# Discoverying Cloud Services by Preconditions and Effects for Compositions

Lorena Erdens[1], Daniela Barreiro Claro[1], Denivaldo Lopes[2] and Patrick Albers[3]
[1]*FORMAS/LASID/DCC/IM, Federal University of Bahia (UFBA), Av. Adhemar de Barros, s/n, Salvador, Bahia, Brazil*
[2]*FORMAS/LESERC, Federal University of Maranhao (UFMA), Campus do Bacanga-CCET, Sao Luis, Maranhao, Brazil*
[3]*ESEO, 4 Rue Merlet de la Boulaye, BP 30926, 49000, Angers, France*

Keywords: Cloud Services, Discovery, Semantic Web, Preconditions and Effects, Information Systems.

Abstract: Nowadays, most of companies deploy their services as a Web service format over the Internet. Additionally, cloud environments are dealing with web services published on the Internet. The use of the services deployed is subject to its discovery. Many techniques are being developed to increase the description of a web service, thus automating their discovery and providing more significant results. Such techniques are based on semantic concepts to provide more unambiguous information on describing such services. This paper proposes to discover cloud services based on the semantic Web, especially on preconditions and effects. Several works have been carried out on preconditions and effects, however we have achieved more accurate results using service descriptions based on conditions. We present our approach examining three main aspects: (i) relevant retrieved services (ii) complexity and (iii) execution time and we compare with other closely related work in an attempt to position our work and our results.

## 1 INTRODUCTION

A great number of business processes in information systems have been developed in a web service format because of their standard and loosely coupled features. Most of these web services can be published over a distributed environment, such as the Internet, an enterprise (as a local network) or into a cloud.

The proliferation of such services is enabling the creation of various types of web services. From a global view, we can classify two types of services: (a) a client-side service, developed in JavaScript, using technologies such as AJAX and HTML 5 and handled by customers in their respective browsers. We call these services a light-weight service (b) a server-side service, where connections are made between web servers and their development environments. This kind of service (b) can usually be used through a proxy and allows interoperability between heterogeneous applications and can be considered as a heavy-weight service. Both types of services are loosely coupled and are immerged in a distributed environment, such as a cloud (Zhang et al., 2010).

Some of the services on the Internet are incorporated into cloud environments as a SaaS (Software as a Service) (Höfer and Karagiannis, 2011). Sometimes, a single SaaS does not achieve a user request, thus it has to be composed. These services are combined on-the-fly inside their clouds as a CaaS (Composition as a Service)(Höfer and Karagiannis, 2011). Nowadays, GoogleAppEngine (Höfer and Karagiannis, 2011) is a platform (PaaS) that can be used for free of charge. Some other platforms need a valid credit card number, even if it is not charged. This was our main motivation for using this platform within an academic research project. However, GoogleAppEngine only supports heavy-weight services, that is a server-side approaches. Thus in this work, we carried out only heavy-weight services. As our SaaS is being treated as heavy-weight services, our experiments were carried out over a server-side service approach.

With the continuing increase in service availability, a major challenge has been the automated discovery for new services closer to user/machine requirements. Among other features that can be incorporated into an automation process, the addition of a semantic web is widely used. A semantic description of web services can minimize the ambiguities and consequently facilitate such automated discovery (Berners-Lee et al., 2001).

One way to incorporate semantics into web ser-


Erdens L., Barreiro Claro D., Lopes D. and Albers P..
Discoverying Cloud Services by Preconditions and Effects for Compositions.
DOI: 10.5220/0004006302640270
In *Proceedings of the 14th International Conference on Enterprise Information Systems* (ICEIS-2012), pages 264-270
ISBN: 978-989-8565-11-2

vices is through the use of semantic description languages, such as SAWSDL (Kopecky et al., 2007), OWL-S (Burstein et al., 2004) and WSMO(Roman et al., 2005). Among these languages, OWL-S can be coupled to an OWL(Mcguinness and van Harmelen, 2004) domain ontology, which makes inferences through its classes, properties and axioms, thus reducing the gap existing between a machine/user request and the offered service.

