Semantic Web Service Composition in Social Environments

Ugur Kuter[1] and Jennifer Golbeck[2]

[1] Institute of Advanced Computer Studies,
University of Maryland,
College Park, Maryland 20742, USA
[2] College of Information Studies
University of Maryland,
College Park, Maryland 20742, USA

Abstract. This paper describes how to generate compositions of semantic Web services using social trust information from user ratings of the services. We present a taxonomy of features, such as interoperability, availability, privacy, security, and others. We describe a way to compute social trust in OWL-S style semantic Web services. Our formalism exploits the users' ratings of the services and execution characteristics of those services. We describe our service-composition algorithm, called Trusty, that is based on this formalism. We discuss the formal properties of Trusty and our implementation of the algorithm. We present our experiments in which we compared Trusty with SHOP2, a well-known AI planning algorithm that has been successfully used for OWL-S style service composition. Our results demonstrate that Trusty generates more trustworthy compositions than SHOP2.

1 Motivations

Web services are finding life in new forms, which is forecasting the future of what the web service environment will look like.

Apps for the iPhone and iPod touch are essentially web services with a wrapper to make them run on the iPhone. The Apple App Store provides a model of how web services are likely to evolve if they are to remain relevant and useful. There is a demand for these apps - there are roughly 20,000 apps in the store - and even simple services in app form receive thousands of downloads. As web services are integrated into forms like apps where there is user demand, we expect to see the same evolution with thousands of services available, many with the same functionality and semantic structure (inputs and outputs). As this becomes the web service environment, a mechanism for choosing from many similar services will be important for creating service compositions that meet the user's preferences and that the user trusts.

There are many apps offering classic web service functionality - finding weather conditions and forecasts, currency conversion, stock quotes, zip code lookup, and others. Consider weather forecasts; there are at least a dozen apps that provide current conditions for a given location. How does a user decide which app (or web service) to use if

A. Bernstein et al. (Eds.): ISWC 2009, LNCS 5823, pp. 344–358, 2009.

the functionality is identical? There are many attributes to consider. Cost is one; some apps are free, others charge a small fee. The quality of the information is another consideration; some weather services are more accurate or current than others. The service provider is also an option. A user may trust the National Weather Service as a provider because they have no commercial interest in the user's personal information while a corporate entity may store the user's personal data for other purposes. Other users may be more inclined to trust the way companies will handle their data more than the way the government would. Ratings of the app provided by other users are also very useful in making a decision about which to choose.

Artificial Intelligence (AI) planning techniques have been successfully used to decide which Web services to use and how to compose them in order to achieve a functionality on the Web. In this approach, a service composition is a ground sequence of service invocations that accomplishes a goal or task. A planning algorithm takes as an input a formal description of the Semantic Web service composition problem and generates a sequence of actions (a.k.a., a *plan* or a *composition*) that achieves the goals specified in the problem description.

Existing automated AI planning techniques have been focused on a purely *functional automation* of the service-composition process in great detail, and there have been great strides in this direction [1, 2, 3, 4, 5]. However, we believe that the next-generation of Web services will possibly have the following characteristics, which have not been entirely addressed in the existing Web Service Composition (WSC) research:

- One of the most significant aspects of planning and service composition on the Web is the "human factor;" i.e., the users who are exploiting the services and doing compositions on the Web to achieve some objectives. Although it makes sense to develop automated systems in order to perform service composition to achieve a functionality on the Web, in many situations it is important to appreciate the input of the human users who are initiating the composition process and generate reliable compositions that the user will be comfortable with.

  For example, the users may have concerns and preferences about the quality of a service and may not rate that service as reliable and trustworthy with respect to their objectives. The security and privacy policies of services may also be critically important to the users, even if they do not affect the direct outcome. The organization offering the service may also be of concern to users who want do not want to share their personal information with certain entities.
- Existing WSC planning algorithms usually make the assumption that the inputs/outputs/preconditions/results of a service process model are known; otherwise, the composition algorithm would not be able to do the chaining of services in order to generate a composition. In reality, however, it is not usually possible to know whether the execution of the service will follow exactly what is stated in the description/process model of that service. Web services do not always produce the outcomes as specified in their process models. This uncertainty makes the execution of the service unreliable for the user.

