A Formal Model for Classifying Trusted Semantic Web Services

Stefania Galizia, Alessio Gugliotta, Carlos Pedrinaci

Knowledge Media Institute
The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK
(S.Galizia, A.Gugliotta, C.Pedrinaci)@open.ac.uk

Abstract. Semantic Web Services (SWS) aim to alleviate Web service limitations, by combining Web service technologies with the potential of Semantic Web. Several open issues have to be tackled yet, in order to enable a safe and efficient Web services selection. One of them is represented by trust. In this paper, we introduce a trust definition and formalize a model for managing trust in SWS. The model approaches the selection of trusted Web services as a classification problem, and it is realized by an ontology, which extends WSMO. A prototype is deployed, in order to give a proof of concept of our approach.

Keywords: Semantic Web services, Selection, Trust, Classification

1 Introduction

Semantic Web services (SWS) research aims at automating the development of Web service-based applications through semantic Web technology. By providing formal representation with well-defined semantics, SWS facilitate the machine interpretation of Web service (WS) descriptions. According to SWS vision, when a client expresses a goal that it wishes to achieve, the most appropriate Web service is automatically discovered and selected on the basis of the available semantic descriptions. Since the user does not know a priori the selected WS, the notion of trust should play an important role during the WS selection phase. However, the most common approaches for describing SWS, such as WSMO [3] or OWL-S [12], do not currently provide exhaustive means to model trust, and thus do not support trust-based selection of WS.

Notice that trust is a multifaceted concept. A trust understanding can indeed involve multiple – and not always the same - parameters, such as Quality of Service (QoS), reputation and security.

In our opinion, the main issue with representing trust in all its faces lies in its contextual nature – i.e. the same user may have different trust understandings in different contexts. For example, a user may trust a WS with a highly rated security certificate whenever she has to provide her credit card details. Conversely, the same user weights the opinions of past users about a specific WS in other situations – i.e. the evaluation of the WS reputation is a priority in the current trust understanding of the user. Moreover, distinct users may privilege different trust parameters in the same

context; in this case, their priorities may depend on their different personal preferences.

In this paper, we introduce a definition and formalize an abstract model for trust in SWS that enables interacting participants – i.e. both WS users and providers - to represent and utilize their own trust understanding with a high level of flexibility, and thus take the possible multiple interacting contexts into account. The essential contribution of our approach is therefore a generally applicable yet completely automated mechanism for selecting trusted services according to context-specific criteria.

Specifically, in our model we characterize the trust-based WS selection as a classification problem. Firstly, all participants specify their own requirements and guarantees about a set of trust parameters. Then, at runtime, our goal is to identify the class of WS that matches the trust statements of involved participants, according to an established classification criterion. To accomplish this, we have based the proposed model on a general-purpose classification library.

In order to apply our approach to an existing SWS working environment – and thus verify its benefits - we represented our model within a specific ontology: Web Services Trust-management Ontology (WSTO). The latter makes use of WSMO as reference approach for SWS. WSMO is in fact the underlying model of IRS-III [2], the SWS execution environment developed within our research group. As a result, we enhanced IRS-III with the trust-based selection of WS.

It is worth highlighting that an earlier version of WSTO was described in a previous work [5]. Whereas in [5] we outlined the idea of characterizing trust as a classification process, in the present work we propose a more complex model that is able to accommodate multiple trust understandings and parameters. Moreover, while in [5] we proposed a general thesis on the different meanings of trust, we did not supply our definition of trust. We now provide our trust definition as well as its formal semantics.

The paper is organized as follows: Section 2 provides the background knowledge useful for placing our work; in Section 3, we outline our approach, while in Section 4 we provide the formal details of our methodology; in Section 5, we describe an implemented application; Section 6 compares our approach with related work and, finally, Section 7 concludes the paper and outlines our future work.

2 Background

In this section, we first outline WSMO, our basic vision of SWS, and its ontological specification. Then, we outline the different existing approaches on trust.

WSMO.

The Web Service Modelling Ontology (WSMO) [3] is a formal ontology for describing the various aspects of services in order to enable the automation of Web service discovery, composition, mediation and invocation. The metamodel of WSMO defines four top level elements:

- *Ontologies* provide the foundation for describing domains semantically. They are used by the three other WSMO elements.

- *Goals* define the tasks that a service requester expects a Web service to fulfill. In this sense they express the requester's intent.
- *Web Service descriptions* represent the functional behavior of an existing deployed web service. The description also outlines how web services communicate (choreography) and how they are composed (orchestration).
- *Mediators* handle data and process interoperability issues that arise when handling heterogeneous systems.

