CARL: A Complex Applications Interoperability Language based on Semantic Technologies for Platform-as-a-Service Integration and Cloud Computing

Enrique Jiménez-Domingo
enrique.jimenez@uc3m.es
Computer Science Department
Universidad Carlos III de Madrid
Av. Universidad 30, Leganés, 28911, Madrid, SPAIN
Phone: +34 91 624 5936

Juan Miguel Gómez-Berbís
juanmiguel.gomez@uc3m.es
Computer Science Department
Universidad Carlos III de Madrid
Av. Universidad 30, Leganés, 28911, Madrid, SPAIN
Phone: +34 91 624 5958

Ricardo Colomo-Palacios (Corresponding Author)
ricardo.colomo@uc3m.es
Computer Science Department
Universidad Carlos III de Madrid
Av. Universidad 30, Leganés, 28911, Madrid, SPAIN
Phone: +34 91 624 5958

Ángel García-Crespo
angel.garcia@uc3m.es
Computer Science Department
Universidad Carlos III de Madrid
Av. Universidad 30, Leganés, 28911, Madrid, SPAIN
Phone: +34 91 624 9417

ABSTRACT

Cloud Computing technologies have recently gained momentum and moved from a hyped trend to a mature set of technological innovations providing an infrastructure for the Software-as-a-Service (SaaS) paradigm. However, a number of questions about Cloud-deployed applications interoperability and cross-integration, stemming from multiple areas of Computer Science domains, have also been raised from a Platform-as-a-Service angle. In this paper, we present CARL, a novel, cutting-edge, interoperative and integration-oriented language based on Semantic Technologies. We provide a formal model and semantics to enable Complex Application management and integration in PaaS environments, together with a thorough demonstration of the breakthroughs being provided by CARL.

Keywords: Semantic Web, Ontologies, Cloud Computing, SaaS, PaaS, interoperability.

ACM Computing Classification: D.2.12, D.2.11, I.2.4

1. Introduction

The Internet of services is turning the Web into a new platform for business services and transactions based on information exchange. In recent years there has been an increasing trend for large organizations to present their Business Processes through the Web for the large-scale development of software, as well as to share their services within and outside the

organization. However, new paradigms for software and services engineering such as Cloud Computing and SaaS and PaaS models are promising to create new levels of efficiency through large-scale sharing of functionality and computing resources. Cloud Computing can be defined as an IT infrastructure provisioning model in which applications and services from many organizations are hosted in a single large-scale facility. Tightly related to Cloud Computing are SaaS and PaaS models. SaaS is a software distribution model in which, besides providing software, the supplier offers additional services such as maintenance and support. The advantage of SaaS is that the software is distributed and hosted on the Internet, so eliminating the requirement for the users to install the software in their own computing infrastructure, or any related data either. PaaS is one of the three provision service models that can be defined as the set of platforms composed of one or more application servers and databases that offer the possibility of executing applications. The work presented focuses on the integration of applications at PaaS level, there will be special emphasis on the concepts related to it.

Cloud Computing and these models open the doors for large economies-of-scale, but they face a number of challenges. Foremost among these are: (i) the lack of proven models for determining under what conditions it is cost-effective for IT user organizations to migrate towards these models, taking into account legal, business and technical factors; (ii) the lack of proven methods (e.g. architectural guidance) for facilitating this migration; and (iii) the lack of integration methods at different levels which make the process of developing and delivering software able to communicate with other parts of the cloud platform harder, so losing one of its most powerful features. This last challenge is far and away the hardest of all and will receive special attention in this paper. Fundamentally, the main reason is that since Cloud Technologies emerged, transforming the ways we do business and, consequently, redesigning the world, they have faced the same problems as previous waves of technology Web Services or Application-based Enterprise Application Integration (García-Sánchez et al., 2009). One of those problems is the lack of integration and interoperability among different applications and their enabled cross-communication.

In addition, specific challenges include how to represent SaaS and Cloud Computing capabilities and requirements, and how to enable brokers to match between such capabilities and requirements.

This work seeks to bring these technologies closer together and solve the problem of integration at application level with the challenges mentioned. It presents an original approach based on semantic technologies and languages - the design of a semantic declarative language able to capture the content of the applications uploaded to a cloud platform and also able to understand the data requirements for establishing a nexus between applications which allows the communication between them, so making the most of the characteristics provided by the paradigm. Summarizing, our work addresses one of the hardest problems on the Web - the integration of heterogeneous applications - a that the Semantic Web has been confronting since its very beginnings Semantic technologies deal with adding machine-understandable and machine-processable metadata to Web resources through its key-enabling technology: ontologies. Ontologies are a formal, explicit and shared specification of a conceptualization. Ontologies acquire high importance in this work as the tool that will allow us to represent the knowledge in the domain and be able to recognize the necessary aspects to establish communication between applications in the cloud environment. A design of the ontology used is provided in the “Conceptual model” section in order to show the information needed to achieve the goals.

The aim of CARL is to gather these emerging concepts (SaaS, Semantic Technologies, Business Process Modeling, Cloud Computing and interoperability) to foster dramatic evolution of new platforms oriented towards application interoperability and cost reduction which can impact significantly on industry. Thus, CARL can be defined as a complex application interoperability

language based on semantic technologies for Platform-as-a-Service integration and Cloud Computing.

The remainder of the paper is organized as follows. Section 2 outlines the relevant literature in the area including Cloud Computing, declarative languages and Semantic Web. Section 3 describes the conceptual model and requirements. Section 4 presents the architecture of CARL. Section 5 describes the testing performed in order to ascertain the suitability of the systems and section 6 concludes the paper.

2. Background

Giving the ambitious aims of this system, it is necessary to establish different points of view when talking about the related work behind it. Three main concepts and paradigms are involved in this project, namely, Cloud Computing (SaaS and PaaS), declarative languages and Semantic Web. By combining the state of the art of all of them, and finding a new breakthrough in terms of the current state of the art, which is the goal of our project, it is possible to build the language and system we are focusing on, with an orientation towards interoperability, cost reduction and automation, which can impact significantly on industry. A brief review of these technologies is now provided.

2.1 Cloud Computing: SaaS & PaaS

To fully understand the concepts related to Cloud Computing, it is necessary to clarify what Cloud Computing is. The concept of “cloud” is used as a metaphor of the Internet, as an abstraction of the complex infrastructure that it represents as appears in Armbrust et al. (2009). For this reason, as mentioned in Hayes (2008), we can assert that Cloud Computing is the paradigm or the information technology model which offers computational services throughout the Internet.

Cloud Computing is often defined as a new technology but it is not. It is more appropriate to state that is a new approach that combines known technologies (Buyya et al.,2008a) like operative systems, databases, servers, networks, multitenancy, middleware, virtualization, management tools, etc. There exists a common misunderstanding between Cloud Computing and autonomous computation, grid computing or utility computing. Actually, Cloud Computing usually depends on grids for its implementation. It has some autonomy (it reacts when the demand increases and readjusts the resources automatically) and it is charge on demand. Otherwise, Cloud Computing has a higher scope as pointed out in Foster et al. (2009). The essence of Cloud Computing is not focused so much in the tools that it uses (software, platforms, technologies...), but in the way it uses, groups and makes up by unifying some of the most important principles for creating a proper dynamics, as shown in Buyya et al. (2008b).