  For example, when a user is trying to finalize a travel itinerary something using a travel-agency Web service, the user may not get the desired itinerary due to

a database problem at the end or the service may request the user to recreate it due to a communication problem between the provider of the service and the user. All these issues are natural, but makes the service unreliable in the eyes of the user. Then the question that a user will face is, given prior information about the Web services (e.g., his/her experience with it as well as other users' ratings of the service), should the user use that service?

2 Overview

In this paper, we describe a way to address the above aspects of Web service composition by using users' ratings about their experiences with the service and exploit this information to compute our "trust" in the service. A composition algorithm can then decide whether to execute the service based on that trust value.

Our contributions are as follows:

- We outline a taxonomy of features of Web services. This taxonomy specifies service characteristics such as privacy, security, usability/availability, reliability, and others. Most of the features in this taxonomy are already discussed in distributed systems and computing, multi-agent systems, and similar research areas, and our taxonomy is not complete by any means. But, to the best of our knowledge, these features have not been considered for the purposes of semantic Web service composition. While this feature taxonomy may change from application to application, it is meant to provide a general example here.
- We describe a formal treatment of how to take into account users' previous history of service ratings. This formalism is based on our taxonomy of service features mentioned above. The formalism allows us to develop ways of computing a user's trust in service composition process and we present such way of reasoning with trust in composing semantic Web services.
- We describe a new WSC planning algorithm, called Trusty. In Trusty, We focus on semantic Web services that are modeled in the prominent OWL-S service markup language [6]. Trusty generalizes the Hierarchical Task-Network (HTN) planner SHOP2, a well-known AI planning algorithm that has been successfully used for Web service composition [1], by incorporating reasoning mechanisms for social trust. This new WSC procedure uses the trust information on the services in order to infer the trustworthiness of a composition (or a partial composition), and returns the most trustworthy composition as a solution to a service-composition problem.
- We define three trust-computation strategies for Trusty; namely, overly-cautious, overly-optimistic, and average. These strategies differ in the way they compute the trust in a composite OWL-S service given the trust values of its sub-services. An overly-cautious trust computation produces compositions that maximize the minimum possible trust in the services. An overly-optimistic computation, on the other hand, maximizes the maximum possible trust in the services.
- We present theorems showing the correctness properties of Trusty.
- In our experimental evaluation, we compared Trusty using the three trust-computation strategies with SHOP2 in the well-known WSC domain of scheduling

doctor appointments for patients [1, 2]. Our statistical analyses of the results of the experiments, using ANOVA and t-tests as statistical methods, showed that overly-optimistic strategy enables Trusty to generate statistically more trustworthy compositions than SHOP2. The analyses also showed that overly-cautious strategy results in statistically worse trustworthy compositions compared to those generated by SHOP2. Although this might seem a bad result at first, note that it actually shows that the compositions generated by the overly-cautious strategy are the most trustworthy ones under the assumption that the worst will happen during the execution of those services. Consequently, both of these strategies, taken together, provide the most information for the human user doing the composition.

3 A Feature Taxonomy for the Behavioral Characteristics of Web Services

We compiled a feature taxonomy based on some known behavioral characteristics of Web Services. These characteristics describe the social aspects of a service that a user (human or machine alike) will consider when deciding whether to include the service in the composition being generated.While this feature taxonomy may change from application to application, it is meant to provide a general example here.

The following is a description of our feature taxonomy for WSC:

Interoperability. Web services provide a way for seamless communication, data and functional capability exchange between software applications and themselves. Service interoperability requires the following features to be implemented and handled properly by a service system: service management, data and meta-data handling, service descriptions, messaging, and service usability/availability.

Service management involves techniques for implementing how a service communicate and exchange data with other services on the Web.

Web services were originally designed to be *stateless*; i.e., a service is a functionality that performs a computation given some inputs and returns the outputs of that computation. However, many Web services use a database or a data storage/management backend in order to have a memory of their computations and operations and the results of those operations. *Data Handling* involves managing this backend system of a service by the service itself.