One of the main characteristic features of WSMO is the linking of ontologies, goals and web services by mediators which map between different ontological concepts within specific WSMO entity descriptions. In order to facilitate appropriate mapping mechanisms, four classes of mediators are considered within WSMO. For example, an OO-mediator may specify an ontology mapping between two ontologies whereas a GG-mediator may specify a process or data transformation between two goals.

Classification Library.
The classification framework that we use and extend for our work is a library of generic, reusable components developed within the European project IBROW [9]. Its purpose is to support the specification of classification problem solvers. The basic structure is the UPML framework [4], on which WSMO is based. The library has been specified in the OCML modelling language [8], and implemented in IRS-III [2].

Within the classification framework, we use the term 'observables' to refer to the known facts we have about the object (or event, or phenomenon) that we want to classify. Each observable is characterized as a pair of the form *(f, v)*, where *f* is a feature of the unknown object and *v* is its value. Here, we take a very generic viewpoint on the notion of feature. By feature, we mean anything which can be used to characterize an object, such as a feature which can be directly observed, or derived by inference. As is common when characterizing classification problems - see, e.g., [19], we assume that each feature of an observable can only have one value. This assumption is only for convenience and does not restrict the scope of the model.

The solution space specifies a set of predefined classes (solutions) under which an unknown object may fall. A solution itself can be described as a finite set of feature specifications, which is a pair of the form *(f, c)*, where f is a feature and *c* specifies a condition on the values that the feature can take. Thus, we can say that an observable *(f, v)* matches a feature specification *(f, c)* if *v* satisfies the condition *c*.

As we have seen, generally speaking, classification can be characterized as the problem of explaining observables in terms of predefined solutions. To assess the explanation power of a solution with respect to a set of observables we need to match the specification of the observables with that of a solution. Given a solution, *sol*: $((f_{sol1}, c_1).....(f_{solm}, c_m))$, and a set of observables, *obs*: $((f_{ob1}, v_1).....(f_{obn}, v_n))$, four cases are possible when trying to match them:

- A feature, say f_j, is inconsistent if $(f_j, v_j) \in obs$, $(f_j, c_j) \in sol$ and v_j does not satisfy c_j;
- A feature, say f_j, is explained if $(f_j, v_j) \in obs$, $(f_j, c_j) \in sol$ and v_j satisfies c_j;
- A feature, say f_j, is unexplained if $(f_j, v_j) \in obs$ but f_j is not a feature of *sol*;
- A feature, say f_j, is missing if $(f_j, c_j) \in sol$ but f_j is not a feature of *obs*.

Given these four cases, it is possible to envisage different solution criteria. For instance, we may accept any solution, which explains some data and is not inconsistent with any data. This criterion is called positive coverage [14]. Alternatively, we may require a complete coverage - i.e., a solution is acceptable if and only if it explains all data and is not inconsistent with any data. Thus, the specification of a particular classification task needs to include a solution (admissibility) criterion. This in turn relies on a match criterion, i.e., a way of measuring the degree of matching between candidate solutions and a set of observables. By default, our library provides a match criterion based on the aforementioned model. That is, a match score between a solution candidate and a set of observables has the form *(I, E, U, M)*, where *I* denotes the set of inconsistent features, *E* the set of explained features, *U* the set of unexplained features and *M* the set of missing features. Of course, users of the library are free to specify and make use of alternative criteria.

In many situations, specifying the conditions under which a candidate solution is indeed a satisfactory solution is not enough. In some cases, we may be looking for the best solution, rather than for any admissible one. In these cases we need to have a mechanism for comparing match scores and this comparison mechanism becomes then part of the specification of the match criterion. By default, our library includes the following score comparison criterion.

Given two scores, $S_1 = (i_1, e_1, u_1, m_1)$ and $S_2 = (i_2, e_2, u2, m_2)$, we say that S_2 is a better score than S_1 if and only if:

$(i_2 < i_1) \vee$

$(i_2 = i_1 \wedge e_1 < e_2) \vee$

$(i_2 = i_1 \wedge e_2 = e_1 \wedge u_2 < u_1) \vee$

$(i_2 = i_1 \wedge e_2 = e_1 \wedge u_2 = u_1 \wedge m_1 < m_2)$

In the notation above $x_i < x_j$ indicates that the set x_i contains less elements than the set x_j.

In conclusion, our analysis characterizes classification tasks in terms of the following concepts: observables, solutions, match criteria, and solution criteria.

Trust Approaches.

Since trust can have different meaning in different contexts, several specifications can be found in literature. We can classify existing models into the following three main approaches:

- *Policy-based*. Policies are a set of rules that specify the conditions to disclose own resources;
- *Reputation based*. Reputation based approaches make use of rating coming from other agents or a central engine, by heuristic evaluations;
- *Trusted Third Party-based (TTP)*. Trusted Third Party based models use an external, trusted, entity that evaluates trust.