Nowadays some systems have merged, using these approaches and presenting different points of view or focusing on different aspects of this paradigm. In this context we can find a study (Buyya et al., 2009) where an extensible simulation toolkit that enables modelling and simulation of Cloud Computing environments is presented or (Lenk et al., 2009) where an integrated Cloud Computing architecture is proposed.

Within Cloud Computing paradigm, three provision service models can be distinguished (Vaquero et al., 2009): Infrastructure-as-a-Service (IaaS), Platform-as-a-service (PaaS) and SaaS depending on the levels of services provided. It is important to underline that any one of these models can exist in an isolated environment, so it is imperative to understand the close relation between them (Keller and Rexford, 2010). IaaS is the base of all the services in the cloud and is focused oat network architects level; PaaS is articulated on top of IaaS establishing it in an

application developer level and finally SaaS is also articulated on IaaS and it appears as the front-end for final users.

Software-as-a-Service (SaaS) (Campell-Kelly, 2009) is a term that refers to a software distribution model where the supplier of the service provides, besides the software itself, other services like maintenance, help and support. The main advantage is that the required software is distributed and placed on the Internet. Thanks to this, the user does not need to install this software and other aspects like security, quality of service and performance are completely managed by the service provider. In other words, the client has his system hosted in the IT company (Knorr and Gruman, 2009). Due to this, in functional areas like human capital management, SaaS continues to grow at two to three times the pace of on-premises (Galinec, 2010). This change in the conception of software distribution brings great advantages like high scalability, drastic decrease in costs and a higher efficiency in terms of performance and availability for the corporative information systems in any segment of the market (Armbrust et al., 2009). A further interesting aspect is that the use of the application is carried out in centralized servers, instead of being hosted by clients' locations, who can access the services over the Web.

Furthermore, in relation to data storage, it should be mentioned that user data are stored on an external server, thereby ensuring that the client does not require large storage capacities when working with large quantities of information, such as metadata (Aymerich et al., 2008). This also guarantees that the software user always has secured copies of his data, entrusting the supplier company with responsibility for data storage, without having to be concerned about the data. The distribution model is based on the premise that the supplier company offers the maintenance and support service. This entails the assumption that all of the information, processing and business logic is stored in the same location, a fact from which ICT companies greatly benefit, given that the lack of such a concept (which is so often the case) frequently leads to the dispersion and lack of integration and communication between distinct business processes. We are then in a model that joins products and services to provide a complete solution for the user that optimizes costs and resources.

The most well-known case of a company working with this paradigm is Salesforce (2009), which offers SaaS applications through its Cloud Computing platform and it has been a leader of this movement from the very beginning, providing different services depending on client needs. An interesting economic study of SaaS is presented in Vaquero et al. (2009) where all the implications of investments are clearly set out.

PaaS can be defined, as commented before, as the set of platforms composed of one or more application servers and databases (the existence of a database in the platform is not necessary) that offers the possibility of application execution. The provider will be responsible for the resources scale and, when necessary, for maintaining an optimal performance of the platform, ensuring the security of it, etc.

There are not many works that centre their efforts at the PaaS level but it is interesting how Lawton (2008) expresses the possibility of creating software based on this part of the Cloud Computing paradigm due to the fact that, in terms of configuration and customization, PaaS level allows designers to build the platform according to the features and needs that they consider in each case. CARL benefits from most of its intrinsic features (Web based user interface creation tools, Web services and databases integration, Support for development team collaboration, Integrated environment to cover the application cycle from development to maintenance…) and establishes the foundations of the applications of semantic interoperability enabled by PaaS.

2.2 Declarative languages

A declarative language can be defined as a programming language based on Mathematics and Logics of imperative languages near to human reasoning, due to its not pointing out how to perform a task, but what has to be done (Dekel et al., 2005). In this sense, declarative languages provide an interesting approach in the development of a variety of paradigm systems as indicated in Hanus (2007).

Declarative languages have been used in several ambits and scopes due to their flexibility and adaptability to different environments. The works already done in Cloud Computing systems are limited but it is possible to find the work of Alvaro et al. (2010), where a declarative programming model for distributed environments like cloud is addressed.

The most relevant works in this field can be found in approaches to the security of the Web (Abadí and Thau Loo, 2007) or for structuring its information, as in Lakshmanan et al. (2006) where a simple logic system based on declarative languages is capable of retrieving information from HTML documents in the Web. It is also possible to find some approaches for business. This is the case of Pesic et al. (2006) where a declarative approach is proposed to reach a fundamental paradigm for a flexible dynamic process management, due to the property of these models for specifying what should be done instead of how, so facilitating the work of the users who in most cases are not able to complete these tasks. Nearer to CARL, Panetto et al. (2004) tackle the problem of interoperability in enterprise models by creating a modeling language that serves as an Interlingua for the communication between the different entities that take part in the process.

2.3 Semantic Web

Semantic Technologies are a new way of supporting knowledge in a wide range of domains, including recommender systems (García-Crespo et al., 2010), biomedical (García-Sánchez et al., 2008; Ruiz-Martínez et al., 2011), security (Blanco et al., 2011; García-Crespo et al., 2011), innovation (Colomo-Palacios et al., 2010) or software engineering (García-Crespo et al., 2009), to cite just a few of the most recent and relevant works. The knowledge representation technology used in the Semantic Web is the ontology, which formalizes this meaning and facilitates the search for contents and information (Jiang and Ah-Hweetana, 2009), as well as improving crawling (Zeng et al., 2008) by providing a common framework which enables data integration, sharing and reuse from multiple sources. A number of different ontology definitions can currently be found in the literature. In this work we adopt the following: “ontology is a formal and explicit specification of a shared conceptualization” (Studer et al., 1998). In this context formal refers to the need for machine-understandable ontologies. This definition emphasizes the need for agreement in carrying out a shared conceptualization. Besides, the ontology language selected in this work was OWL (Web Ontology Language) (McGuinnes and Van Harmelen, 2004), which is the ontology language recommended by the World Wide Web Consortium (W3C).

There are many ontologies with different purposes that cover a concrete domain, for example this is the case of BORO (Business Object Reference Ontology) (Partidge and Stefanova, 2001), an ontology that is intended to be suitable as a basis for facilitating, among other things, the semantic interoperability of an enterprise’s operational systems, or another example from a completely different domain like security in the publication of Vorobiev and Bekmamedova (2010), where a ontology-driven approach allows the enforcement of the security of information. More within the domain of Cloud Computing, we find in Youseff et al. (2008) one of the first attempts to establish an ontology of the cloud and to provide a better understanding of the technology so as to develop more efficient portals and gateways for cloud environments.

When dealing with the application of semantic technologies it is important to comment that the sheres in which they are used are varied and range from a recent application of Semantic Web techniques to access control in networks (Fitzgerald et al., 2009) to integration and analysis models for information systems (Sheth and Ramakrishnan, 2003) or even applications in e-business, as in Singh et al. (2005). Semantic web has also been used in the creation of models for interoperability in the Web as presented in Melnik et al. (2000), where a layered approach is used.