The *description* of a service also plays an important role in the interoperability of the service; the users of a service must know what the service does, what input/output requirements it has, and what conditions need to be satisfied in order to execute the service. If these are not known or the service does operate according to its description then the user may consider it as unreliable to include in a composition.

The *availability* of a service is also important for a user considering to use it. A service that publishes itself for the achievement of a functionality but that is never available (e.g., the server that runs the service is down or has limited processing power most of the time) is of no use to the user.

Privacy. Privacy concerns exist wherever personally identifiable information is collected and stored - in digital form or otherwise. In the context of Web services,

the service is expected to implement a privacy policy in order to control the collection and dissemination of any personal information from the users. Thus, such privacy policies implemented by Web services are usually an important piece in the user's decision process of which service to use and whether he/she should execute it.

Security. The security of Web services have increasingly become an important issue. Security related features include the authentication of the service, its credentials, and whether a communication protocol (e.g., SSL) used by the service. *Authentication* and *credentials* describe how reliable the service is with respect to its provider and its description (see above for a discussion of service descriptions as an interoperability issue).

The communication protocol that a service uses is important for some users, depending on the objective of the service. For example, a user might look for secure connections from a Web service that performs banking transactions; on the other hand, a Web service that displays the current weather conditions at a particular location does not necessarily need a secure connection to its server.

Computation. Computational issues arise in Web services because a service usually operates on a remote server (or on a cluster of remote servers) on the Web and the user does not necessarily have access or control on that computational resource.

Thus, it is important for a service provider to ensure some measure of *computational reliability* of the service to the requesters of that service. Computational reliability of a service involves a good and honest description of what the service does (i.e., the functional definition of the service from its inputs to outputs), what it does not do, and what it may do when executed. This last issue arises if the description of the service and the outcome of its execution do not match to each other. In such cases, the user may not achieve his/her goals in using the service.

Another feature of a Web service is its *complexity* – i.e., its *functional granularity*. Web services are usually designed to be simple, achieving basic functionalities on the Web and designed to be composed into more complex functionalities via service-composition algorithms. A Web service that implements many functionalities in a very fine-grained way may or may not be attractive to the users depending their preferences and objectives.

The taxonomy and the features mentioned above are not complete by any means, but include the most typical examples of such behavioral features of Web services that influence users' decision on whether and how to use those services. Depending on the particular composition problem, this taxonomy can be expanded to include some domain-specific features as well. For example, a shopping service may require a licensing agreement with its provider and the user. Depending on the terms of that requirement, the user may or may not choose to use that service (e.g., if the terms are very restrictive, then the user may choose to use another service that has less restrictive requirements).

4 Social Trust

Users can determine the trust they have for a service by considering the features of the taxonomy above and their experience with a service. However, when a user has no

experience with the service, it can be difficult to judge how trustworthy the service is. Information from other users who have interacted with the service can be used to help make these decisions. This is parallel to considering the ratings and reviews of apps for the iPhone. However, rating systems can be easily manipulated. For example, a malicious user can build a service that steals credit card numbers and then create hundreds of accounts to assign high ratings this service. To circumvent this, we consider *trust* between users along with ratings to help find trustworthy services for the composition.

4.1 Computing Trust between Users

There are several methods by which we can compute the trust that user c has in user u. These fall into two broad categories: trust can be computed using a social network or approximated through the ratings they have provided. Which approach to use, or if a combination of the two methods is preferable, will depend on the data available and the specific application.

When a social network is available, any trust-inference algorithm such as TidalTrust [7], advogato [8], Appleseed [9] , moletrust [10], and SUNNY [11] can be used to compute $\mathsf{trust}(c, u)$. In the Web service composition spaces, typically services are not associated with social networks but the data is available to approximate trust using user ratings. For example, in the Apple AppStore, each app/service is rated by many users and those ratings are publicly available for a user considering to use that app/service. Thus, we are going to focus here how to compute trust via user ratings.