These general approaches can be refined and/or combined in order to build a concrete trust establishment solution that can be deployed in a real system.

Many models [11, 15] formulate trust policies in semantic Web services by security statements, such as confidentiality, authorization, authentication. W3C Web service architecture [18] recommendations base trust policies on security consideration, even if the way to disclose their security policies is still not clear.

Some policy-based models rely on a TTP, which works as a repository of service description and policies [21] and meanwhile as an external matchmaker that evaluates service trustworthiness according to given algorithms.

Reputation-based models reuse concepts and approaches taken from Web-based social networks. In SWS as well as in social networks, trust is a central issue. In both the cases, interactions take place whenever there is trustworthiness. The idea is that involved participants express their opinion on other participants, by means of a shared vocabulary. Several algorithms for trust propagation and different metrics have been defined, most of them are more generically Quality of service (QoS) based [7; 17], since they consider the service ability the main trust statement.

3 Our Approach

We propose a formal approach for managing trust among semantic Web services, based on two ontologies: WSMO and the classification task ontology, both introduced in Section 2. We build an ontology - named Web Service Trust-management Ontology (WSTO) - that reuses the main concept of those ontologies, and extends them, for supporting SWS trust management. In our model, user preferences are the main elements on which Web service selection depends. Essentially, the user can decide which parameters should be considered in order to determine which class of Web services are trusted, in a given context. We embed trust-based SWS selection in a classification framework, whereas the classification task ontology provides the overall methodology that we adopt for managing trust. For our purposes, we classify Web services according to both the user and Web services trust requirements and guarantees.

In WSTO, the key concepts are *user*, *ws* and *goal*, where *user* denotes the service requester and *ws* is the service provider. Following the basic WSMO notions, a *goal* represents the service requester's desire or intention. The user usually expresses different trust requirements in achieving different goals. For example, she can be interested in data accuracy when retrieving timetable information, and security issues when disclosing her bank accounts. On the other hand, the *ws* aims to provide a set of trustworthy statements, in order to reassure the requester as well as to appear as attractive as possible.

The participants (*ws* and *user*) are associated with trust profiles, represented in WSTO by the class *trust-participant-profile*. A profile is composed of a set of trust requirements and guarantees. *Trust-guarantee* represents observables, pairs of feature and corresponding value *(f, v)*, while *trust-requirement* represents candidate solutions, pairs of feature and condition *(f, c)*.

We distinguish three logical elements in trust requirements: (i) a set of candidate solutions for expressing conditions on guarantees promised by the relevant parties; (ii) a candidate solution for requesting their reliability; and, (iii) a candidate solution for requesting their reputation evaluation. In a participant profile, the three elements are optional; choice depends strictly on the participant preferences in matter of trust.

In turn, the participant trust guarantees have three components: (i) a set of observables for representing the promised trust guarantees; (ii) an observable corresponding

to the evaluation of the participant reliability; and, (iii) an observable for representing the reputation level of the participant. Whereas the promised trust guarantees are a set of promised values stated by the participant - such as *(execution-time, 0.9)* and *(data-freshness, 0.8)* - reliability and reputation guarantees are computed on-the-fly within dedicated execution environments (IRS-III in our use case, see Section 5). As mentioned earlier, a participant profile is composed of requirements as well as guarantees. For example, a Web service may expose high *data-freshness* and strong *confidentiality* as guarantees. Moreover, the same Web service may define security requirements, such as conditions under which a service requester can access it.

Given observables and conditions, a classification criterion is now necessary to classify Web services and find the appropriate class that addresses both user and Web service requirements and guarantees. The classification match criterion we apply is the one described in Section 2, although other classification criteria can be easily represented in WSTO.

As solution admissibility criteria, we can apply *complete coverage* and *positive coverage*. The former demands that all requirements of the interaction have to be satisfied; the latter accepts that some requirements are fulfilled and no inconsistencies exist. Our classification library implements two different classification methods: *single-solution-classification*, and *optimal-classification*. The former implements a hill climbing algorithm with backtracking to find a suitable solution; the latter executes an exhaustive search for an optimal solution. We make use of the *optimal-classification-task* and redefine it as WSMO goal[1], *optimal-classification-goal*, whose *participant-profiles* and *trusted-ws* represent the pre-conditions and post-conditions of the goal, respectively.

Notice that the proposed model is comprehensive of all trust approaches listed in the previous section. In fact, it embeds a policy-based trust management, since the interacting participants express their trust policies in their – semantically described – profiles, while the adopted SWS broker will behave as a TTP by storing participant profiles and reasoning on them. Moreover, the reputation module enables a WS selection based also on reputation ontological statements.