Focusing now on a Cloud Computing environment, there are few works that use semantic technologies to exploit its features but in Ahlmann and Jaatun (2009) we find an interesting survey on the potential of a semantic cloud if privacy challenges are solved. Other approaches have been taken in García-Sánchez et al. (2010), which are more in line with the objectives of this paper where the semantic technologies assume a leading role in the achievement of the proposed goals, combining their capacity with the advantages of SaaS and Cloud Computing and laying the foundations for cloud platforms able to support all this potential and make it available to the user.

3. Conceptual model and Requirements

In this section, we provide first an introductory paragraph about the main source problems that motivated this research. Then we will go into a deeper explanation of the conceptual model and requirements developed in this work. A contextualized and well-referenced description of these elements concludes this section.

3.1 Source Problems

Most applications are aimed to interoperate and to be integrated into more complex systems where their functionality could be appraised with a higher added-value. Cloud Technologies, and particularly, Platform-as-a-Service (PaaS) infrastructures, are a perfect example of such an ecosystem.

Hence, the problems we are trying to deal with in this work stem from the lack of three major aspects of interoperability:

- Lack of Data Interoperability: Applications need to share common understanding and common grounds in terms of the input and output data they exchange.
- Lack of Process Interoperability: Applications should understand and share a common logic of interweaving communication, not only at a communication protocol level, but also from a “control flow” perspective. Notions like sequence, parallelism, conditional communication should be rooted in similar groundings.

Once these two major interoperability issues have been dealt with in the next sections, a number of challenges devised will be also outlined and analyzed.

3.2 Requirements

When introducing requirements of a particular software endeavor, two aspects must be considered. First, it is important to know what is encompassed by those requirements and how current related work and state-of-the-art technologies may fulfill them. Hence, we will first give the rationale for the importance of these requirements by providing a background and context for them. Then, we will explain why they are not fully addressed by the current state of the art.

Our first Requirement is a conceptual description of a Complex Application (CA), which will be described below. A CA is considered one of the entities involved in the application platform and

the main entity of the Conceptual Model. A CA must be made explicit from a conceptual viewpoint and it must also be reflected by the underlying syntax.

The second requirement is Potential Transactionality (PT). PT implies that a number of CAs might hold a number of ACID transactions properties. Those ACID properties encompass Atomicity, Consistency, Isolation and Durability. The adjective "Potential" refers to the fact that CA might hold or profit from these properties but that is not a sine qua non condition to perform in the CARL environment. However, there should be mechanisms in place to enable CAs to benefit from potential "commit" or "roll-back" mechanisms.

A third requirement is Potential Security (PS). Security is a vital requirement when designing a communication mechanism. Given the intrinsic nature of information exchange, privacy and confidentiality must be guaranteed from non intended members of this exchange. Security is a broad topic and covers a multitude of facets and social implications, but in our context, security is defined through four main concepts: secrecy, authentication, non-repudiation and integrity control. Secrecy is about keeping information out of the hands of unauthorized users. Authentication deals with determining the identity of the participants before revealing sensitive information or entering into a business negotiation. Non-repudiation tackles signatures, for instance, how to prove a customer placed an electronic order in the specific terms of the contract, when he claims the opposite. Finally, integrity control enables us to make sure that a received message is what was sent, and not a modified message from a malicious adversary. These four issues can be summarized as the need to ensure that the transmitted information is encrypted and secured and only Sender and Receiver follow the conversation. Secrecy can be achieved by encrypting the information. In a communication network, encryption takes place at the highest level, from which the information can be encrypted and then communicated. There are many encryption techniques based on different cryptographic algorithms and combinations of them. Currently, two types of algorithms are used: symmetric key and asymmetric key algorithms. Symmetric key algorithms use the same key for encrypting and decrypting the information. They are fast but need to distribute the key and the key could be stolen or revealed. Asymmetric algorithms use a public key, which is of public domain and known by other users and a private key, known only to its owner. Messages are encrypted by the public key of the sender and de-encrypted by the private key of the receiver. Currently, the most common techniques used are Pretty Good Privacy (PGP) or Secure Socket Layer (SSL) (Rescorla, 2001). Claiming Potential Security (PS), similarly to Potential Transactionality (PT), implies that most of the CAs included in the platform might or might not include security as a basic feature and constraint, but there must definitely be a means of implementing security if needed in the CARL scope.

The fourth requirement is the Conceptual Compliance of CARL. We define Conceptual Compliance as the actual compliance of the CARL approach with a particular conceptual model. Conceptual Compliance is difficult to be gauged, but in Computer Science, it is vital to distribute a particular vision throughout an ambitious research effort, like CARL, which will encompass a software architecture and implementation. A conceptual model is a set of concepts and relationships which describes a specific domain. Conceptual modeling is a very active research area and has been the subject of intensive research activity, especially by the Knowledge Representation research area (Patil et al., 1992).

Our conceptual model consists of a set of concepts, relationships and instances. We will use a UML graphical notation to represent it, but it will be accompanied by a textual description of the concepts. This model is the conceptual grounding for the CARL underlying language that will be rooted in Semantic Technologies, that is, by means of the CARL ontology.

A problem often encountered when trying to define relationships between concepts is the lack of clarity of the definition of a concept, which relationships a concept can present and how

individuals of these concepts can be categorized. A common example is the following. If “Plane” is a concept, i.e., an abstraction which represents all planes, a “Boeing 747” may be an individual of this concept, i.e., the “Boeing 747” that has left the airport or a concept encompassed by the “Plane” concept, i.e., a subconcept, which represents all “Boeing 747” planes.

For the sake of clarity, we make use of a well-known and widely meta-model to describe the different levels of abstraction of our conceptual model. This meta-model is the Meta Object Facility (MOF) (MOF, 2002), a specification to define abstract languages and frameworks for specifying, constructing and managing technology-neutral meta-models. MOF defines a meta-data architecture consisting of four layers as follows:

- Information layer: This comprises the data that we want to describe.
- Model layer: This comprises the meta-data that describes data in the information layer.
- Meta-model layer: This comprises the descriptions that define the structure and semantics of the meta-data.
- Meta-meta-model layer: This comprises the description of the structure and semantics of the meta-model layer data.

Hence, the CARL conceptual model has several classes and relationships. The MOF meta-meta-model layer defines the corresponding meaning (semantics) of what we understand by concepts and relationships as the MOF Class construct (and implicitly the sub-Class generalization as construct), and the Relationships (e.g. Association, Aggregation), respectively (MOF, 2002). In the meta-meta-model layer, we only have, then, Class, sub-Class and relation constructs.

Finally, the fifth and last requirement for CARL is to hold a Formal Foundation. In CARL, the Formal Foundation is based on Formal Semantics, which provides meaning to computer programs. This meaning enables reasoning about such programs, based on the mathematical properties of the applied semantics. Reasoning is the process of drawing conclusions from facts

Three main styles of formal reasoning about computer systems, focused on giving semantics (meaning) to programs are defined:

- Operational semantics: A computer program is modeled as an execution of an abstract machine. A state of such a machine is defined as an evaluation of variables. Simple program instructions represent transitions between states (McCarthy, 1963).
- Denotational semantics: Computer programs are just represented as a function between the input and the output (Scott et al., 1971).
- Axiomatic semantics: Programs are proved to be correct using proof methods. Central notations are program assertions, proof triples consisting of precondition, program statement, post-conditions and invariants (Hoare, 1985).