Trust can also be computed from nuanced similarity measures over ratings users add into a system. In this context, the ratings would be on the web services. Research has shown that there is a strong correlation between trust and overall similarity [12, 13]. Further work indicates that trust is more complex than overall user-user similarity, and shows that trust can be estimated by a combination of more nuanced similarity measures [14]. That research identified four measures made over users' item ratings that relate to trust: overall similarity, similarity on extremes , the single largest difference, and the source's propensity to trust. We can compute similarity measures in two ways: as mean average error (MAE) and using the Pearson correlation. Thus, we had six total measures: the average difference (AD), overall correlation (COR), average difference on extremes (XD), correlation on extremes (XCOR), the single largest difference (MD), and the source's propensity to trust (PT). A linear combination of these values can predict trust is given in the following equation:

$$\begin{aligned}\mathsf{trust}(source, sink) = \theta_{AD} * AD + \theta_{COR} * COR + \theta_{XD} * XD + \\ \theta_{XCOR} * XCOR + \theta_{MD} * MD + \theta_{PT} * PT\end{aligned}$$

When the weights are estimated using multivariate linear regression, this equation generates a trust estimate accurate to within approximately 10%. This is as good or better than most trust inference algorithms that run on a social network as described above.

4.2 Semantic Web Services, User Ratings and Trust

We assume the existence of a finite set W of Web services that are given. Let F be a finite set of all of the available taxonomical features related to the services in W. A

user's *service-composition rating* (or *rating*, for short) is partial function that models a user's rating of a service w:

$$\rho : W \times F \rightarrow [0, 1].$$

Intuitively, a user rating specifies a subjective value of a Web service w with respect to a behavioral feature f. For example, let f be the privacy policy of the service w. A low $\rho(w, f)$ value would mean that the user does not agree with the terms of the privacy requirements of the service, whereas a high $\rho(w, f)$ value would point to a more positive rating.

In most systems using social trust, a single value is computed that indicates the trust user c has for a user u. However, in the case of Web service composition, we have a set of features F of services and we can potentially compute the trust between c and u on feature f. Some methods for inferring trust between users will not have the data available to compute the values on each feature. In those cases, we will use the single trust value as the trust for all features. For example, if an algorithm computes that the user c should trust the user u at a level of 0.8 on a $[0, 1]$ scale, we would use 0.8 as the trust value for each feature $f \in F$.

As mentioned before, we focus on the well-known semantic Web services formalism OWL-S [6] to model Web services. OWL-S differentiates between *atomic* and *composite* services. The former can be directly executed in the world, whereas the latter must be decomposed into atomic ones for execution. Other formalisms model only atomic services. Below, we describe our definitions for trust to encapsulate both service formalisms.

Suppose $\mathcal{R}$ is a finite set of user ratings and let w be a Web service. Let U_ρ be the set of all users who rated the service w. In other words, for each user $u \in U_\rho$, there exists a user rating $\rho_u \in \mathcal{R}$ such that $\rho_u(w, f)$ is defined for some feature f. Finally, let c denote the user who is performing the Web service composition and $F_c(w)$ be the set of features the user c is interested in the service w.

We define the trust value $t_c(w, f)$ that the composer user c has in a Web service w based on the feature f as follows:

$$t_c(w, f) = \sum_{u \in U_\rho} \rho_u(w, f) \times \mathsf{trust}(c, u),$$

where $\mathsf{trust}(c, u)$ is the trust that c has in the user u, as described in the previous section.

Then, the expected trust that c has in the service w is:

$$t_c(w) = \frac{\sum_{f \in F_c(w)} t_c(w, f)}{|F_c(w)|}.$$

If there is a composite Web service w in W then the idea is to generate a composition (i.e., a sequence of atomic services) for the composite service and propagate the trust values of the atomic services in the composition upwards, starting from smaller services to the composed ones and eventually to w.

