An Hybrid, QoS-Aware Discovery of Semantic Web Services Using Constraint Programming[*]

José María García, David Ruiz, Antonio Ruiz-Cortés, Octavio Martín-Díaz, and Manuel Resinas

Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática
Av. Reina Mercedes s/n, 41012 Sevilla, España
josemgarcia@us.es

Abstract. Most Semantic Web Services discovery approaches are not well suited when using complex relational, arithmetic and logical expressions, because they are usually based on Description Logics. Moreover, these kind of expressions usually appear when discovery is performed including Quality-of-Service conditions. In this work, we present an hybrid discovery process for Semantic Web Services that takes care of QoS conditions. Our approach splits discovery into stages, using different engines in each one, depending on its search nature. This architecture is extensible and loosely coupled, allowing the addition of discovery engines at will. In order to perform QoS-aware discovery, we propose a stage that uses Constraint Programming, that allows to use complex QoS conditions within discovery queries. Furthermore, it is possible to obtain the optimal offer that fulfills a given demand using this approach.

Keywords: Discovery Mechanisms, Quality-of-Service, Semantic Matching, Constraint Programming.

1 Introduction

Most approaches on automatic discovery of Semantic Web Services (SWS) use Description Logics (DLs) reasoners to perform the matching [7,13,15,18,26,27]. These approaches have limitations regarding with the expressiveness of searches, especially when there are Quality-of-Service (QoS) conditions integrated within queries. For instance, a condition like "find a service which $availability \geq 0.9$, where $availability = MTTF/(MTTF + MTTR)$"[1] can not be expressed in DLs. Although there are proposals that extend traditional DLs with concrete domains in many ways [9], they still have limitations on expressing complex conditions [1,14], as in the previous example. These complex conditions usually

[*] This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web-Factories (TIN2006-00472).

[1] $MTTF$ stands for "Mean Time To Failure", while $MTTR$ stands for "Mean Time To Repair". Both of them are QoS parameters often used to define service availability.

appear when performing a QoS-aware discovery, so in this case DLs reasoning is not the most suitable choice.

QoS conditions are contemplated in several SWS discovery proposals. For instance, Wang *et al.* extend WSMO framework to include QoS parameters that allow to discover the best offer that fulfills the demanded conditions [30]. Benbernou and Hacid propose the use of constraints, including QoS-related ones, to discover SWS [3]. Moreover, Ruiz-Cortés *et al.* model the QoS conditions as Constraint Satisfaction Problems (CSPs) [23], but in the context of non-semantic Web Services.

Our proposal is an hybrid architecture to discover SWS. Discovery may be split into different stages, each of them using the best suited engine depending on the features of the corresponding stage. We identify at least two stages in this process: QoS-based discovery and functional (non-QoS) discovery. The former may be done using Constraint Programming (CP), as proposed in the case of non-semantic Web Services in [23], while the latter is usually performed by DLs reasoners, although it is not restricted to use other techniques.

Our approach allows to filter offers, stage by stage, using a proper search engine until the optimal offer that fulfills a demand is found. This optimization is accomplished due to the proposed use of CP in the QoS-aware discovery stage, also enabling the definition of more complex conditions than defined ones using DLs. Furthermore, our proposed architecture is loosely coupled and extensible, allowing the addition of extra discovery engines if necessary.

The rest of the paper is structured as follows. In Sec. 2 we introduce related works on discovering SWS, discussing their suitability to perform a QoS-aware discovery. Next, in Sec. 3 we present our hybrid discovery proposal, explaining the proposed architecture and how CP can be used in a QoS-aware semantic discovery context. Finally, in Sec. 4 we sum up our contributions, and discuss our conclusions and future work.

2 Discovering Semantic Web Services

In this Section, we discuss related work on discovering SWS, describing the different approaches and analyzing their suitability to handle QoS parameters and conditions, in order to perform a QoS-aware discovery.

2.1 Preliminaries

Each proposal uses its own terminology to refer to the entities involved in the discovery process, especially its descriptions of the requested and provided services. For the sake of simplicity, we use one single notation along this paper.