As we are dealing with composite services (CaaS), it is important to analyze the semantic links between each service in a composition. In the literature, several approaches have focused on service discovery, especially on semantic links. Some approaches deal with inputs and outputs (Paolucci et al., 002b; Amorim et al., 2011; Lécué, 2011). Others have demonstrated that it is necessary to perform an additional analysis on conditions in order to retrieve more useful services (Bener et al., 2009; Bellur and Vadodaria, 2008). In addition to these semantic-based approaches, several authors have incorporated a syntactic analysis complement of the semantic analysis, such as (Amorim et al., 2011; Paolucci et al., 2003; Klusch et al., 2006; Junior et al., 2009). Our previous work proposed a hybrid approach, combining a semantic analysis based on inputs and outputs and a syntactic analysis based on the class structure. This work aims to extend our previous approach (Amorim et al., 2011) to incorporate preconditions and effects on semantic link analysis.

This paper is structured as follows: section 2 presents our proposed approach; section 3 presents our experiments and results; section 4 compares our work with related ones; section 5 shows our tool; and, finally, section 6 discusses our conclusions and future directions.

## 2 PROPOSED APPROACH

The basic idea behind our approach is to match the set of IOPE (input, output, preconditions and effects) of a request (user/machine) to the IOPE of a service. In this work, we have concentrated our efforts in OWL-S profile to describe services and requests because of its relationship with a domain ontology and its ability to be classified into a heavy-weight service, i.e. a cloud service.

### 2.1 Problem Definition

Given a set of finite services $S = \{s_1, ..., s_n\}$ and a machine/user request, the semantic matching between the request $r$ and a service $s_i \in S$ can be defined based on semantic similarities between its inputs, outputs, preconditions and effects. The semantic similarity is analyzed based on a matching function $match(C_a, C_b) = d$ within two concepts $(C_a, C_b)$ inside the same ontology or knowledge representation. The matching result is given by a similarity degree $d$, where $d \in D$ and $D = \{Exact, Plugin, Subsumes, Sibling, Fail\}$. Ideas of this similarity degree were kept track of (Paolucci et al., 002b; Samper et al., 2008).

Each service $s_i$, where $i = [1..n]$ has a set of inputs $(I_i)$, outputs $(O_i)$, preconditions $(P_i)$ and effects $(E_i)$. Each set of inputs $(I_i)$ has atomic inputs $y_j$, where $y_j \in I_i$. Each atomic input $y_j$ is matched through the function $match(C_a, C_b) = d$ as a concept $C_a$ inside an OWL ontology. The concept $C_b$ corresponds to the user/machine request's input value. For the output set the process is similar.

Concerning the preconditions and effects, some minor differences need to be analyzed. In SWRL (Horrocks et al., 2004), each precondition $p_k$, where $p_k \in P_i$ of a service $s_i$ is composed of a *predicate* and *attributes*. SWRL rules extend OWL axioms to include some Horn rules, with an antecedent and a consequent. For instance, the SWRL rule *hasCard(User, Account)* means that if *User* and *Account* are true, then *hasCard* is true. In SWRL, the predicate is the consequent of a rule and the attributes are the antecedents, i.e. *hasCard* is a predicate in SWRL and *User* and *Account* are attributes in SWRL.

Our approach analyzes separately each predicate and attributes. Thus, our function $match(C_a, C_b) = d$ firstly performs the comparison between a service predicate $(p_j[predicate])$ and a request predicate $(q[predicate])$ as $C_a$ and $C_b$ respectively. Thus, we can have a function similarity of predicates as follows:

$$simPredicate(p_j, q) = match(p_j[predicate], q[predicate])$$

As regards attribute matching, for each precondition $(p_j)$ two attributes are analyzed. The semantic similarity between these attributes is the maximum degree of matching between the service attribute and request attribute. For instance, if an attribute has a *Subsumes* degree ($d = Subsumes$) and another semantic similarity analyzes another attribute and it returns $d = Exact$, thus the result attribute value is those with an *Exact* degree of matching. Our semantic similarity function of the attributes can be described as follows:

$$simAttribute(p_j, q) = \max(p_j[attribute] \times q[attribute])$$

The function similarity precondition is given by the minimum degree of similarity between these func-

tions. For instance, if the function *simPredicate* returns $d = Subsumes$ and the function *simAttribute* returns $d = Plugin$ and $d = Sibling$ respectively, thus the similarity function returns the minimum degree $d = Plugin$. With this mechanism, we can ensure that a minimum similarity degree between our semantic approach occurs.

$$simPrec(p_j, q) = \min(simPredicate(p_j, q) \times simAttribute(p_j, q))$$

Finally, the overall semantic similarity function between a service and a request is given by the minimum degree given by the above mentioned function as follows: The function similarity analyzes the minimum degree between a service ($s_i$) and a request ($q$) as follows:

$$similarity(s_i, q) = \min(simIn, simOut \times simPrec \times simEff)$$

In order to illustrate, suppose a service precondition $P_s$ and a request precondition $R_s$ within the following structure $predicate(attribute, attribute)$.