Let M be a set of process models for w and for each process model $m \in M$, let $\mathsf{sub}(m, w)$ denote the set of services that are sub-processes of w in m. We devise three different strategies that define the trust value $t(w)$ of w. These are as follows:

Overly-Cautious. The overly-cautious strategy for defining the trust in a service w aims to maximize the minimum expected trust value that the composer user has in the sub-processes of w. In other words, this strategy implements the well-known *minimax* game-tree evaluation criteria [15], adapted for social trust in Web service composition. Formally, the trust value in a service is

$$t(w) = max_{m \in M} Q(m)$$
$$Q(m) = \min_{w' \in \mathsf{sub}(m,w)} t(w')$$

Overly-Optimistic. This strategy can be seen as an opposite of the overly-cautious strategy above. The overly-cautious strategy assumes that if something bad could happen in the composed services (i.e., if the least trusted service fails our trust as expected), it would definitely happen. On the other hand, the overly-optimistic strategy assumes the opposite: nothing bad will happen (i.e., even if we have a low trust on a service, that service will not fail the expectations from it), and therefore, we can evaluate a composition based on the user's maximum trust on the sub-processes of a service. More formally,

$$t(w) = max_{m \in M} Q(m)$$
$$Q(m) = \max_{w' \in \mathsf{sub}(m,w)} t(w')$$

Average. Finally, the Average strategy computes the trust value in a process model m for w by taking the average of the trust computed for each of the sub-processes of w in m:

$$t(w) = max_{m \in M} Q(m)$$
$$Q(m) = \frac{\sum_{w' \in \mathsf{sub}(m,w)} t(w')}{|\mathsf{sub}(m,w)|}$$

Above $|\mathsf{sub}(m,w)|$ denotes the number of services in $\mathsf{sub}(m,w)$.

5 Web Service Composition with Social Trust

We start with our definitions for trust-based service composition. We define a *trust-based service composition problem* as a tuple $P = (S, c, G, W, F, \mathcal{R})$, where S is the initial state, c is the user that performs the composition, G is the OWL-S process model to be achieved, W is the available set of OWL-S services, F is the set of all features in the problem, and $\mathcal{R}$ is a finite set of user ratings.

A *solution* for P is a composition that maximizes the trust value of achieving G.

Figure 1 shows a high-level description of our Web service-composition algorithm, Trusty. Trusty is inspired by HTN planning algorithms for WSC, in particular, from the previous work on SHOP2 [1]. Unlike SHOP2, Trusty incorporates the notion of social trust and several trust computation strategies to reason with trust in order to generate trustworthy compositions.

Procedure Trusty$(s, c, G, W, F, \mathcal{R}, \pi)$
1. if $G = \emptyset$ then **return**(π)
2. nondeterministically choose a subgoal w from G that has no predecessors and remove w from G
3. if w can be achieved in s by an atomic service then
4. if there is no such process model in W for w then
5. **return**$(failure)$
6. insert the process model for w into π
7. $\tau \leftarrow 0$
8. for each feature $f \in F$ do
9. for each user u who has a rating for w do
10. $\tau \leftarrow \tau + \rho_u(w, f) \times \text{trust}(c, u)$
11. $\Theta(w) \leftarrow \frac{t}{|\mathcal{R}||F|}$
12. $\pi \leftarrow$ Trusty$(s', c, G, W, F, \mathcal{R}, \pi)$
13. else
14. $\mathcal{D} \leftarrow \{d \mid m$ is a process model for w in s and d is a composition generated by applying m to w in $s\}$
15. $\pi_{max} \leftarrow \emptyset$
16. for each composition $d \in \mathcal{D}$ do
17. $G' \leftarrow$ UpdateComposition(G, s, t, m, d)
18. $\pi \leftarrow$ Trusty$(s, c, G', W, F, \mathcal{R}, \pi)$
19. if $\pi = failure$ then **return**$(failure)$
20. $Q(d) \leftarrow$ ComputeTrust(d, Θ)
21. if $Q(d) > \Theta(w)$ then $\pi_{max} \leftarrow \pi$; $\tau \leftarrow Q(d)$
22. $\Theta(w) \leftarrow \tau$
23. **return**(π)

Fig. 1. A high-level description of the Trusty procedure for Web service composition using trust. Above, s is the current state, c is the user who requested the composition, G are the goals of the user c, W is the set of available services, F is the set of features of those services, $\mathcal{R}$ is the set of user ratings, π is the (partial) solution composition. Initially s is the initial state, π is the empty composition, and β is 0. Θ is a global table that holds all of the trust information over the services that is computed by the algorithm.