We refer to a demand (denoted by the Greek letter delta, i.e. $\delta emand$) as a set of objectives that clients want to accomplish by using a service that fulfills them. It may be composed of functionality requirements and QoS conditions that the requested service must fulfill, such as "find a service which $availability \geq 0.9$, where $availability = MTTF/\left(MTTF + MTTR\right)$". The different proposals refer to demands as goals [22], queries [2,3,13], service request [7,19,27] or user requirements [30].

An offer (denoted by the Greek letter omega, i.e. ω*ffer*) of a service is the definition of a SWS that is publicly available from a service provider. An offer may be composed of functionality descriptions, orchestration, choreography, and QoS conditions of the given service. For instance, an offer can consist in a QoS condition like "$MTTF$ is from 100 to 120 inclusive and $MTTR$ is from 3 to 10 also inclusive". Different approaches refer to offers as advertisements [2,13,30], service capabilities [19,22,27], or service profiles [7,15,16].

Most proposals on discovering SWS are built upon one of the following description frameworks. Firstly, OWL-S [16] is a DARPA Agent Markup Language program initiative that defines a SWS in terms of an upper ontology that contains concepts to model each service profile, its operations and its process model. It is based on OWL standard to define ontologies, so it benefits from the wide range of tools available. Secondly, the Web Service Modeling Ontology (WSMO) [22] is an European initiative whose goal, as OWL-S, is to develop a standard description of SWS. Its starting point is the Web Service Modeling Framework [6], which has been refined and extended, developing a formal ontology to describe SWS in terms of four core concepts: ontologies, services, goals and mediators. Finally, the METEOR-S project from the University of Georgia takes a completely different, but aligned approach than the others. Its main target is to extend current standards in Web Services adding semantic concepts [25], among others contributions discussed here. These extensions make use of third party frameworks, including the previous two, to semantically annotate Web Service descriptions. These proposals have extensions to take care of QoS parameters.

2.2 Related Work

In the context of DAML-S (the OWL-S precursor), Sycara *et al.* show how semantic information allows automatic discovery, invocation and composition of Web Services [27]. They provide an early integration of semantic information in a UDDI registry, and propose a matchmaking architecture. It is based on a previous work by Paolucci *et al.*, where they define the matching engine used [19]. This engine matches a demand and an offer when this offer describes a service which is "sufficiently similar" to the demanded service, i.e. the offered service provides the functionality demanded in some degree. The problem is how to define that degree of similarity, and the concrete algorithm to match both service descriptions. They update their work to OWL-S in [28].

Furthermore, there are proposals that perform the matchmaking of SWS using DLs [7,13,15]. Particularly, González-Castillo *et al.* provide an actual matchmaking algorithm using the subsumption operator between DLs concepts describing demands and offers [7]. They use existing DLs reasoners, as RACER [8] and FaCT [11], to perform the matchmaking. On the other hand, Lutz and Sattler [15] do not provide an algorithm, but give the foundations to implement it using subsumption, like Li and Horrocks [13], who also give hints to implement a prototype using RACER.

These three works define different matching degrees as in [27], from exactly equivalents to disjoint. All of them perform this matching by comparing inputs and outputs. Apart from that, neither of them can obtain the optimal offer using QoS parameters. However, Benatallah *et al.* propose to use the degree of matching to select the best offer in [2], but it results to be a NP-hard problem, as in any optimization problem [4].

On the other hand, Benbernou and Hacid realise that some kinds of constraints are necessary to discover SWS, including QoS related ones, so they formally discuss the convenience of incorporating constraints in SWS discovery [3]. However, instead of using any existing SWS description framework, their proposal uses an *ad-hoc Services Description Language*, in order to be able to define complex constraints. In addition, the resolution algorithm uses constraint propagation and rewriting, but performed by a subsumption algorithm, instead of a CSP solver.

Concerning WSMO discovery, Wang *et al.* propose an extension of the ontology to allow QoS-aware discovery [30]. The matchmaking is done by an *ad-hoc* algorithm to add QoS conditions to offers and demands. Their implementation has some limitations, as the algorithm can only be applied to real domain attributes, and is restricted to three types of relational operators.