4 The Formal Model

This section provides the formal definition of trust we adopt in our approach, as well as its semantics. Trust is a binary evaluation of trustworthiness: “trust” or “distrust”. Whenever conditions for trustworthiness are established, the interaction between participants occurs; otherwise, it is not possible.

The trustworthiness $T_u^g(ws)$ that a user *u* perceives towards a Web service *ws*, when she invokes a goal *g*, is given by the expression:

$$T_u^g(ws) = \Psi\left(P_g(u), \Omega_g(ws)\right)$$

Ψ is a classification operator, that provides a class of Web services matching the user’s trust requirements, according to the match criterion presented in Section 3. If

[1] WSMO goals can be seen as an evolution of UPML tasks.

trust requirements do not meet any Web service trust guarantees, Ψ returns a *null* value. This means that no trusted communication can occur.

Trust as perceived by the user *u*, can be either strong or weak. It is strong when the operator Ψ classifies Web services by adopting complete coverage as solution admissibility criterion. When the criterion selected is positive coverage, trust is regarded as weak. We did consider using two different operators - Ψ_s for strong trust, and Ψ_w for weak trust - however, we decided against this in order to increase the readability of our notations. Without losing generality, in the rest of this section, we assume only strong trust.

$P_g(u)$ is a function which selects a profile from the set of trust profiles associated (provided or accepted) with the user, according to the current goal. As mentioned earlier, a user can have different trust preferences in different contexts. The *current-selected-profile* is used to associate a trust-profile with a goal, according to the user's ontological statements.

The user trust profile suitable for a given goal is represented by a list of user requirements: $(f_1, c_1), .., (f_n, c_n), (f_{rL}, c_{rL}), (f_r, c_r)$. The user requirements $(f_1, c1), .., (f_n, c_n)$ are conditions on WS promised guarantees. They can involve QoS statements, or concern security issues. Moreover, the user could be interested to know more about the reputation and the reliability of the Web services she will interact with. The requirements (f_{rL}, c_{rL}) and (f_r, c_r) respectively express conditions on reliability and other user preferences concerning Web service behaviors.

Given a goal *g*, $\Omega_g(ws)$ is a complex operator that provides information about the WS profile, where *ws* satisfies *g*. The operator provides thus (i) the guarantees promised by the WS, (ii) a record of WS monitored behavior, and (iii) WS behavior as evaluated by other users. For conformity, we also refer to components (ii) and (iii) as guarantees, however they are automatically calculated by IRS-III, and are not strictly speaking guarantees. We should also note that, in principle, *ws* reputation and its historical evaluation may not always reassure the user.

More formally, Ω_g has three components:

$$\Omega_g = \left(\Pi_p^g, \Pi_h^g, \Pi_r^g\right)$$

Given a Web service *ws*, satisfying a goal *g*, $\Pi_p^g(ws)$ supplies the component of the *ws* profile published by *ws* itself. The *ws* guarantees, are pairs *(feature, value)*: $\{(f_{p1}, v_{p1}), ..., (f_{pm}, v_{pm})\}$. The published guarantees can involve QoS parameters, certificated security parameters issued by Certification Authorities, or any ontological statements certificated by TTP or simply provided by the WS for trust assurance purposes. The values $v_{p1},\ldots, v_{pm}$ are normalized and homogenized. They are normalized to non-negative real numbers in the range [0,1]. Moreover, we assume that they are homogeneously scaled, where higher values correspond to higher performance. For example, higher performance for the parameter "execution time" would normally be indicated by a smaller value, but we normalize to a scale where a higher value indicates better performance. We are aware that this process can increase complexity, especially for those guarantees related to security issues, however, describing the

normalization process is out of the scope of our current work. For alleviating the difficulty of representing numerical normalized values, we use a number of previously described heuristics.

$\Pi_h^g(ws)$ assigns the value v_{rL} to the *ws* reliability f_{rL}, by stating the observable (f_{rL}, v_{rL}). Reliability is a measure of how the Web service behaviour conforms with its related guarantees. Let $F_p = \{f_{p1},\ldots, f_{pm}\}$ be the set of features with associated values of guarantees, and $F_h = \{f_{h1},\ldots,f_{hk}\}$ the set of the monitored features for *ws*. We define $F_{ph} = F_p \cap F_h = \{f_1,\ldots,f_j\}$, as the set of both promised and monitored features for *ws*. Whenever a feature is monitored more than once, we consider only the last observed, because we assume that Web service performance can alter, and the last value is closer to its predicted behavior.

