OntoCorePipeFilter Ontology.

7 Conclusions and future work

Semantic Web Techniques in organizations and companies based on Ontologies, Reasoners and semantic Queries are possible by means of core ontologies, reasoners, and SPARQL queries. Ontologies are usually expressed in a logic-based language (Description-Logic). Core Ontologies give more expressive meaning, maintaining computability, do not require the validation of experts or apply a complex methodology for its construction. This core ontology for Pipes and Filters architectures increases the reuse of it and decreases the time in the development of future circuits. The use of a core ontology of Pipes and Filters Architectural Patterns allows us to validate a design based on this architecture and we can verify the correct assembling of its components using the Pellet reasoner and a SPARQL query with semantics 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 an extended and new version of the prototype called Chichen-Itza II and described Semantic Web Techniques used for verifying the Architectural Design based on Pipes and Filters architectural pattern. We illustrate this through a VoIP example.

8 Acknowledgments

This study was partially supported by different grants from Mexico's PRODEP and CONACYT.

References

1. Abi-Antoun, M., Aldrich, J., Garlan, D., Schmerl, B., Nahas, N., Tseng, T.: Modeling and implementing software architecture with acme and archjava. In: Proceedings of the 27th international conference on Software engineering. pp. 676–677. ACM (2005)
2. Berners-Lee, T., Hendler, J., Lassila, O., Others: The semantic web. Scientific American 284(5), 34–43 (2001)
3. Castillo-Barrera, F.E., Medina-Ramírez, C., Durán-Limón, H.A., Gayo, J.E.L., Sadjadi, S.M.: Verifying the behavioral contracts among components by means of semantic web techniques. In: Proceedings of the International Conference on Software Engineering Research and Practice (SERP). p. 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp) (2012)
4. Clements, P.C.: Software architecture in practice. Ph.D. thesis, Software Engineering Institute (2002)
5. CORBA, O., Specification, I.: Object management group (1999)
6. Davidson, J., Peters, J., Gracely, B.: Voice over IP fundamentals. Cisco press (2000)
7. Davies John, Stunder Rudi, W.P.: Semantic web technologies trens and research in ontology-based systems (2006)
8. Di Nitto, E., Rosenblum, D.: Exploiting adls to specify architectural styles induced by middleware infrastructures. In: Proceedings of the 21st international conference on Software engineering. pp. 13–22. ACM (1999)
9. Duran-Limon, H., Meda-Campaña, M.E., Sapien-Aguilar, A.L., Piñón-Howlet, L.C.: Un marco de trabajo de una fábrica de software para el reuso del diseño arquitectónico y de componentes de software (2008)
10. Garlan, D., Allen, R., Ockerbloom, J.: Exploiting style in architectural design environments. SIGSOFT Softw. Eng. Notes 19(5), 175–188 (Dec 1994), http://doi.acm.org/10.1145/195274.195404
11. Garlan, D., Monroe, R., Wile, D.: Acme: an architecture description interchange language. In: CASCON First Decade High Impact Papers. pp. 159–173. IBM Corp. (2010)
12. Gomaa, H., Farrukh, G.: Composition of software architectures from reusable architecture patterns. In: Proceedings of the third international workshop on Software architecture. pp. 45–48. ACM (1998)
13. Gómez-Pérez, A., Fernández-López, M., Corcho, O.: Ontological engineering with examples from the areas of knowledge management,e-commerce and the semantic web (2003)
14. Kicillof, N., Yankelevich, D.: Detecting and solving architectural problems with jacal
15. Knight, C., Gasevic, D., Richards, G.: An ontology-based framework for bridging learning design and learning content. Educational Technology & Society 9(1), 23–37 (2006)

16. Oreizy, P., Medvidovic, N., Taylor, R.N., Rosenblum, D.S.: Software architecture and component technologies: Bridging the gap. In: Proceedings of the Workshop on Compositional Software Architectures (1998)
17. Parsia, B., Sirin, E.: Pellet: An owl dl reasoner. In: In Proceedings of the International Workshop on Description Logics (2004)
18. Shaw, M., Clements, P.: The golden age of software architecture. IEEE Software 23(2), 31–39 (March 2006)
19. Vestal, S., Binns, P.: Scheduling and communication in metah. In: Real-Time Systems Symposium, 1993., Proceedings. pp. 194–200. IEEE (1993)
20. Zave, P.: Audio feature interactions in voice-over-ip. In: Proceedings of the 1st international conference on Principles, systems and applications of IP telecommunications. pp. 67–78. ACM (2007)