Ruiz-Cortés *et al.* provide in [23] a framework to perform QoS-aware discovery by means of CP, in the context of non-semantic Web Services. They show the soundness of using CP to improve the automation of matchmaking from both theoretical and experimental points of view. Although CP solving is a NP-hard problem, the results of their experimental study allow to conclude that CP-based matchmakers are practically viable despite of its, theoretical, combinatorial nature. This work is the starting point of our approach on using CP to perform QoS-related stages of our hybrid SWS discovery proposal.

2.3 Frameworks

As an application of their previous work, Srinivasan *et al.* present an implementation to development, deployment and consumption of SWS [26]. It performs the discovery process using the proposals introduced in [19,27]. They show performance results and detail the implementation of OWL-S and UDDI integration, so it can be used as a reference implementation to OWL-S based discovery, but without QoS conditions.

IRS-II [18] is an implemented framework similar to WSMF [6], that is able to support service discovery from a set of demands. It uses descriptions of the reasoning processes called Problem Solving Methods (PSM), similar to OWL. Moreover, IRS-III [5] updates this previous implementation, using WSMO ontology to model SWS, and providing an architecture to discovery, composition and execution SWS. All of them can not handle with QoS conditions, although they are extensible so they may support them.

Another interesting proposal is done in [24], where Schlosser *et al.* propose a graph topology of SWS providers and clients, connected between them as in a peer-to-peer (P2P) network. In this scenario, searching, and specially publishing, are done very efficiently, without the need of a central server acting as a register of offers and demands. In addition, the network are always updated, due to an efficient topology maintenance algorithm. This structure of decentralized registries is proposed in METEOR-S for semantic publication and discovery of Web Services [29]. The semantic matching algorithm uses templates to search inputs and outputs of services described with ontological concepts, without the use of a specific reasoner, or the possibility to express QoS conditions. Although the matchmaking is too simple, the idea of a P2P network can be adopted in our proposal without troubles.

Our proposal is open to be implemented in the context of any of the presented frameworks in this section. The proposed architecture that we present in the following section does not impose any restriction on the SWS framework used (i.e. OWL-S, WSMO or METEOR-S), and can be composed of any number of the discovery engines discussed in Sec. 2.2, due to its hybrid nature. Furthermore, it can be materialized as the discovery component of implementations like IRS-III [5] or OWL-S IDE [26].

3 Our Proposal

The addition of constraints, specially QoS-related ones, to SWS descriptions, turns most approaches on discovering SWS insufficient, because they mainly use DLs, which are usually limited to logical and relational expressions when describing QoS conditions. CP becomes necessary to manage more complex QoS conditions, so a demand can be matched with the best available offer. Instead of using solely CP to perform the discovery, we present an hybrid solution that splits the discovery into different stages.

3.1 Hybrid Semantic Discovery Architecture

An abstract architecture of our proposal is sketched in Fig. 1, where we show how the different components are connected between them. Here, the dashed line defines the boundaries of our hybrid discovery engine.

Q document corresponds to the query that a client wants to use to discover services, i.e. the demand. This query may be expressed in any desired language that the scheduler can handle, such as a SPARQL query [21], a WSMO goal, a faceted search [20], or even it may be defined visually using a GUI.

R is the result set of offers that fulfill the query Q. It is the output of the discovery process, possibly being an empty set, the best offer, or an ordered list of offers by an optimality criterion. The format of this output should conform the specification of a concrete SWS framework in order to successfully invoke the discovered service(s).