For execution semantics, we follow the first approach, also called operational semantics. Execution semantics describes how the program evolves and behaves, but it is more efficient when described by a formal method. A formal method is used to describe mathematically and reason about a computer system. There are two advantages of describing execution semantics formally.

Firstly, we believe that one advantage of using a formal method to model the execution semantics is that, if the specification is written in a logical language that could use inference, it

is feasible to derive consequences from the specification. Using this inference feature, properties of the specification that were not explicitly stated can be proven. In this way, invisible behavior and properties of the system can be predicted and tested without, for instance, deadlocks or errors needing to be implemented.

Secondly, the verification of properties is another advantage. Two well-established approaches for verification with formal methods are model checking and theorem proving. These formal methods are used to analyze a system for desired properties.

Model checking is a technique that relies on building a finite model of a system and checking that a desired property holds in that model. In other words, an exhaustive state space search check is performed which is guaranteed to finish, given that the model is finite. Given this restriction, model checking must devise algorithms and data structures to handle large search spaces. This technique has been widely used in hardware and protocol verification and recently, the intention has been to use it for analyzing specification of software systems.

Fundamentally, there are two general approaches to model checking, as discussed in (Clarke et al., 1987). The first is temporal model checking. In this approach, specifications are expressed in Temporal Logic and systems are modeled as finite state transition systems. The procedure used is to check if a given finite state transition system is a model for the specification. The second approach is called automaton model checking. It requires an automaton behavior for the specification. The system is, then, also modeled as an automaton, and both are confronted to determine if the system behavior conforms to the specification.

In contrast to theorem proving, model checking is automated and fast, it checks partial specifications, even if the system is just partially designed and produces counterexamples, i.e., situations in which the model does not comply with the specifications, and this can be used in debugging.

The main drawback of model checking is state explosion. State explosion happens when the number of states of the behavioral representation of the limited grows exponentially and reaches a number from which it is difficult to calculate or verify all the states. There a number of methods to alleviate this problem, such as appropriate reduction or abstraction techniques (Clarke et al., 1987), which basically allow checking an almost unlimited number of states.

Theorem proving is used when system properties are specified in a certain mathematical logic. This logic defines a set of axioms and a set of inference rules. Theorem proving is the process of finding a proof of a property from the axioms of the system. This technique is being increasingly used in mechanical verification of safety-critical properties of software designs.

Theorem proving can deal with infinite states if it relies on techniques such as structural induction to prove over infinite domains.

Formal execution semantics is also used as a prescription during the implementation of a system, where it is of the utmost importance that the specification is human-understandable. Otherwise the situation could arise where the specification is perfect, several properties have been checked and verified, but since the developer does not understand the specification correctly, the implementation does not follow the specification and the system does not behave correctly.

Our requirement is to use formal execution semantics in the design of our language. The reasons are twofold: to automatically check the system properties and to help developers understand the specification.

3.3 Conceptual Model

The conceptual model represents the different components that take part in an information system. In this case, the conceptual model refers to the ontology designed to capture the necessary context knowledge for performing the previously defined tasks and requirements.

Figure 1 shows the different classes created in CARL Ontology that allows modeling the interoperability between different applications:

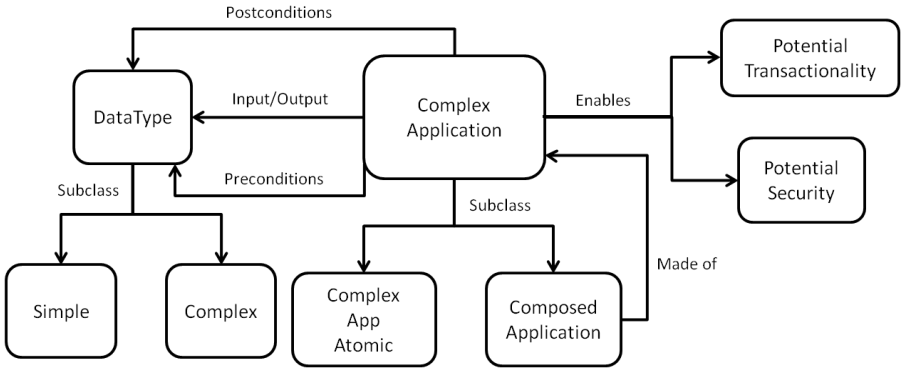

Figure 1. CARL Ontology/Conceptual Model

A brief description of each class in the ontology is provided:

- DataType (DT). This class models the different types of data that can go through the system. These data types will be inputs and outputs of the applications, and allow the ontology to know the requirements of these applications in order to be able to establish the communication between them.
- Simple/Complex. These classes represent the kind of data that are allowed by the applications depending on whether it is simple or complex as normal conventions.
- Complex Application (CA). This is the class that manages the CA that can be interconnected through the language. As can be seen in the figure, this class has a relationship with DT which establishes that a CA works with certain types of data. The most interesting aspect of this relationship is the existence of preconditions and post-conditions within the data types of the complex application. These pre- and post-conditions are axioms of CARL Ontology, which means that it is possible to make logical operations with them and use the power of the ontology to improve the quality of the process and ensure the feature of the whole system. This point clarifies the use of an ontology instead of other language representations for the following reasons. First and foremost, more syntax-based notations, such as BNF or XML, would prevent the use of logical operations and hence fail the requirements stipulated in the previous sections. Second, ontologies can by their own nature easily absorb the conceptual specifications raised in the Conceptual Model section of this work. They also provide an easy integration with inference engines and other knowledge-based mechanisms that will bolster CARL to meet the Data and Process interoperability expectations outlined in the Source Problems section. Finally, ontologies are also a significant underlying concept of the software architecture, by providing solid foundations as a Data input and output mechanism for the different software components.

- Complex App Atomic (CAA). With this class a single application is represented that is susceptible to being interconnected with another. As it is easy to see, it is directly related with CA in order to manage the interoperability process.
- Composed Application (CAPP). This class represents a set of applications that are already connected and all its associated properties. As expressed in the figure, it is composed of several CAA.
- Potential Transactionality (PT)/Potential Security (PS). These classes show possible features that can be enabled by the CA. As PT and PS are requirements established for the language, it is necessary to represent them as classes with their own properties that can vary from one application to another. In each case, these properties will have different values that will be managed by the CA in order to complete the process of interoperation.

With the design of this ontology, all the requirements are fulfilled and the main goals of the language can be reached. The next section will explain the architecture followed by CARL in order to implement this platform.

4. Architecture

In this section, we present the CARL architecture, a three-layer software architecture, which partitions the functionality of the system into Graphical User Interface (GUI), Business Logic and Persistence and Storage Systems level. Each level has a different functionality to tackle with the various challenges CARL faces when enabling Complex Applications Interoperability through Semantic Technologies. The final architectural approach is a tailor-made value-adding technological solution which addresses the aforementioned challenges and provides a basis for the implementation that will be shown in the next section. The CARL architecture is comprises a number of components, depicted in the following figure.