**Service:** $P_s(A1_s, A2_s)$, where $P_s$ is the predicate and $A1_s$ and $A2_s$ are two attributes of a service.

**Request:** $P_r(A1_r, A2_r)$, where $P_r$ is the predicate and $A1_r$ and $A2_r$ are two attributes of a request

Each function *simPredicate* and *simAttribute* is executed separately. There is no comparison between predicates and attributes, only within a service predicate and request predicate. The *simPrec* function is executed and returns a set of 3-tuple with a term of a service, a request and the degree of match (Gi), as described below:
$\{(P_r, P_s, G1), (A1_s, A1_r, G2), (A1_s, A2_r, G3), (A2_s, A1_r, G4), (A2_s, A2_r, G5)\}$

Suppose $G2 > G3$ and $G4 > G5$, this means $G2$'s similarity degree is higher than $G3$, i.e. $G2$ is *Exact* and $G3$ is *Subsumes*. Thus the precondition function returns $(A1_s, A1_r, G2)$ and $(A2_s, A1_r, G4)$, because of their major degree.

It is important to understand that an attribute could be used twice if it has a higher degree of similarity than other attributes.

$\forall P_i, \exists (P_s^i, P_r^i, G_i) | G_i \neq Fail$, where $P_r^i$ and $P_s^i$ are the preconditions of a request and a service respectively.

Thus, we can ensure that for the execution of a service, it is important that at least one degree of similarity be different from *Fail*.

# 3 EXPERIMENTS AND RESULTS

Despite the rapid growth of semantic services, there has been little work in experimental tests and comparison among discovery methods. A public collection of currently available tests is called **OWL-S Test Collection 2**, but it does not include pre-conditions nor effects. Authors in (Bener et al., 2009) modified this collection by adding preconditions and effects described in SWRL (Horrocks et al., 2004). Despite the fact that they incorporate SWRL preconditions and effects into this dataset, the version used is not compatible with OWL-S API 1.2, thus no preconditions or effects can be actually retrieved. Thus, we enhanced this test set adapting it to the latest version of OWL-S 1.2 files that can be read automatically by an algorithm. This means that a machine can automatically interpret each condition described.

## 3.1 Relevant Retrieved Services

In order to compare our approach with the algorithm proposed by (Bener et al., 2009) we developed two case studies:

**First:** Reuse of preconditions and effects.

**Second:** Predicate of service preconditions and the predicate of request preconditions are not exactly equal.

In the first test, suppose we have a service and a request with the following preconditions:

**Service:** *buyAutomobile(User, Model)* and *buyCar(User, Model)*

**Request:** *buyAutomobile(User, Model)* and *buyFood(User, Type)*

Suppose as well an ontology with *buyCar* and *buyBus* as descendant of *buyAutomobile*.

The algorithm proposed by (Bener et al., 2009) performs the matching between the service precondition *buyAutomobile(User, Model)* and the given request. However, if it tries to find a request precondition that matches the service precondition *buyCar(User, Model)*, such an algorithm returns *Fail* because the request precondition *buyFood(User, Type)* is not similar to *buyCar(User, Model)*, unlike our approach that re-uses preconditions. Thus, our proposed algorithm matches the service precondition *buyAutomobile(User, Model)* with a similar request precondition *buyCar(User, Model)*. Additionally, trying to

match a similar precondition *buyCar(User, Model)* our approach retrieves the same precondition once again *buyAutomobile(User, Model)*, as we have similar preconditions *buyCar* and *buyAutomobile* as presented in the ontology. In this case our approach is able to retrieve a similar service precondition where the algorithm proposed by (Bener et al., 2009) does not retrieve the service; their approach fails.

The second experiment carries on the service precondition ***buyCar****(User, Model)* and the request precondition ***buyBus****(User, Model)*. Suppose the existence of the predicate *buyAutomobile*. The algorithm proposed by (Bener et al., 2009) defines the similarity degree between such preconditions as *Fail* because it does not analyze the semantic similarity between predicates through an ontology. Thus, they conclude that ***buyCar****(User, Model)* and ***buyBus****(User, Model)* are different, because these preconditions are not the same, thus the algorithm discards this useful service. In our approach, we analyze each predicate and attributes, thus these predicates match as *Sibling* and can return the potential service.