The different stages of the hybrid discovery are performed by the best suited discovery engine. In Fig. 1, $E_1...E_n$ represent the engines to be used in each corresponding stage. The core component of our proposed architecture is responsible to send the input data to each engine, by decomposing the query Q

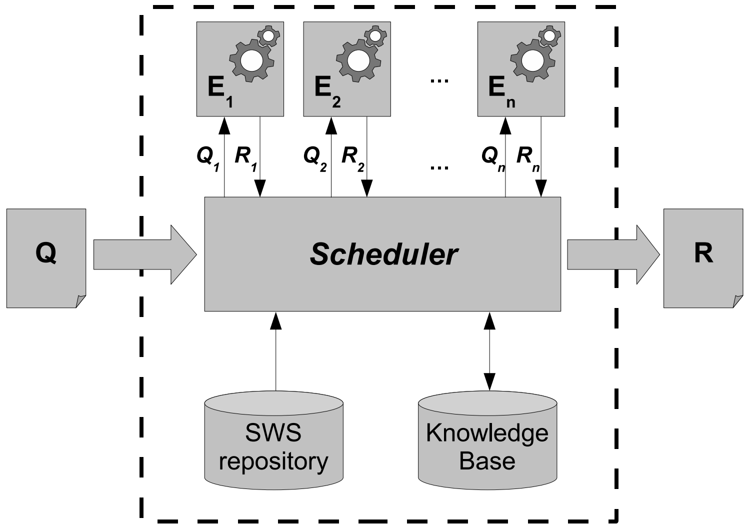


Fig. 1: Architecture of our hybrid discovery proposal.

in subqueries (Q_i), and to recover its corresponding output (R_i), joining all of them to output the final result R. These input and output formats depend on the concrete engine of each stage. Thus, if we are performing a QoS-aware stage, the input must be modeled as a CSP, so CP can be applied to perform this kind of stage. Additionally, it is possible to use a DLs engine to perform non-QoS discovery, or a template matchmaker [29], for instance.

Offers have to be published in some kind of repository so they can be matched with demands by means of the different discovery engines used in our approach. This SWS repository may be implemented in different ways: as a semantically-extended UDDI registry [26], as a decentralized P2P registry [29], or as a WSMO repository [5], for instance.

In addition, our architecture takes care of the NP-hard nature of optimization [4], so we propose to include a knowledge-base (KB) that cache already performed discoverings, so the execution of the discovery process becomes faster. Thus, we store executed queries related with their result set of SWS from the repository component, into the KB. IRS-II implementation uses a similar idea to accelerate discovery [18].

Finally, the core component of our proposal is the scheduler. It has to analyze the query Q, split the discovery task into stages, and communicate with discovery engines, in order, providing them with a correct input, and obtaining a corresponding output. These different outputs are processed stage by stage, so the set of matching offers from the SWS repository are gradually made smaller. Each discovery stage may be concurrently or sequentially launched in order, depending on the query nature. Moreover, the scheduler update the KB using the results of discovery process, which is output as R.

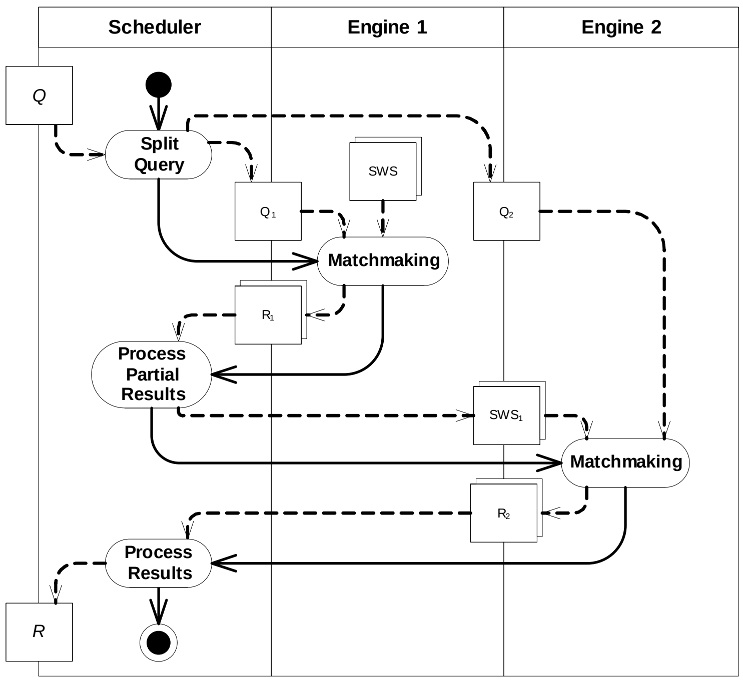


Fig. 2: Activity diagram of our discovery process.