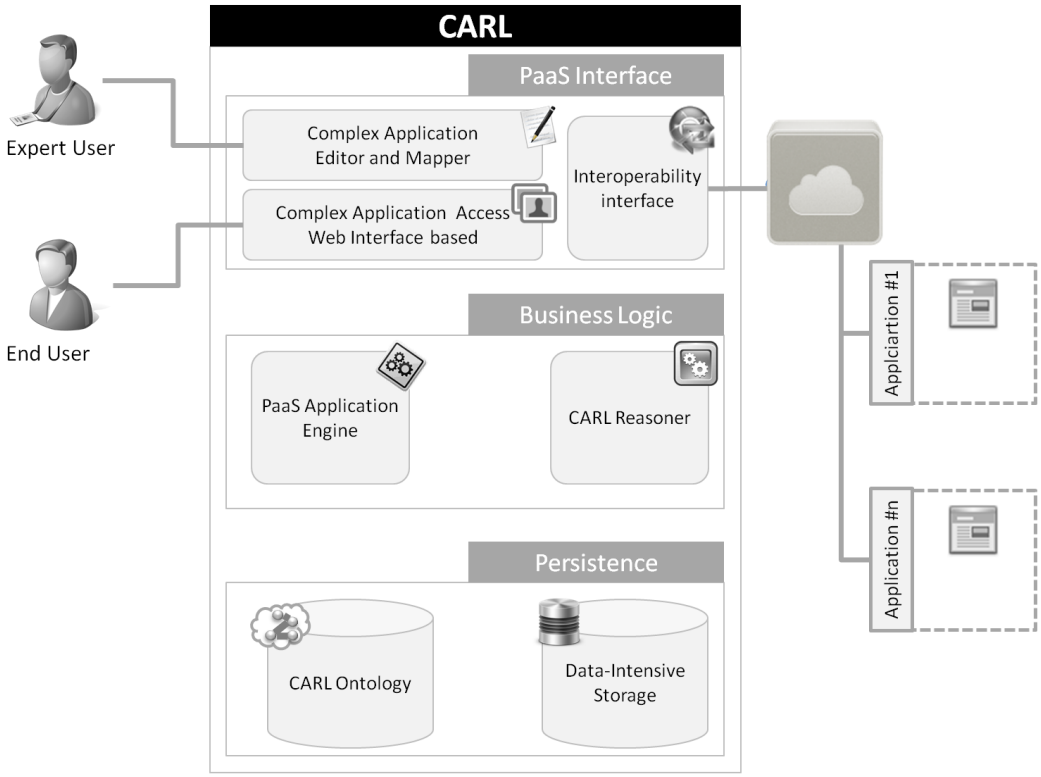

Figure 2. CARL Architecture

We will now detail the different components without focusing on the software layer where they belong. This is not necessary since the three functionalities are well defined and have a commonly shared and used pattern:

- Complex Application Editor and Mapper: This component interacts with a particular expert user by providing a set of graphical elements to annotate the resources by means of semantic annotations based on the CARL ontology schema. This would include input and output, simple or complex data types.
- Complex Application Access Web Interface-based: This component offers an interaction with the Complex Application that the Platform-as-a-Service (PaaS) will always provide for starters.
  This module also provides retrieval-related capabilities by providing a query system for finding Complex Applications, grounded in the CARL reasoner.
- Interoperability interface. This component is in charge of the management of the communication between the applications which allow the interoperability process to be completed.
- CARL Reasoner: This component derives facts from a knowledge base, reasoning about the information with the ultimate purpose of formulating new conclusions. It is based on a particular implementation of an OWL Description Logics based Reasoning Engines - the Renamed A-Box and Concept Expression Reasoner (RACER). It uses subsumption to find sets and subsets of annotations based on logical constraints.
- PaaS Application Engine: The PaaS Application Engine is the core of the operational semantics of the PaaS. It allows the execution of applications and its particular implementation is based on existing approaches for SaaS services.
- CARL Ontology: This software component implies a semantic data store system that enables ontology persistence, querying performed by the Business Logic layer components and offers a higher abstraction layer to enable fast storage and retrieval of large amounts of OWL DL ontologies together with their RDF syntax while keeping a small footprint and a lightweight architecture approach.
- Data-intensive Storage: This software component adds networked online storage where data is stored on the Infrastructure as a Service (IaaS) provided by the underlying Cloud Computing infrastructure.

The CARL language, which is represented by an ontology for the reasons outlined, is parsed and exchanged by the different software components, resulting in the common underlying format for input and output operations between the different building blocks. All these components form the foundations of the system and allow communication between applications by means of CARL language. In the following section a set of test results that prove the performance of the whole system will be shown.

5. Testing

In order to find out if CARL provides desirable outputs, a testing process of the prototype was conducted. The whole testing was carried out in a single multi-core machine whose hardware configuration is AMD Phenom X6@3.2 GHz with 8GB RAM DDR3-1333. This machine is located in our laboratory and it is used as a server and as a testing platform for different research lines.

The testing consisted of a set of executions of fifteen of the basic operations supported by CARL. Each operation was executed a hundred times and the results of the operations were encoded using the statistical analysis tool GNU R. Testing data was designed in order to provide

proper test cases, including main errors that could appear in CARL execution environments. Table 1 shows main statistics from executions.

Table 1: Statistics extracted from the tests performed

	Successful Executions		Unsuccessful Executions			
	%	Exec Time	%	% Rec.	Exec Time	R Time
Open Channel	98	44	2	50%	88.5	74
Accept Open	98	63	2	50%	120.0	81
Operate	80	159	20	5%	294.0	253
Operate Result	98	663	2	50%	1020.5	571
Check	98	41	2	50%	80.5	68
Check Result	98	99	2	50%	156.0	103
Convert	87	302	13	40%	546.6	237
Close Request	98	39	2	50%	71	74
Close Result	98	58	2	50%	111	72
Ping	98	125	2	0%	158	-
Ping Result	98	102	2	0%	147.5	-
Begin Trans	90	204	10	29%	321.85	78
Rollback Trans	90	762	10	29%	1100.71	679
Commit Trans	90	432	10	29%	877.57	355.5
End Trans	90	321	10	29%	472.71	169.5

As can be seen, Table 1 is divided in two different blocks which represent those executions with and without errors. In this way it is easy to check whether a result is suitable for the achievements of the goals in the test phase or not. For a better understanding of the information captured in the table, a brief definition of each field is provided. The first column shows the name of the fifteen operations taken for the test phase. The “%” columns illustrate the percentage of executions that belong to the group of successful or unsuccessful executions. “Exect Time” is the abbreviation of “Execution Time” and depicts the average time taken to perform the operation in question. The “% Rec.” column shows the percentage of the unsuccessful group where a recovery process has been made after an error occurred. Finally, “R Time” is the average time taken in the recovery process.

Looking at Table 1, we can entail that the test performed reaches interesting values in terms of efficiency and time. The chosen operations show their behaviour and the differences of complexity among them by showing different execution times. It is also important to note that recovery times are very different depending on the operation performed, a fact that gives a new dimension of the operation’s complexity and which, in addition, shows how some operations are able to recover the system when an error occurs and others not.