We calculate the feature *conformance*, as defined by [16]. For every feature f_i belonging to F_{ph} with $1 \leq i \leq j$, we determine the conformance of f_i by value $\delta = \frac{v_i^h - v_i^p}{v_i^p}$, where v_i^h is the normalized monitored value associated to f_i, and v_i^p is its normalized promised value, for the Web service *ws*.

The conformance value δ_i falls in the range [-1, 1]. It is a negative value when the promised value is better than the monitored one. It holds 0 when $v_i^p = v_i^h$, i.e., the promised value corresponds to the monitored one. Finally, when the Web service behaviour around the feature f_i is better than promised, δ_i is a positive value.

If $\delta_i \geq 0$ for every $1 \leq i \leq j$, then $\Pi_h^g(ws)$ will have the value 1, the maximum value for reliability, otherwise, reliability is represented by the normalized arithmetic average of each feature's conformance:

$$\Pi_h^g(ws) = v_{rL} = \left\| \frac{\sum_{i=1}^{j} \delta_i}{j} \right\|$$

Web service reliability, evaluated through the operator $\Pi_h^g(ws)$, provides a value for the feature f_{rL}, where the observable (f_{rL}, v_{rL}) is a component of the *ws* profile. v_{rL} is a guarantee that will be automatically generated by the adopted SWS broker by processing the *ws* published and monitored guarantees. For example, IRS - our reference SWS broker - automatically logs all interactions with Web services [13] and thus reliability is straightforward to compute.

The operator $\Pi_r^g(ws)$ provides a measure of Web service reputation. Users who have previously interacted with WS can supply ontological statements for describing perceived trustworthiness. These statements are observables - pairs *(feature, value)* - as annotated by users.

We introduce a reputation evaluation for making our model as context/user oriented as possible, because some users may be interested in the opinions that come from previous requesters. Nevertheless, we do not intend to emphasize this aspect of our trust evaluation because reputation statements may derive from malicious users

interested in providing false evaluations for a variety of reasons. Therefore, we consider only reputation statements that have high conformance.

Let $F_r = \{f_1,\ldots f_{rr}\}$ be a set of features, and $\{(f_i, v_{i1})..,\ \{(f_i, v_{ij})\}$ the corresponding *ws* observables for the feature f_i, with $1 \leq i \leq rr$, respectively reputed by the users $\{u_{i1},.. u_{ij}\}$. We consider the reputation around the feature f_i can be estimated if and only if the standard deviation *SDi* from the average of the normalized values $\{v_{i1}..,\ v_{ij}\}$ is lower than a given threshold *D*.

We can now define $\Pi_r^g(ws)$ as the average trustworthiness perceived by the users towards the Web service *ws*:

$$\Pi_r^g(ws) = v_r = \frac{\sum_{i=1}^{rr} w_i \bar{v}_i}{\sum_{i=1}^{rr} w_i}$$

Where w_i is a weight that excludes the reputation statements that cannot be estimated. It can hold {0,1}: $w_i = 1$ when the $SDi \leq D$, otherwise its value is 0. The value v_r is assigned to the feature f_r, where the observable (f_r, v_r) is a component of the *ws* profile, computed within IRS-III.

Having extracted the participant profiles, the operator Ψ classifies Web services, i.e., it solves the problem of finding a class that best explains a set of known Web service guarantees, according to user trust requirements. The output is binary: the WS class exists or not, which corresponds to the trust or distrust value for the function $T_u^g(ws)$.

5 Case Study: A Trusted Virtual Travel Agent

The proposed formal model has been implemented within an existing SWS execution environment: IRS-III [2]. The reasons for adopting IRS-III are the following: firstly, this framework has been designed and built within our institution; secondly, WSMO (Section 2) has been incorporated and extended as the core IRS-III epistemological framework; finally, the classification library we use and extend (Section 3) is represented in OCML [8], the ontological representation language used by IRS-III.

IRS-III is a platform and a broker for developing and executing SWS. By definition, a broker is an entity which mediates between two parties and IRS-III mediates between a service requester and one or more service providers. A core design principle for IRS-III is to support capability-based invocation. A client sends a request which captures a desired outcome or goal and, using a set of semantic Web service descriptions, IRS-III will: a) discover potentially relevant Web services; b) select the set of Web services which best fit the incoming request; c) mediate any mismatches at the data, ontological or business process level; and d) invoke the selected Web services whilst adhering to any data, control flow and Web service invocation constraints. Additionally, IRS-III supports the SWS developer at design time by providing a set of tools for defining, editing and managing a library of semantic descriptions and also for grounding the descriptions to either a standard Web service

with a WSDL description, a Web application available through an HTTP GET request, or code written in a standard programming language (currently Java and Common Lisp).