Fig. 2 shows the activity diagram of an hybrid discovery process performed in two stages using two different engines. This diagram can be easily extended if we need more stages. For instance, using a similar format from [13], a query $Q = (ServiceProfile \cap A \geq 0.9)$, where A corresponds to availability, is split by the scheduler into two subqueries: $Q_{DL} = (ServiceProfile)$ being the part expressed in DLs, and $Q_{CP} = (\{A\}, \{[0..1]\}, \{A \geq 0.9, A = MTTF/(MTTF + MTTR)\})$ the part modeled by a CSP.[2] $ServiceProfile$ corresponds to the definition of a demand in terms of the OWL-S profile of a service. In this scenario, the scheduler perform a matchmaking firstly using a DLs engine with Q_{DL}, obtaining the offers that satisfy this subquery. Then, with this resulting subset of SWS from the registry, the scheduler performs a matchmaking using a CP engine and Q_{CP}, so the final result is the optimal offer that satisfies the whole query Q. For the sake of simplicity we do not contemplate the KB role in Fig. 2, because it only provides a way to speed up the process.

This hybrid discovery architecture has many advantages. It is loosely coupled, due to the possibility to use any discovery engine. Also, the input query format

[2] A CSP consists in a three-tuple of the form (V, D, C) where V is a finite, non-empty set of variables, D is a finite, non-empty set of domains (one for each variable) and C is a set of constraints defined on V. The solution space of a CSP is a set composed of all its possible solutions, and if this set is not empty, the CSP is said to be satisfiable.

is not restricted, as the scheduler can analyze a given query, so it can infer the concrete engines to use and their order. Moreover, our proposed architecture can be applied to any existing SWS framework and corresponding repositories, taking benefit of the wide range of tools already implemented. Our proposal is able to use the best suited engine to perform the corresponding search of a part of the input query, being used in most cases CP for QoS-related part, and DLs for non-QoS discovery, but without restrictions on adding more engines.

3.2 QoS-Aware Semantic Discovery

Focusing on the QoS-aware discovery stage, the scheduler sends the QoS-related part of the query to a CSP solver, so the set of offers that fulfills the requirements of a given demand can be obtained, or even obtain the optimal offer. To do so, QoS conditions and their involved QoS parameters, defined in demands and offers, must be mapped onto constraints in order to use a CSP solver.

Thus, each parameter must be mapped onto a variable (with its corresponding domain), and each condition must be mapped onto a constraint. At this point, we have to extend the demand and offer concepts previously presented because both of them may contain complementary information. We consider they are composed of two parts: requirements and guarantees. On the one hand, a demand δ is composed of two parts: δ^γ, which asserts the conditions that the client meets (i.e. γuarantees), and δ^ρ, which asserts the conditions that the provider shall meet (i.e. ρequirements). Similarly, an offer ω can also be considered composed of ω^γ (what it guarantees) and ω^ρ (what is required from its clients).

For example, consider the demand "The availability shall be less than 0.9, where $A = MTTF/(MTTF + MTTR)$" (δ^ρ); and the offer "The mean time to failure is from 100 to 120 minutes inclusive, while the mean time to repair is from 3 to 10 minutes inclusive" (ω^γ). Assuming that $MTTF$, $MTTR$ and A range over real numbers, their corresponding CSPs are defined as follows:

$$\begin{aligned}\delta^\rho &= (\{A, MTTF, MTTR\}, \{[-\infty, +\infty], [0, +\infty], [0, +\infty]\},\\ &\quad \{A \geq 0.9, A = MTTF/(MTTF + MTTR)\})\\ \omega^\gamma &= (\{MTTF, MTTR\}, \{[0, +\infty], [0, +\infty]\},\\ &\quad \{100 \leq MTTF \leq 120, 3 \leq MTTR \leq 10\})\end{aligned}$$