The behavior of our approach is advantageous because it retrieves more useful services, avoiding the wrong elimination of services. In some cases, it is better to retrieve more and leave the decision to a final user, instead of wrongly eliminating a potentially interesting service. For instance, in criminal investigations suspicious, it is better to retrieve more suspects, instead of wrongly discarding a criminal.

## 3.2 Complexity Analysis

We analyzed the complexity of our algorithm within semantic filters on preconditions and effects. We considered that for each service precondition, our algorithm compares each request precondition, and for each request effect, our algorithm compares each service effect.

Based on (Cormen et al., 2001), we can conclude that, in the worst case, the complexity involved in our algorithm is $O(n^2)$. Our algorithm analyzes the effects that work just like the pre-conditions, then its complexity is also $O(n^2)$. On the other hand, our algorithm analyzes inputs and outputs and has complexity $O(n^2)$ as stated by (Amorim et al., 2011).

The flow of the algorithm has the following order: Input analysis $(O(n^2))$ $\rightarrow$ Output analysis $(O(n^2))$ $\rightarrow$ Preconditions analysis $(O(n^2))$ $\rightarrow$ Effects analysis $(O(n^2))$.

Thus, the total complexity of our algorithm is: $O(n^2)+O(n^2)+O(n^2)+O(n^2) = 4(O(n^2))$

## 3.3 Execution Time

In order to evaluate the execution time of our algorithm, we carried out an experiment within 20 requests and a repository with 100 services adapted from the Test Collection 2 (Bener et al., 2009). All 20 requests were created within the same input and output sets so as to ensure that preconditions and effects were the only variable factors between requests. Thus, we can concentrate in a single factor to analyze, i.e. runtime.

We performed two experiments. The first experiment was to analyze inputs and outputs, calculating the average execution time. The second experiment dealt with inputs, outputs as well as preconditions and effects. With these two measures, it was possible to calculate the overhead between our new approach (with IOPE) and other works with only (IO).

Table 1: Comparative of runtime analysis between an IO algorithm and our IOPE algorithm only for requests $r3$, $r4$, $r7$ and $r8$.

| Request | IO(s) | IOPE(s) | IO/IOPE | Precond | Effects | Total P/E |
|---|---|---|---|---|---|---|
| r3 | 2,38 | 40,07 | 16,86 | 1 | 0 | 1 |
| r4 | 2,49 | 40,73 | 16,39 | 1 | 0 | 1 |
| r7 | 1,84 | 14,61 | 7,95 | 0 | 1 | 1 |
| r8 | 2,40 | 19,36 | 8,08 | 0 | 1 | 1 |

Considering the requests $r3$, $r4$, $r7$ and $r8$ specifically on Table 1, we can observe that the relation between the execution time (in ms) values of IO and IOPE taking the requests $r3$ and $r4$ (i.e. $16,86$ and $16,39$ respectively) is considerably higher than the relation of execution time values between the requests $r7$ and $r8$ (i.e. $7,95$ and $8,08$ respectively).

Despite the fact that all four requests contain the same total number of preconditions and effects, the difference between the requests $r3$ and $r4$ is that they have preconditions but no effects. On the other hand, we can observe that requests $r7$ and $r8$ have an effect but no precondition. The difference between the computational time of $r3$, $r4$, $r7$ and $r8$ can be justified by the algorithm flow analysis of preconditions and effects. Suppose a service $s1$ has a precondition and an effect. To make the comparison between $s1$ and $r3$, the algorithm first examines whether $r3$ has preconditions. If there is a positive result, an analysis is performed on the preconditions between $s1$ and $r3$. Then the algorithm verifies that $r3$ has effects, obtaining a negative result. Thus, the algorithm stops its execution. The same applies to the request $r4$. Now suppose the comparison between $r7$ and $s1$. First the algorithm checks that $r7$ has preconditions. If the result is negative, the algorithm stops its execution immediately. The same happens with the request $r8$. Thus,

we can conclude that for requests $r3$ and $r4$, the algorithm performed the comparison on preconditions, but fails to compare the effects. In the case of requests $r7$ and $r8$, the algorithm does not compare the preconditions nor the effects. This explains why the runtime for requests that have pre-conditions is greater than the runtime for requests that do not have preconditions.