The input to Trusty consists of a service composition problem with social trust $P = (s, c, G, W, F, \mathcal{R})$ and the empty plan π. We define Θ, a table that holds the trust-values on services. We assume that Θ is accessible globally by the Trusty procedure and initially, it specifies a trust value of 0 for each service in W.

With this input, Trusty starts from the goal G and recursively generates compositions for G and its subgoals, until a solution composition is generated or a $failure$ is returned during the problem-solving process.

At each invocation of the algorithm, Trusty first checks whether the goal functionality is achieved or not (i.e., whether $G = \emptyset$ or not). If the goal is empty, then Trusty returns the current composition π in Line 1. Otherwise, the algorithm nondeterministically chooses a subgoal w from G that has no predecessors in G. If w can be achieved by an atomic service, and there is a service process model defined for w

in W, then Trusty inserts that process model into π since when executed, that service process will achieve the goal w.

Next, Trusty computes the trust the current user c has that the service, when executed, will conform to c's composition profile (Lines 7–10). The algorithm then updates the trust-values table with the newly-computed trust value for the service for w (Line 11). It continues with the remainder of the goals to be accomplished in G via a recursive call to itself at Line 12.

If the subgoal w chosen at Line 2 is a nonprimitive functionality, then Trusty generates the set D of all possible compositions that might achieve w. At Line 14, the set D of compositions are generated via the available methods for w that are applicable in the current state of the world.

Then, for each possible composition for w, Trusty performs a series of steps as follows. Trusty first generates the next goal to achieve given the goal functionalities in the current composition for w and the remaining goals in G. The UpdateComposition function is responsible for this operation. UpdateComposition is a standard HTN task decomposition mechanism described in [1, 2], so we are not going into the details of this subroutine in this paper.

Then, Trusty calls itself recursively to achieve the new goal G'. When Trusty returns, there are two bits of knowledge that the algorithm needs to process. One is the partial composition π returned by the call. If π is a failure, than means that the current composition cannot generate a solution for w; in that case, Trusty ignores this composition and continues with the next one (if any).

If π is not failure when Trusty returned, then the trust table Θ must contain a value for the trust of c computed for each sub-service for w. The Q value of the current composition d is then computed by applying a trust computation strategy. As we described in the previous section, we developed three strategies for this purpose: namely, overly-cautious, overly-optimistic, and average. In Line 20, the ComputeTrust subroutine is responsible for implementing one of these strategies as specified by the user.

If the Q value of the decomposition d is greater than the current α value, then Trusty returns updates the trust value for the service w since it found a partial composition that has a higher value trustworthiness than the previously best one. It also keeps track of the current partial composition by marking as the current best solution candidate (see Line 21).

After Trusty processes and computes the best composition for w, it updates the trust table Θ with the trust value of that composition (Line 22) and returns that composition from the current invocation (Line 23).

The following theorems establish the soundness and completeness of Trusty:

Theorem 1. *Let $P = (s, c, G, W, F, \mathcal{R})$ be a social trust-based composition problem. Then, if Trusty returns a composition π for this problem P, then π is a solution of P.*

Note that there are two aspects of a solution π for P. First, π must achieve the functionality G specified in P. Second, if there is another composition π' then the user c's trust in π' must be lower or equal that in π. Trusty guarantees the first aspect given the set of services in the input W since we assume that for each goal that can be achieved by an atomic service, there is one corresponding process model in W, and for each goal that can be achieved by a composite service, there is one process model in W for that

goal and for each of its subgoals. Then, the proof follows by induction on the number of decompositions of the G that needs to make until we generate a sequence of atomic services that, when executed, accomplishes G.

The second aspect follows from the bounding condition of Line 21 of the pseudo-code shown in Figure 1.