Additionally, the demand may also contain the condition "My host is in Spain" (δ^γ); and the offer "For American and British clients only" (ω^ρ), so the offer provider requires from its clients some guarantees. Consequently, assuming that $COUNTRY$ variable ranges over the powerset of $\Lambda = \{ES, US, UK, FR\}$, i.e. $\mathcal{P}(\Lambda)$, their corresponding CSPs are defined as follows:[3]

$$\delta^\gamma = (\{COUNTRY\}, \{\mathcal{P}(\Lambda)\}, \{COUNTRY = \{ES\}\})$$

[3] Note QoS parameters can be linked together in order to express more complex conditions, such as $\{COUNTRY = \{ES, UK, FR\} \Rightarrow 5 \leq MTTR \leq 10, COUNTRY = \{US\} \Rightarrow 5 \leq MTTR \leq 15\}$. These conditions can be interpreted as "the $MTTR$ is guaranteed to be between 5 and 10 if client is Spanish, British, or French, else between 5 and 15 if client is American".

$$\omega^{\rho} = (\{COUNTRY\}, \{\mathcal{P}(\Lambda)\}, \{COUNTRY \subseteq \{UK, US\}\})$$

The conditions previously expressed in natural language should be expressed in a semantic way, using QoS ontologies such as the one proposed by Maximilien *et al.* in [17]. Thus, semantically defining QoS parameters that take part in such conditions, and integrating these descriptions in any SWS framework, they can be interpreted later as a CSP so a solver can process them in the corresponding discovery stage.

These CSPs allow to check for consistency and conformance of offers and demands. A demand or an offer is said to be consistent if the conjunction of its corresponding CSPs (of requirements and guarantees) are satisfiable. On the other hand, an offer ω and a demand δ are said to be conformant if the solution space of the CSP of the guarantees of the offer (denoted by ψ^{γ}_{ω}) is a subset of the solution space of the CSP of the requirements of the demand (ψ^{ρ}_{δ}), and vice versa ($\psi^{\gamma}_{\delta} \subseteq \psi^{\rho}_{\omega}$) [23]. In the previous example, ω and δ are consistent, but they are not conformant, because $COUNTRY$ is guaranteed to be ES, but the offer requires it to be UK or US.

Finally, the ultimate goal of the matchmaking of offers and demands is to find a conformant offer that is optimal from the client's point of view. To do so, it becomes necessary to model the optimization task as a CSP, as with consistency and conformance checks. More specifically, finding the optimal can be interpreted as a Constraint Satisfaction Optimization Problem (CSOP), which requires to explicitly establish a preference order on the offer set. This order can be defined using a weighted composition of utility functions, which can be taken as a global utility function for the client.

Thus, each QoS parameter can have a utility function defined, and an associated weight to successfully describe how important the values that can take are for the client. Fig. 3 shows an example of how to discover optimal offers. In this case, we are assuming that the demand only specifies its requirements (Fig. 3a) and the offer only specifies what it guarantees (Fig. 3b), so the offer is conformant with the demand. The corresponding utility functions of the QoS parameters involved, i.e. $MTTF$ and $MTTR$, ranging over $[0, 1]$, are shown in Fig. 3c and 3d, respectively.

The utility function for $MTTF$ (Fig. 3c) is a piecewise linear function that defines a minimum utility if $MTTF$ is below 60 minutes; the utility grows linearly if $MTTF$ is between 60 and 120 minutes, and the utility reaches its maximum value if $MTTF$ is above 120. On the other hand, the utility function for $MTTR$ showed in Fig. 3d is a decreasing piecewise linear function. In order to obtain a global utility function of the offer, we consider that $MTTF$ has a weight of 70% and $MTTR$ 30%.

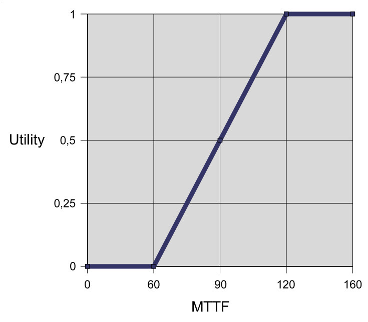
The offer from Fig. 3b must be checked for conformance with the demand from Fig. 3a, supposing that both descriptions have been previously checked for consistency, and that both are based on same QoS parameters, or they are defined using a compatible ontology. In this case, there is only one offer conformant, but there could be more than one, being necessary to obtain the optimal offer. To do so, utility functions for each offer have to be computed in