Figure 1 depicts the behavior of the runtime variation by the number of preconditions and effects.

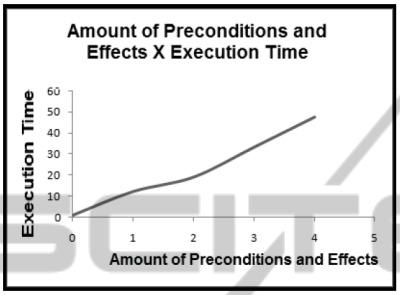


Figure 1: Number of preconditions and Effects X Runtime.

Concerning our execution time, as we are using preconditions and effects from an OWL ontology, our algorithm does not have good results to scale up when there is a large number of preconditions and effects inside a service. We also analyzed empirically that in almost all tests sets experiments found in the literature, authors usually dealt with two preconditions. Within two preconditions and effects we can consider that our algorithm is working pretty well as other related approaches. However, we are working on an improvement to our algorithm to scale up better in the presence of more than two preconditions and effects.

# 4 ANALYSIS OF RELATED WORK

The wide use of service has motivated some researchers to develop discovery algorithms. This section presents some related work and makes a comparative analysis of these works and our approach.

The use of semantic filters were first introduced by (Paolucci et al., 002b) with input and output parameters. This is a flexible approach based on the similarity degree, taking into account the distance between two nodes in a tree (ontology). This work proposed four degrees called *exact*, *plugin*, *subsumes* and *fail*.

Authors in (Amorim et al., 2011) have studied IO (input and output) on semantic filters. However, they proved that only semantic features were not sufficient to achieve good accuracy; thus they proposed an hybrid approach in two steps: (i) functional semantic based on semantic filters (Paolucci et al., 002b) over their inputs and outputs parameters and (ii)structural semantic that analyzes each neighbor concept inside an ontology structure. The second step is only carried out in case of failure on the first step due to performance analysis. The major problem here is tackling only inputs and outputs parameters.

As regards some closely related work, SAM+ (Bener et al., 2009) is an algorithm that deals with preconditions and effects by using Paolucci filters (Paolucci et al., 002b; Paolucci et al., 002a). Firstly, their algorithm analyzes inputs and outputs and then their preconditions and effects described in SWRL (Horrocks et al., 2004). Their proposed approach uses three methods: **subsumption** that uses semantic filters and gives an intermediate weight; **semantic distance** that calculates the semantic distance between two concepts inside an ontology and the third method, **wordnet**, that uses the wordnet as a database of synonyms. Thus, a final weight is given for the overall matching.

Work proposed by (Bellur and Vadodaria, 2008) uses semantic filters (Paolucci et al., 002b) to analyze the compatibility of web service parameters. They also divide their approach into three steps: **parameter compatibility** that uses the semantic filters within IOPE parameters; **condition equivalence** that proposes a structural analysis between two conditions; **condition evaluation** that evaluate if such conditions really satisfy each other.

Table 2 summarizes and compares previously related work with our proposed approach.

Table 2: A summary among semantic discovery algorithms and their features.

| Feature | Paolluci | Amorim | Bener | Bellur | Our |
|---|---|---|---|---|---|
| Use of OWL-S 1.1 | X | X | X | X | |
| Use of OWL-S 1.2 | | | | | X |
| Analysis of preconditions and effects | | | X | X | X |
| Independency of the quantity of parameter | | | | | X |
| Re-use of preconditions and effects | | | | | X |
| Analysis of semantic predicate | | | | | X |

The first column is the feature analyzed and the following columns depict the presence (mark as X) and absence (empty column) of each feature within each approach: (Paolucci et al., 002b; Amorim et al., 2011; Bener et al., 2009; Bellur and Vadodaria, 2008) and our approach.

# 5 OUR TOOL IMPLEMENTATION

In order to spread and socialize our work, we developed a tool OWL-S Composer[1]. This tool was developed in Java language, with some additional features such as Jena 2.6.3 (JENA, 2000), pellet 2.2.2 (ClarkParsia, 2010) and OWL-S API 3.1-SNAPSHOT (OsirisNext, 2010), which was the lastest available version of OWL-S API. These additional features were necessary to read and analyze OWL and OWL-S description of services.