In order to depict the distribution of errors in greater depth, Figure 3 shows error distribution among operations in terms of error appearance:

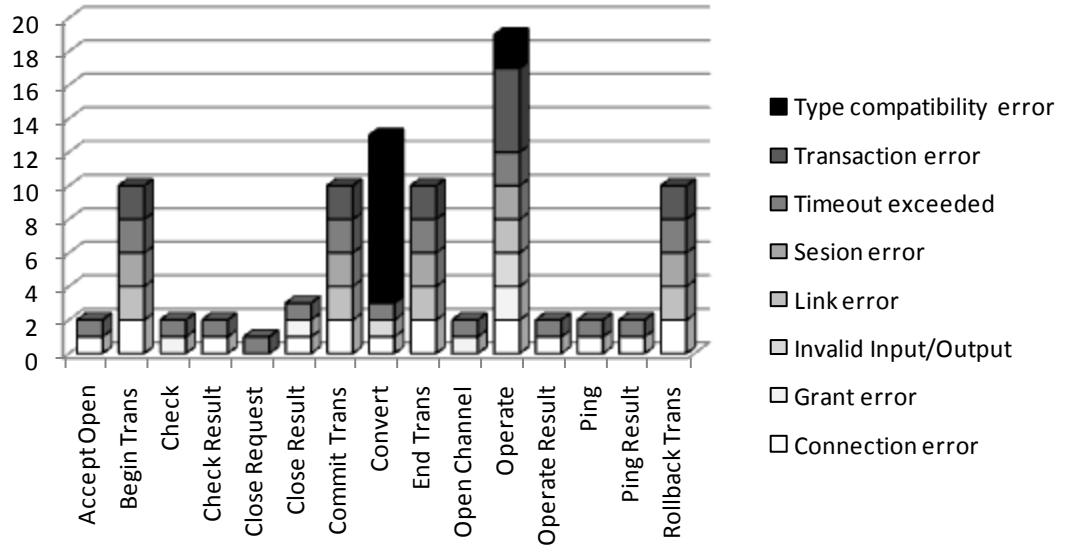

Figure 3. Error distribution among operations and errors

Results show that the higher amounts of errors appear in Operate, followed by Convert and all transaction related operations. A deeper view of the implications of errors shows that in 18 of 91 cases, almost 20%, CARL recovers from these errors and resumes operations in a proper way. Figure 4 shows the distribution of these errors.

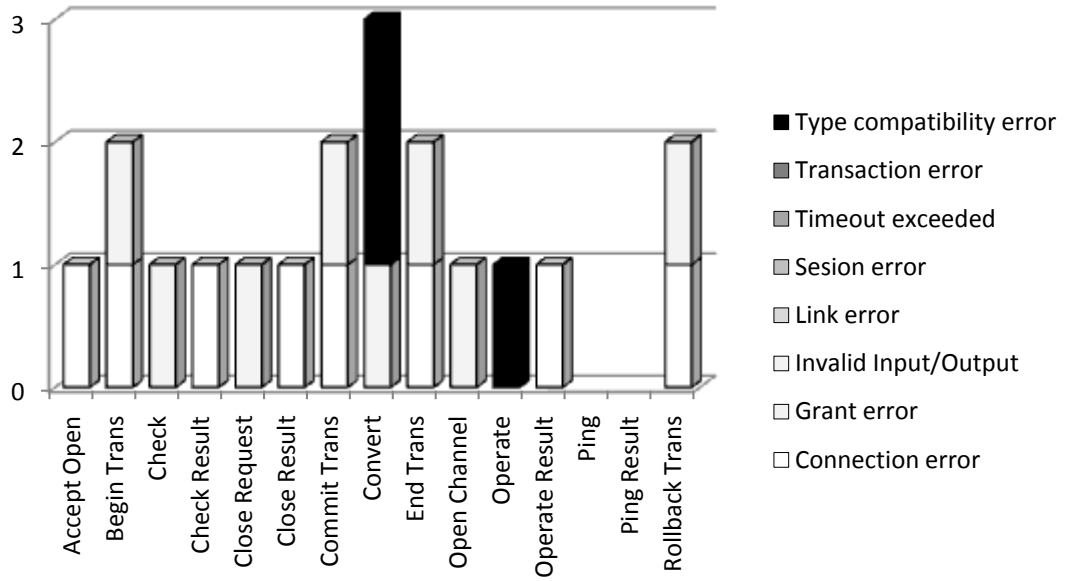

Figure 4. Recovery figures per error

Most error recoveries come from bad data types that CARL is able to correct in an intelligent way. Regarding performance, error recovery implies a loss of throughput, but not necessarily a dramatic loss of performance. Figure 5 depicts global execution times and recovery times for errors present in Figure 4.

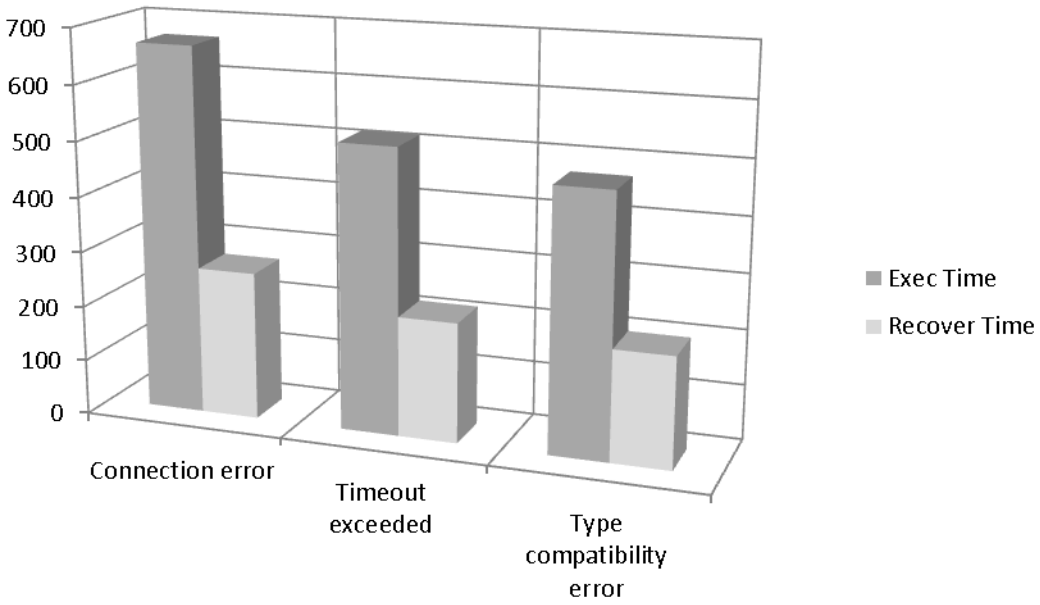

Figure 5. Execution and Recovery time in errors

Figure 5 shows that recovery times are more than acceptable, being around 40% of the total execution time in all cases (Connection error: 40.2%; Timeout exceeded: 39.1%; Type compatibility error: 41.1%). However these figures must be compared with correct executions of the operations recovered to contribute useful results. Figure 6 shows compared results of recovered executions and correct executions.