$$\delta^{\rho} \equiv A \geq 0.9 \wedge$$
$$A = \frac{MTTF}{MTTF + MTTR}$$

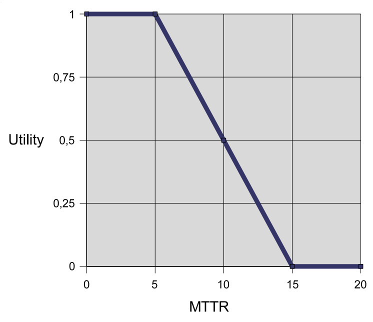
(a) Demand requirements.

$$\omega^{\gamma} \equiv 100 \leq MTTF \leq 120 \wedge$$
$$3 \leq MTTR \leq 10$$

(b) Offer guarantees.



(c) $MTTF$ utility function.



(d) $MTTR$ utility function.

```
//variables
range TYPE_MTTF 0..255;
var TYPE_MTTF MTTF;
range TYPE_MTTR 0..255;
var TYPE_MTTR MTTR;
range TYPE_UTILITY 0..100;
var TYPE_UTILITY U_MTTF;
var TYPE_UTILITY U_MTTR;
var TYPE_UTILITY UTILITY;
minimize
 UTILITY
subject to {
 // Offer guarantees
100<=MTTF<=120;
3<=MTTR<=10;
// Utility function of MTTF
MTTF<=60 => U_MTTF=0;
60<MTTF<=120 =>60*U_MTTF=MTTF-60;
MTTF>120=> U_MTTF=1;
// Utility function of MTTR
MTTR<=5 => U_MTTR=1;
5<MTTR<=15 => 10*U_MTTR=15-MTTR;
MTTR>15 => U_MTTR=0;
// Utility aggregate of matching
UTILITY = 70*U_MTTF + 30*U_MTTR;
};
```

(e) OPL model for computing utility.

Fig. 3: An example on obtaining optimal offers.

order to compare them and get the maximum utility value, which corresponds with the optimal offer. In Fig. 3e we show the OPL [10] model for the computing of the utility function of the showed offer.

Note that we compute the minimum value of the utility function, taking the worst-case scenario. This way, we say that an offer ω is optimal with regard to a utility function U if the minimum value of this function is the maximum among minimum values of all conformant offers. It is also possible to take other approaches when computing the utility function, like using the maximum value, a mean value, or the more general case of a weighted composition of the maximum and minimum value [12].

4 Conclusions and Future Work

In this work, we show that using a unique engine to discover SWS is not appropriate, due to each engine is usually designed for a concrete kind of search. For instance, DLs reasoners are well suited when discovering SWS in terms of concepts and relations, but they can not handle complex numerical QoS conditions. Although there are extensions to allow concrete domains in DLs, reasoners have to implement them, and they may bring undecidability results.

We present an hybrid solution that consists in a n-stages discovery process, where each stage is performed using the most appropriate technique. Furthermore, we propose to use CP to perform QoS-aware discovery stages, so the optimal service(s) offered that fulfills a given demand can be found. In addition, our proposed architecture is extensible and loosely coupled, allowing to define complex QoS conditions, and to use utility functions based on QoS parameters to obtain the optimal offer. This architecture does not impose any restriction on the SWS framework and repository to use, allowing its materialization as a discovery component for current SWS implementations.

For future work, we are considering to define more precisely the scheduler and its interaction with the rest of the components. The query split mechanism has to be characterized, so do the results merging for each engine. Thus, a catalog of stages would be defined, including their order of execution. Moreover, we are considering to extend current SWS frameworks using a QoS ontology to define QoS parameters and conditions, allowing to express complex arithmetic, relational, and logical expressions in demands and offers.

Acknowledgments. The authors would like to thank the reviewers of the *5th International Conference on Service Oriented Computing*, whose comments and suggestions improved the presentation substantially.