Theorem 2. *Let $P = (s, c, G, W, F, \mathcal{R})$ be a social trust-based composition problem. Then, if there exists at least one solution composition π for P then* Trusty *returns π.*

If there is no solution to P, then Trusty *returns $failure$.*

If there is a solution π for the service-composition problem P, then Trusty will eventually generate that solution since (1) it searches for every possible composition for the goal G, and (2) it prunes a composition only if that composition induces an inferior trust value compared to an alternative one. If there is no solution to P, then Trusty will try every possible composition (with possible prunings), and return failure in one of its recursive invocations.

The next section describes our implementation of Trusty and the experiments we ran with our implementation.

5.1 Discussion

Investigating the uncertainty due to the interactions between the users and their ratings is extremely interesting and realistic; however, reasoning about such uncertainty is hard and complex too. In fact, it indeed requires a different model to compute incremental inferences over social trust correctly, where a newer service with few or no ratings (and hence low trust) arrives and it is to end up in a solution composition, given the fact that there are many services that already have higher trust.

We to stay away from that uncertainty in our current model by assuming that each user and his/her rating is independent of another. This assumption allows us to adapt a typical assumption with most service-composition research that all the services (with their ratings in our case) are available up front and there are no changes to the service set during composition. Certainly, here are cases in which this assumption is not realistic; however this assumption simplified the initial model presented in this paper and allowed us to gain insights.

6 Experiments

We implemented a prototype of the Trusty algorithm using the code base of the SHOP2 planning system [16, 1], which was previously use for WSC quite successfully. We extended this planning system in order to implement our trust reasoning mechanisms and our optimizing search search algorithm to extend SHOP2's usual depth-first search implementation.[1] The particular implementation of optimization in Trusty depends on the strategies which one of the overly-cautious, overly-optimistic, or average trust computation is used.

[1] The SHOP2 planning system also implements a form of branch-and-bound search for simple optimization planning problems but it is designed for reasoning with action costs, not with trust values of both atomic and composite compositions.

Table 1. Trust values of the solution composition found by Trusty and SHOP2. In Trusty, we used three trust-inference strategies as described in text: Average, Overly-Cautious, and Overly-Optimistic. Each data point is an average of 50 random service-composition problems.

Algorithm	Trust-Inference Strategy	Average Trust
Trusty	Average	0.0043
Trusty	Overly-Cautious	0.0016
Trusty	Overly-Optimistic	0.0097
SHOP2	None (Depth-First Search)	0.0039

Our experimental scenario was based on the well-known WSC domain on *scheduling* of doctor-patient appointments. This scenario was first described in the Scientific American article about the Semantic Web [17]. In this domain, two people are trying to take their mother to a physician for a series of treatments and follow-up meetings. The service-composition problem is to come up with a sequence of appointments that will fit in to everyone's schedules, and at the same time, to satisfy everybody's preferences.

Since our system requires a mechanism for computing trust, we had to artificially generate realistic underlying data that could be used to compute trust. To do this, we used the social network and user ratings from FilmTrust [18], a social network where users rated their trust for one another and also rated movies. We kept the same social network and trust ratings between users, and randomly mapped users' ratings of movies to ratings of services for doing doctor-patient appointment assignments. This provided a realistic test dataset that represented user's distribution of preferences and their social relationships.

We generated 50 random mappings (i.e., random user ratings and trust values among users), and ran both SHOP2 and Trusty in this scenario with those ratings. We used the three trust-computation strategies in Trusty. In all of these runs, Trusty was able to generate compositions that have higher trust values than those generated by SHOP2, as summarized in Table 1.

Using the Overly-Optimistic trust-inference strategy, Trusty the average trust value of the solutions generated by SHOP2 was 0.0039, whereas that of the solutions generated by Trusty was 0.0097. The average trust value of the composition generated by Trusty's Average trust-inference strategy was 0.0016. An ANOVA shows statistically significant differences among the population, and pairwise t-tests indicate that the value of the Overly-Optimistic strategy statistically significantly higher than the Overly-Cautious strategy for $p < 0.01$.

In fact, our analyses showed that Trusty's Overly-Cautious strategy caused the algorithm to generate compositions whose average trust value is statistically worse than the average trust values generated by SHOP2. Although this result may suggest that this strategy is not good for service composition, this is not necessarily true. The objective of this strategy is to be conservative – i.e., the objective is to maximize the minimum possible trust in the services so that if a service fails fails to meet the composer's expectations, the ramifications of such a failure is minimal. The overly-cautious strategy just ensures this criterion while guaranteeing to generate service compositions whenever there are any for a composition problem.