In our work, we implemented a new IRS-III module that exploits WSTO and thus enhances the current functional-based (i.e. based on pre and post conditions, assumption and effect descriptions) selection mechanism of IRS-III with a trust-based selection mechanism. Given several Web services, semantically annotated in IRS-III and all with the same functional capability, but different trust guarantees, the class of Web services selected will be the one that matches closest with the user trust requirements, according to the classification mechanism introduced in the previous section.

As a test-bed for our module, we deployed a prototype application (the Virtual Travel Agency) and compared the existing version of IRS-III (non trusted) with the improved one (trusted). In the proposed scenario, IRS-III acts as SWS execution environment as well as TTP, by storing participant profiles and reasoning on them. The current prototype considers participant observables and needs, but it does not include the reputation module and the historical monitoring. The prototype is implemented in OCML and Lisp. The goal is to find the train timetable, at any date, between two European cities. Origin and destination cities have to belong to the same country (European countries involved in our prototype are: Germany, Austria, France and England). The client that uses this application in IRS-III publishes her trust-profile, with trust requirements and/or trust guarantees. In our prototype, we provide three different user profiles and three different Web services, able to satisfy the user goal. User profiles are expressed through trust requirements, without trust guarantees. All user requirements are performed in terms of security parameters: *encryption-algorithm*, *certification-authority* and *certification-authority-country*. Every user expresses a qualitative level of preference for every parameter.

```
USER4
(def-class trust-profile-USER4 (trust-profile)
      ((has-trust-guarantee :type guarantee-USER4)
       (has-trust-requirement :type requirement-USER4)))

(def-class requirement-USER4 (security-requirement)
      ((encryption-algorithm :value high)
       (certification-authority :value medium)
       (certification-authority-country :value medium)))

USER5
........
(def-class requirement-USER5 (security-requirement)
      ((encryption-algorithm :value medium)
       (certification-authority :value low)
       (certification-authority-country :value low)))

USER6
........
(def-class requirement-USER6 (security-requirement)
      ((encryption-algorithm :value low)
       (certification-authority :value high)
       (certification-authority-country :value high)))
```

Listing 1 User Profiles

For instance, the *user4* would like to interact with a Web service that provides a high security level in terms of encryption algorithm, but she accepts medium value for Certification Authority (CA) and CA country. Representing user requirements in a qualitative way seems to be more user-friendly. Heuristics are necessary for expressing quantitative representations in qualitative form. The listing below is an example of heuristic.

```
ENCRYPTION-ALGORITHM HEURISTIC
(def-instanceencryption-algorithm-abstractor abstractor
((has-body '(lambda (?obs)
             (in-environment
               ((?v . (observables-feature-value ?obs
                                'encryption-algorithm)))
               (cond ((== ?v DES)
                      (list-of 'encryption-algorithm 'high
                               (list-of (list-of
                               'encryption-algorithm ?v))))
                     ((== ?v AES)
                      (list-of 'encryption-algorithm 'medium
                               (list-of (list-of
                               'encryption-algorithm ?v))))

                     ((== ?v RSA)
                      (list-of 'encryption-algorithm 'low
                                (list-of (list-of
                               'encryption-algorithm ?v))))))))
```

Listing 2 Encryption Algorithm Heuristic

The heuristic *encryption-algorithm-abstractor* establishes that whenever the encryption algorithm adopted by a Web service provider is like *DES*, then its security level is considered high. Whenever both User and Web service describe their profiles, they implicitly agree with the qualitative evaluation expressed by the heuristic. In turn, whenever the Web service provider makes use of an algorithm like *AES*, according to the heuristic in Listing 2, its encryption ability is deemed medium, otherwise, if the adopted algorithm is like *RSA*, the security level is low. Other heuristics provide qualitative evaluations of *CAs*, and *CA countries*. For instance, security level of *globalsign-austria* is retained high, conversely German CAs are considered medium-secure.

The user can apply these heuristics, or define her own, sharing her expertise and knowledge with other users. Alternatively, the user can even express her requirements in a precise/quantitative way, by specifying the exact values expected from Web service guarantees, for example, the CA issuing security token has to be *VeriSign*. Given several Web services, semantically described in IRS-III, all with the same capability, but different trust profiles, the class of Web service selected will be the one that matches closest with the user trust profile.

We developed a user-friendly Web application to test our implementation, which is available at http://lhdl.open.ac.uk:8080/trusted-travel/trusted-query.

The snapshot in Figure 1 shows the Web application interface. The user who would like to know train timetable between two European cities enters the desired city names and date. The user owns a trust profile associated to her name: *dinar* is instance of *user4* trust profile, *vanessa* of *user5*, *stefania* of *user6*.