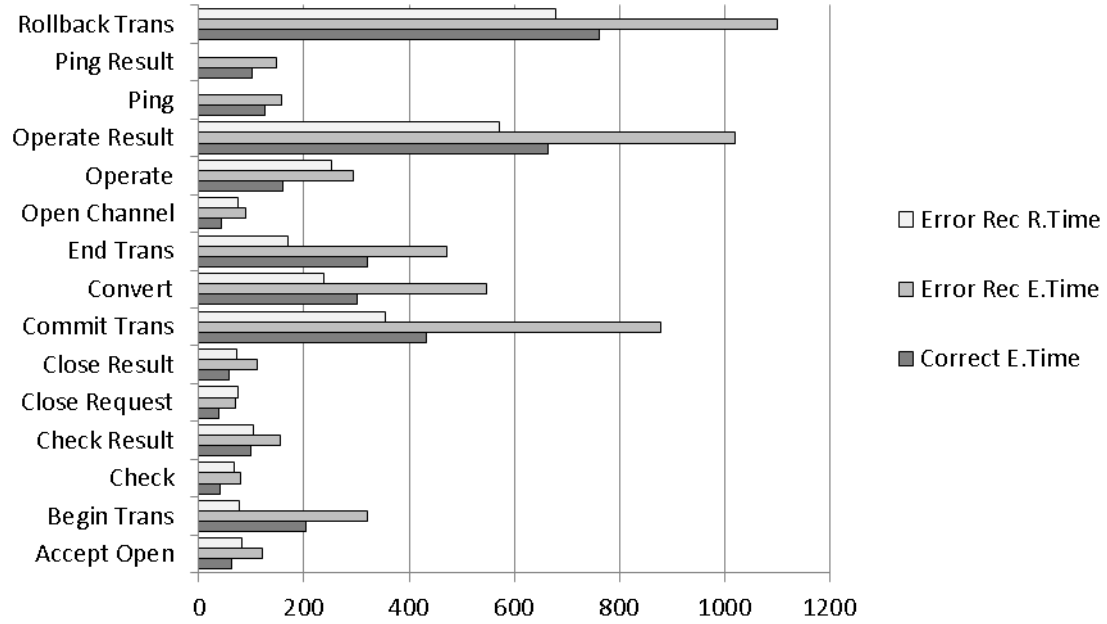

Figure 6. Average execution time of correct operations compared with recovery and execution time of recovered operations with error

On average, correct execution time represents 61.3% of the total error recovery execution time and error recovery time represents 50.5% of this total error recovery execution time. This implies that, on average, adding the correct execution time to error recovery time gives a figure that is over the total error recovery execution time. This means that there are improvements in the total recovery time that have their roots in time savings.

Regarding errors, the last test to be performed is the distribution of execution times with respect to operations. ANOVA analysis was used to analyze whether there were differences between operations, ,since the means comparison comprised more than two groups. Results indicated that groups presented statistically significant differences ($F(90)=31.393$, $p<.05$). The result of this test means that different operations denote different error handling times. It is also important to know if such differences appear in error recovery times with respect to operation type. The results show that these differences appear in this case ($F(18)=25.429$, $p<.05$).

6. Conclusions and future work

The research challenges of application interoperability and integration have been around for the last forty years of Computer Science research. Cloud-based Platform-as-a-Service domains offer a unique environment where a number of properties and constraints for interoperability can be set up front. Due to the very exclusive nature of these domains, a particular number of requirements such as transactionality and security in the applications should be respected, together with a formal representation of most of the features the applications could develop.

In this work, we have presented CARL, an interoperative and integration-oriented strategy to enable Complex Application, a concept which encompasses applications being deployed, managed and used in a PaaS environment in order to benefit from a very specific set of rules to enact cross-domain integration and tackling several challenges that could not occur in open spaces like the World Wide Web.

The future work challenges are tremendously open and varied. In the short term, on the one hand, we will focus on how applications behave and interact among themselves. On the other hand, we will research the benefits of adding new approaches of Semantic Technologies to CARL, such as Linked Data or Open Linked Data.

In the long term, our goals are much more ambitious. Since CARL will be evolving from a dynamic requirement real-world base scenario perspective, this canonical work will be the fundamental seed of upcoming evolutions of the language in our goal to answer and address part of those questions in a set of environments which can yield very different results from previous attempts in Enterprise Application Integration and B2B Integration.

References

ABADI, M. and THAU LOO, B. (2007): Towards a declarative language and system for secure networking. In *Proceedings of the 3rd USENIX international workshop on Networking meets databases*.

AHLMANN, A. and JAATUN, M. G. (2009): Privacy in a Semantic Cloud: What's Trust Got to Do with It? *In Proceedings of First International Conference, CloudCom 2009, Lecture Notes in Computer Science* **5931**(1): 107-118.

ALVARO, P., CONDIE, T., CONWAY, N., ELMELEEGY, K., HELLERSTEIN, J. and SEARS, R. (2010): Boom analytics: exploring data-centric, declarative programming for the cloud. *In Proceedings of the 5th European conference on Computer systems EuroSys'10*, 223-236.

ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., STOICA, I. and ZAHARIA, M. (2009): Above the Clouds: A Berkeley View of Cloud Computing, Technical report from UC Berkeley Reliable Adaptive Distributed Systems Laboratory.

AYMERICH, F. M., FENU, G. and SURCIS, S. (2008): An approach to a Cloud Computing network. *In Proceedings of First International Conference on the Applications of Digital Information and Web Technologies, ICADIWT 2008*, 113-118.

BLANCO, C., LASHERAS, J., FERNÁNDEZ-MEDINA, E., VALENCIA-GARCÍA, R., & TOVAL, A. (2011): Basis for an integrated security ontology according to a systematic review of existing proposals. *Computer Standards & Interfaces*, **33**(4): 372-388.

BUYYA, R., YEO, C. S. and VENUGOPAL, S. (2008): Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. *In Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications*, 5-13.

BUYYA, R., YEO, C. S., VENUGOPAL, S., BROBERG, J. and BRANDIC, I. (2008): Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* **25**(6): 599-616.

BUYYA, R., RANJAN, R. and CALHEIROS, R. N. (2009): Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. *In Proceedings of International Conference on High Performance Computing & Simulation, HPCS '09*, 1-11.

CAMPBELL-KELLY, M. (2009): Historical reflections – the rise, fall, and resurrection of software as a service. *Communications of the ACM*, **52**(5):28–30.

CLARKE, E.M. and GRUMBER, O. (1987): Avoiding the State Explosion problem in temporal logic model checking. *In Proceedings of Sixth Annual ACM Symposium on Principles of Distributed Computing*, **12**(2): 294-303.

COLOMO-PALACIOS, R., GARCÍA-CRESPO, Á., SOTO-ACOSTA, P., RUANO-MAYORAL, M., & JIMÉNEZ-LÓPEZ, D. (2010). A case analysis of semantic technologies for R&D intermediation information management. *International Journal of Information Management*, **30**(5): 465-469.

DEKEL, U., COHEN, T. and PORAT, S. (2003): Towards a Standard Family of Languages for Matching Patterns in Source Code. *In Proceedings of IEEE International Conference on Software-Science, Technology & Engineering*, 10-19.

FERNÁNDEZ-BREIS, J.T., CASTELLANOS-NIEVES, D., VALENCIA-GARCÍA, R. (2009) Measuring Individual Learning Performance in Group Work from a Knowledge Integration perspective. *Information Sciences*, **179**(4): 339–354.

FITZGERALD, W. M., FOLEY, S. N. and FOGHLÚ, M. (2009): Network Access Control Configuration Management using Semantic Web Techniques. *Journal of Research and Practice in Information Technology*, **41**(2), 99-118.