References

1. F. Baader and U. Sattler. Description logics with aggregates and concrete domains. *Information Systems*, 28(8):979–1004, December 2003.
2. B. Benatallah, M. Hacid, C. Rey, and F. Toumani. Semantic reasoning for web services discovery. In *WWW Workshop on E-Services and the Semantic Web*, 2003.
3. S. Benbernou and M. Hacid. Resolution and constraint propagation for semantic web services discovery. *Distributed and Parallel Databases*, 18(1):65–81, 2005.
4. P. Bonatti and P. Festa. On optimal service selection. In *14th international conference on World Wide Web*, pages 530–538, 2005.
5. L. Cabral, J. Domingue, S. Galizia, A. Gugliotta, V. Tanasescu, C. Pedrinaci, and B. Norton. IRS-III: A broker for semantic web services based applications. In *International Semantic Web Conference*, pages 201–214, 2006.
6. D. Fensel and C. Bussler. The web service modeling framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.
7. J. González-Castillo, D. Trastour, and C. Bartolini. Description logics for matchmaking of services. Technical Report HPL-2001-265, Hewlett Packard Labs, 2001.

8. V. Haarslev and R. Möller. RACER system description. In *Automated Reasoning, First International Joint Conference, IJCAR 2001*, pages 701–706, 2001.
9. V. Haarslev and R. Möller. Practical Reasoning in RACER with a Concrete Domain for Linear Inequations. In *Int. Workshop on Description Logics*, 2002.
10. P. Van Hentenryck. Constraint and integer programming in OPL. *INFORMS Journal on Computing*, 14(4):345–372, 2002.
11. I. Horrocks. FaCT and iFaCT. In *Int. Workshop on Description Logics*, 1999.
12. K. Kritikos and D. Plexousakis. Semantic QoS metric matching. In *ECOWS 2006*, pages 265–274. IEEE Computer Society, 2006.
13. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Int. World Wide Web Conference*, pages 331–339, 2003.
14. C. Lutz. Description logics with concrete domains – a survey. In *Advances in Modal Logic*, pages 265–296, 2002.
15. C. Lutz and U. Sattler. A proposal for describing services with DLs. In *Int. Workshop on Description Logics*, 2002.
16. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, et al. OWL-S: Semantic Markup for Web Services. Technical Report 1.1, DAML, November 2004.
17. E. M. Maximilien and M. P. Singh. A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, 8(5):84–93, 2004.
18. E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A framework and infrastructure for semantic web services. In *Int. Semantic Web Conference*, pages 306–318, 2003.
19. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Int. Semantic Web Conference*, pages 333–347, 2002.
20. R. Prieto-Díaz. Implementing faceted classification for software reuse. *Commun. ACM*, 34(5):88–97, 1991.
21. E. Prudh́ommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical Report Working Draft, W3C, March 2007.
22. D. Roman, H. Lausen, and U. Keller. Web Service Modeling Ontology (WSMO). Technical Report D2 v1.3 Final Draft, WSMO, October 2006.
23. A. Ruiz-Cortés, O. Martín-Díaz, A. Durán Toro, and M. Toro. Improving the automatic procurement of web services using constraint programming. *Int. J. Cooperative Inf. Syst*, 14(4):439–468, 2005.
24. M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. A scalable and ontology-based P2P infrastructure for semantic web services. In *Peer-to-Peer Computing*, pages 104–111, 2002.
25. K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding semantics to web services standards. In *Intl. Conference on Web Services*, pages 395–401, 2003.
26. N. Srinivasan, M. Paolucci, and K. Sycara. Semantic web service discovery in the OWL-S IDE. In *Hawaii International Conference on Systems Science*, 2006.
27. K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composition of semantic web services. *J. Web Sem.*, 1(1):27–46, 2003.
28. K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan. Dynamic discovery and coordination of agent-based semantic web services. *IEEE Internet Computing*, 8(3):66–73, 2004.
29. K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, et al. METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services. *Inf. Tech. Management*, 6(1):17–39, 2005.
30. X. Wang, T. Vitvar, M. Kerrigan, and I. Toma. A QoS-aware selection model for semantic web services. In *ICSOC 2006*, pages 390–401, 2006.