Figure 1 Web Application

Whenever the application starts, IRS-III recognizes from the user name, the trust user profile. In the prototype, the requirements expressed by the user are treated as candidate solutions within the classification goal.

The class of Web services whose trust guarantees best match with user requirements is selected. As we applied the complete coverage criterion, the match is strict, that means every user requirement is explained (matches with a Web service trust guarantee) and none is inconsistent.

Figure 2 Web Application Output of the user "dinar" Invocation

Figure 2 is a snapshot of the resulted trusted VTA booking. The application returns the list of Web services able to satisfy the user goal, and that one invoked, which matches with *dinar* trust requirements. It follows the Web service output, the requested timetable. The application can easily be tested with the other user instances implemented, *vanessa* and *stefania*. It can be noticed that *vanessa* trust profile matches with Web service class *get-train-timetable-service-T3*, while *stefania* with *get-train-timetable-service-T2*.

The "non-trusted" based version of the application is available at http://lhdl.open.ac.uk:8080/trusted-travel/untrusted-query. This application implements a virtual travel agent based on the standard IRS-III goal invocation method. The output returns only the train timetable requested, without any trust-based selection.

6 Related Work

A number of current approaches model social aspects of trust [6], while some recent efforts in the last few years concern service-oriented views of trust [1]. However, few approaches provide methodologies for managing trust in a SWS, and none comprehensively incorporate all possible approaches of trust (policy, reputation TTP), as we do in WSTO.

The work proposed by Vu and his research group [16,,17], who use WSMX [20] as an execution environment, is closely related to the work reported here. Vu et al. [16, 17] propose a methodology for enabling a QoS-based SWS discovery and selection, with the application of a trust and reputation management method. Their approach yields high-quality results, even under behaviour that involves cheating. With respect to their work, the methodology we propose is less accurate in terms of service behaviour prediction. However, their algorithm is wholly founded on reputation mechanisms, and is therefore not suitable for managing policy-based trust assumptions. Currently, policy-based trust mainly considers access control decisions via digital credentials. Our framework, by enabling participants to declare general ontological statements for guarantees and requirements, is also able to accommodate a policy-based trust framework.

Olmedilla et al. [11] propose a methodology for trust negotiation in SWS. They employ PeerTrust [10], a policy and trust negotiation language, for establishing if trust exists between a service requester and provider. The main aspect, which distinguishes their methodology from ours, is that they assume that trust is solely based on policy. They do not propose any mechanism for managing reputation or monitoring past service behaviour, as we do. Similar to our approach, they use WSMO as the underlying epistemology. Moreover, they assume delegation to a centralized trust matchmaker, where the participants disclose policies. Similarly, in our approach, we assume that IRS-III plays the role of trust matchmaker. Furthermore, they also address negotiation, which is an important issue in SWS interaction. We do not propose a formal negotiation mechanism here, but, as both requester and provider disclose their guarantees as credentials within IRS-III, we are able to automatically enable an implicit negotiation process.

There are other approaches for managing trust in SWS which are less closely related to ours such as KAoS [15]. Within KAoS a set of platform-independent services that enable the definition of policies ensuring the adequate predictability and controllability of both agents and traditional distributed systems is proposed. Even though they present a dynamic framework, and recognize trust management as a challenge for policy management, the framework is not specifically tailored to trust management in SWS.

7 Conclusions and Future Work

In this paper, we have presented a formal model for managing trust in SWS and have envisaged Web service selection and invocation as a classification problem, where the solution takes the form of a class of Web services matching participating trust profiles. Embodied within the trust profiles are the participant priorities with respect to trust, which can be related to reputation, credentials, or actual monitored behavior. Our definition of trust is described through a binary measure: whenever participant trust profiles match, a trusted interaction can occur, otherwise trusted interaction is deemed to not being feasible. We have adopted WSMO as underlying epistemology for WS description, and used IRS-III as an execution environment.

Trust has different meanings within different contexts: trust can be based on service ability or on reliability. In other contexts, trust can be related to reputation or delegated to TTP evaluations. The main contribution of our approach is to provide a framework that enables a comprehensive range of trust models to be captured. In fact, the framework can easily capture the multiple trust parameters that characterize a specific scenario, by simply specializing the WSTO reference ontology. Future work will extend our implementation to incorporate a comprehensive management suite for WS reputation and reliability. Additionally, we also plan to import a range of sophisticated reputation algorithms, and to improve the monitoring component.