FOSTER, I., ZHAO, Y., RACIU, I. and LU, S. (2009): Cloud Computing and Grid Computing 360-Degree Compared. *In Proceedings of Grid Computing Environments Workshop*, GCE '08, 1-10.

GALINEC, D. (2010). Human Capital Management Process Based on Information Technology Models and Governance. *International Journal of Human Capital and Information Technology Professionals*, **1**(1): 44-60.

GARCÍA-CRESPO, A., COLOMO-PALACIOS, R., GÓMEZ-BERBÍS, J.M., & MENCKE, M. (2009). BMR: Benchmarking Metrics Recommender for Personnel issues in Software Development Projects. *International Journal of Computational Intelligence Systems*, **2**(3): 257-267.

GARCÍA-CRESPO, A., COLOMO-PALACIOS, R., GÓMEZ-BERBÍS, J.M., & RUIZ-MEZCUA, B. (2010). SEMO: a framework for customer social networks analysis based on semantics. *Journal of Information Technology*, **25**(2): 178-188.

GARCÍA-CRESPO, Á., GÓMEZ-BERBÍS, J.M., COLOMO-PALACIOS, R., & ALOR-HENÁNDEZ, G. (2011). SecurOntology: A semantic web access control framework. *Computer Standards & Interfaces*, **33**(1): 42-49.

GARCÍA-SÁNCHEZ, F., FERNÁNDEZ-BREIS, J.T., VALENCIA-GARCÍA, R., GÓMEZ, J.M., MARTÍNEZ-BÉJAR, R. (2008). Combining Semantic Web Technologies with Multi-Agent Systems for Integrated Access to Biological Resources. *Journal of Biomedical Informatics*, **41**(5): 848-859.

GARCIA-SANCHEZ, F., FERNANDEZ-BREIS, E., VALENCIA-GARCIA, R., JIMÉNEZ-DOMINGO, E., GOMEZ-BERBIS, J. M., TORRES-NIÑO, J. and MARTINEZ-MAQUEDA, D. (2010): Adding semantics to software-as-a-service and cloud computing. *WSEAS Transactions on Computers*, **9**(2): 154-163.

GARCÍA-SÁNCHEZ, F., VALENCIA-GARCÍA, R., MARTÍNEZ-BÉJAR, R., & FERNÁNDEZ-BREIS, J.T. (2009): An ontology, intelligent agent-based framework for the provision of semantic web services. *Expert Systems with Applications*, **36**(2):3167–3187.

HANUS, M. (2007): Multi-paradigm declarative languages. *Proceedings of ICLP'07 Proceedings of the 23rd international conference on Logic programming*, 45-75.

HAYES, B. (2008): Cloud Computing. *Communications of the ACM* **51**(7):9-11.

HOARE, A. (1978): Communicating sequential processes. *Communications of the ACM* **21**(8): 666-677.

JIANG, X. and AH-HWEETANA (2009): Learning and inferencing in user ontology for personalized semantic web search. *Information Sciences: an International Journal*, **179**(16):2794–2808.

KELLER, E. and REXFORD, J. (2010): The "Platform as a service" model for networking. *In Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 4-5.

KNORR, E. and GRUMAN G. What Cloud Computing really means. InfoWorld http://www.infoworld.com/ Accessed 19-July-2011.

LAKSHMANAN, L., SUBRAMANIAN, I. and SADRI, F. (2006): A Declarative Language for Querying and Restructuring the Web*. In Proceedings of the 6th International Workshop on Research Issues in Data Engineering (RIDE '96)*, 12-21.

LAWTON, G. (2008): Developing Software Online With Platform-as-a-Service Technology. *Computer*, **41**(6):13-15.

LENK, A., KLEMS, M., NIMIS, J., TAI, S. and SANDHOLM, T. (2009): What's inside the Cloud? An architectural map of the Cloud landscape. *In Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, 23-31.

McGUINNESS, D. L. and VAN HARMELEN, F. (2004): Owl web ontology language overview. W3c recommendation 10 feb 2004, World Wide Web Consortium. http://www.w3.org/TR/owl-features/. Accessed 5-April-2011.

McCARTHY, J (1963): A mathematical basis for computer programming. *Journal of Computer Programming and Formal Systems*, **1**(2): 30-37.

MOF: The Object Management Group (2002) Meta-object Facility, version 1.4.

PANETTO, H., BERIO, G., BENALI, K., BOUDJLIDA, N. and PETIT M. (2004): A Unified Enterprise Modeling Language for enhanced interoperability of Enterprise Models. *In Proceedings of the 11th IFAC INCOM2004 Symposium.*

PATIL, R., FIKES, R., PATEL-SCHENEIDER, P., McKAY, D., FININ, T., GRUBER, T. and NECHES, R. (1992): The DARPA Knowledge Sharing Effort Progress Report. *In Proceedings of the 3rd International Conference of Knowledge Representation and Reasoning*, 777-788.

PARTRIDGE, C. and STEFANOVA, M. (2001): A Synthesis of State of the Art Enterprise Ontologies. Lessons Learned. The BORO Program, LADSEB CNR.

PESIC, M. and VAN DER AALST, W. M. P. (2006): A Declarative Approach for Flexible Business Processes Management. *In Proceedings of BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Lecture Notes in Computer Science*, **4103**: 169-180.

RESCORLA, E. (2001): SSL and TLS, designing and building secure systems. Addison-Wesley.

RUIZ-MARTÍNEZ, J.M., VALENCIA-GARCÍA, R., FERNÁNDEZ-BREIS, J.T., GARCÍA-SÁNCHEZ, F. AND MARTÍNEZ-BÉJAR R. (2011) Ontology learning from biomedical natural language documents using UMLS. *Expert Systems with Applications*, **38**(10): 12365–12378

SALESFORCE http://www.salesforce.com Accessed 20-July-2011.

SCOTT, D.S. and STRACHEY, C. (1971): Towards a mathematical semantics for computer languages. Oxford University Computing Laboratory, Programming Research Group.

SHETH, A. and RAMAKRISHNAN, C. (2003): Semantic (Web) Technology in Action: Ontology Driven Information Systems for Search, Integration and Analysis. IEEE Data Engineering Bulletin **51**, 40-48.

SINGH, R., IYER, L. and SALAM, A. F. (2005): Semantic eBusiness. *International Journal on Semantic Web and Information Systems*, **1**(1):19-35.

STUDER, R., BENJAMINS, R. and FENSEL, D. (1998): Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, **25**(1-2):161–197.

VAQUERO, L. M., RODERO-MERINO, L., CACERES, J. and LINDNER, M. (2009): A break in the clouds: toward a cloud definition. ACM SIGCOMM Computer Communication Review **39**(1): 50-55.

VOROBIEV, A. and BEKMAMEDOVA, N. (2010): An Ontology-Driven Approach Applied to Information Security. *Journal of Research and Practice in Information Technology*, **42**(1): 61-76.

YOUSEFF, L., BUTRICO, M. and DA SILVA, D. (2008): Toward a Unified Ontology of Cloud Computing. *In Proceedings of Grid Computing Environments Workshop*, GCE '08, 1-10.

ZENG, H. T., KANG, B. Y. and KIM H. G. (2008): An ontology-based approach to learnable focused crawling. *Information Sciences* **178**(23):4512–4522.