In this project, Jena API was used for retrieving OWL elements. Jena API also includes the Pellet inference engine that was used to make semantic inferences concerning the axioms inside an ontology.

The developed system also makes use of OWL-S API 3.1-SNAPSHOT. This API transforms the elements of OWL-S document into Java objects and was used to obtain the set of inputs, outputs, preconditions and effects. This version of that API deals with OWL-S 1.2, its latest version.

# 6 CONCLUSIONS AND FUTURE WORK

The cloud environment is facing some new challenges for discovering SaaS (Software as a Service). In this work, we can retrieve more relevant services using preconditions and effects, thus minimizing the recovery of useless services and consequently the time configuring and finding services in a cloud. We also analyzed the execution time with preconditions and effects to ensure that our approach is feasible.

We have incorporated this solution into a OWL-S Composer plugin so as to discover in Google cloud environments.

As future work, we are working on the scalability of our solution and other PaaS, such as Amazon, SalesForce.

# ACKNOWLEDGEMENTS

Some of the authors would like to acknowledge the Brazilian Government by CNPq (Grant 560231/2010-5).

[1]http://homes.dcc.ufba.br/~dclaro/tools.html#owls3

# REFERENCES

Amorim, R., Claro, D. B., Lopes, D., Albers, P., and Andrade, A. (2011). Improving web service discovery by a functional and structural approach. In *IEEE ICWS 2011 - The 9th International Conference of Web Services*, pages 411–418.

Bellur, U. and Vadodaria, H. (2008). On extending semantic matchmaking to include preconditions and effects. In *IEEE International Conference on Web Services*, ICWS '08, pages 120–128.

Bener, A. B., Ozadali, V., and Ilhan, E. S. (2009). Semantic matchmaker with precondition and effect matching using swrl. *An International Journal: Expert Systems with Applications*, pages 9371–9377.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.

Burstein, M., Hobbs, J., Lassila, O., Mcdermott, D., Mcilraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. (2004). OWL-S: Semantic Markup for Web Services. Website.

ClarkParsia (2010). Pellet 2.2.2 release. http://clarkparsia.com/pellet.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001). *Introduction to algorithms*. MIT Press.

Höfer, C. N. and Karagiannis, G. (2011). Cloud computing services: taxonomy and comparison. *Journal of Internet Services and Applications*, 2(2):81–94.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). Swrl: A semantic web rule language combining owl and ruleml. Technical report.

JENA (2000). Jena - a semantic web framework for java. http://jena.sourceforge.net/.

Junior, J. G. S., Lopes, D., Claro, D. B., and Abdelouahab, Z. (2009). A step forward in semi-automatic metamodel matching: Algorithms and tool. In *International Conference on Enterprise Information Systems (ICEIS 2009), LNBIP*, volume 24, pages 137–148.

Klusch, M., Fries, B., and Sycara, K. P. (2006). Automated semantic web service discovery with owls-mx. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, NY, USA. ACM New York.

Kopecky, J., Vitvar, T., Bournez, C., and Farrell, J. (2007). SAWSDL: Semantic Annotations for WSDL and XML Schema. *Internet Computing, IEEE*, 11(6):60–67.

Lécué, F. (2011). Inferring data flow in semantic web service composition. In *IEEE ICWS 2011 - The 9th International Conference of Web Services*, pages 347–354.

Mcguinness, D. L. and van Harmelen, F. (2004). OWL web ontology language overview. W3C recommendation, W3C.

OsirisNext (2010). Owl-s api 3.1-snapshot. http://on.cs.unibas.ch/owls-api/.

Paolucci, M., Kawamura, T., and Blasio, J. (2003). A preliminary report of a public experiment of a semantic

service matchmaker combined with a uddi business registry. In *1st International Conference on Service Oriented Computing (ICSOC 2003)*, Trento, Italy.

Paolucci, M., Kawamura, T., Payne, T., and Sycara, K. (2002a). Semantic matching of web services capabilities. *Lecture Notes in Computer Science*.

Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K. (2002b). Semantic matching of web services capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC2002)/LNCS*, volume 1, page 333347.

Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., and Fensel, D. (2005). Web service modeling ontology (wsmo). *Applied Ontology*, 1:77–106.

Samper, J. J., Adell, F. J., van den Berg, L., and Martínez, J. J. (2008). Improving semantic web service discovery. *Journal of networks*, 3(1):35–42.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.