References

1. Anderson, S., et al. (2004). Web Services Trust Language (WS-Trust), version 1.1. May 2004. http://msdn.microsoft.com/ws/2004/04/ws-trust/.
2. Domingue, J., Cabral, L., Galizia, S., Tanasescu, V., Gugliotta, A., Norton, B. and Pedrinaci, C. (2008). IRS-III: A Broker-based Approach to Semantic Web Services. Journal of Web Semantics, Vol. 6(2), 109-132, Publisher Elsevier 2008.
3. Fensel, D., Lausen, H., Polleres, A., De Bruijn, J., Stollberg, M., Roman, D., Domingue, J. (2006). Enabling Semantic Web Services: Web Service Modeling Ontology. Springer, 2006.
4. Fensel, D., Motta, E., Benjamins, V. R., Decker, S., Gaspari, M., Groenboom, R., Grosso, W., Musen, M., Plaza E., Schreiber, G., Studer, R. and Wielinga, B. (1999). The Unified Problem-solving Method Development Language UPML. IBROW3 Project (IST-1999-19005) Deliverable 1.1.
5. Galizia, S. (2006). WSTO: A Classification-Based Ontology for Managing Trust in Semantic Web Services, in Proceedings of 3th International Semantic Web Conference (ESWC 2006), Budva, Montenegro, June 11-14 2006.
6. Golbeck, J. and Hendler, J. (2006). Inferring trust relationships in web-based social networks. ACM Transactions on Internet Technology, 2006.
7. Maximilien, E. M., Singh, M. P. (2004). Toward Autonomic Web Services Trust and Selection. In Proceedings of 2nd International Conference on Service Oriented Computing (ICSOC 2004), New York, November 2004.
8. Motta E. (1999). Reusable Components for Knowledge Models: Principles and Case Studies in Parametric Design Problem Solving. IOS Press.

9. Motta, E., Lu, W. (2000). A Library of Components for Classification Problem Solving. In Proceedings of PKAW 2000 - The 2000 Pacific Rim Knowledge Acquisition, Workshop, Sydney, Australia, December 11-13, 2000.
10. Nejdl, W., Olmedilla, D. Winslett, M.(2004). PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web. Workshop on Secure Data Management in a Connected World (SDM'04), in conjunction with 30th International Conference on Very Large Data Bases, Aug.-Sep. 2004, Toronto, Canada.
11. Olmedilla, D., Lara, R., Polleres, A., Lausen, H. (2004). Trust Negotiation for Semantic Web Services. 1st International Workshop on Semantic Web Services and Web Process Composition in conjunction with the 2004 IEEE International Conference on Web Services, July, 2004, San Diego, California, USA.
12. OWL-S working group. (2006). OWL-S: Semantic Markup for Web Services. OWL-S 1.2 Pre-Release. (Available at http://www.ai.sri.com/daml/services/owl-s/1.2/).
13. Pedrinaci, C., Lambert, D., Wetzstein, B., Lessen, T., Cekov, L., and Dimitrov, M. (2008) SENTINEL: A Semantic Business Process Monitoring Tool, Workshop: Ontology-supported Business Intelligence (OBI2008) at 7th International Semantic Web Conference (ISWC2008), Karlsruhe, Germany.
14. Stefik M. (1995). Introduction to Knowledge Systems. Morgan Kaufmann, San Francisco, CA, USA.
15. Uszok, A., Bradshaw, J. M., Johnson, M., Jeffers, R., Tate, A. Dalton, J., Aitken, J. S. (2004). KAoS Policy Management for Semantic Web Services. IEEE Intelligent Systems 19(4): 32-41 (2004).
16. Vu, L. H., Hauswirth, M, Aberer, K. (2005). QoS-based Service Selection and Ranking with Trust and Reputation Management, 13th International Conference on Cooperative Information Systems (CoopIS 2005), Agia Napa, Cyprus, Oct 31 - Nov 4, 2005.
17. Vu L., H., Hauswirth, M., Porto, F. Aberer, K. (2006). A Search Engine for QoS-enabled Discovery of Semantic Web Services. International Journal of Business Process Integration and Management (IJBPIM), 2006.
18. W3C (2004). Web Services Architecture. W3C Working Draft 11 February 2004 (Available at http://www.w3.org/TR/ws-arch/).
19. Wielinga, B. J., Akkermans, J.K. and Schreiber, G. (1998). A Competence Theory Approach to Problem Solving Method Construction, International Journal of Human-Computer Studies, 49, pp. 315-338.
20. WSMX working group. (2005). Overview and Scope of WSMX, http://www.wsmo.org/TR/d13/d13.0/v0.2/.
21. Zhengping, W., and Weaver, A. C. (2006). Using Web Service Enhancements to Bridge Business Trust Relationships, Fourth International Conference on Privacy, Security, and Trust (PST'06), University of Toronto, Institute of Technology, Markham, Ontario, Canada, October 30-November 1, 2006.