Consequently, both of these strategies, taken together, provide the most information for the human user doing the composition.

We also ran a second set of experiments with Trusty using the average trust-computation strategy in order to determine the range of the trust values generated by the two planning systems. For these experiments, we used the same 50 randomly-generated mappings as above. In order to find the upper bound of the range of trust values, we set each trust value that appears in the user profiles to 1, the maximum possible trust value in our formalism. For the lower bound, we set each trust value that appears in the user profiles to 0.01, in order to estimate the minimum possible trust value in our formalism. We did not use 0's for the minimum trust values since then the compositions returned by both of the planners would always have 0 values.

When we set all of the trust values in the user profiles to 1, the average trust value of the solutions generated by SHOP2 in this setting was 0.0059, whereas that of the solutions generated by Trusty was 0.0065. When we set all of the trust values in the user profiles to 0.01, the average trust value of the solutions generated by SHOP2 in this setting was 0.000059, whereas that of the solutions generated by Trusty was 0.000065. In both cases, as above, Trusty's solution compositions had higher trust values than those of SHOP2.

The results with the maximum trust values in the profiles also show that SHOP2 cannot generate the best solution (i.e., the solution with maximum possible trust values) in our scenarios; the best composition it can generate has a trust value lower than that is generated by Trusty.

7 Related Work

Existing approaches for Web Service Composition formulate the problem in different ways, depending mainly on how the developers of those approaches perceive the problem. However, as we mentioned at the beginning of the paper, an important commonality among these approaches is that they focused on a purely *functional automation* of the service-composition process without any consideration for social trust, as we studied in this paper.

One of the first techniques developed for WSC is reported in [3]. Here, the states of the world and the world-altering actions are modeled as Golog programs, and the information-providing services are modeled as external functions calls made within those programs. The goal is stated as a Prolog-like query and the answer to that query is a sequence of world-altering actions that achieves the goal when executed in the initial state of the world. During the composition process, however, it is assumed that no world-altering services are executed. Instead, their effects are simulated in order to keep track of the state transitions that will occur when they are actually executed.

In [1], the WSC procedure is based on the relationship between the OWL-S process ontology [6] used for describing Web services and *Hierarchical Task Networks* as in HTN Planning [16]. OWL-S processes are translated into tasks to be achieved by the SHOP2 planner [16], and SHOP2 generates a collection of atomic process instances that achieves the desired functionality.

[19] extends the work in [1] to cope better with the fact that information-providing Web services may not return the needed information immediately when they are

executed, or at all. The ENQUIRER algorithm presented in this work does not cease the search process while waiting answers to some of its queries, but keeps searching for alternative compositions that do not depend on answering those specific queries.

[20] models Web services and information about the world using the "knowledge-level formulation" first introduced in the PKS planning system [21]. This formulation models Web services based not on what is actually true or false about them, but what the agent that performs the composition actually knows to be true or false about their operations and the results of those operations. A composition is formulated as a conditional plan, which allows for interleaving the executions of information-providing and world-altering services, unlike the work described above.

In [4] and [5], a planning technique based on the "Planning as Model Checking" paradigm is described for the automated composition of Web services. The BPEL4WS process models was first translated into state transition systems that describe the dynamic interactions with external services. Given these state-transition systems, the planning algorithm, using symbolic model checking techniques, returns an executable process rather than a linear sequence of actions.

8 Conclusions and Future Work

In this paper, we presented a new formalism for composing Web services when user ratings are present and Trusty, a new service-composition algorithm for creating trustworthy Web service compositions. This incorporates user preferences as considerations when generating compositions. The social trust component also allows users to benefit from the experiences and ratings of others, thus providing some information on the trustworthiness of unknown services. This, in turn, can be used as a mechanism for making choices when generating Web service compositions, resulting in selections and compositions that are personalized and more trustworthy from the user's perspective.

More extensive evaluation is the most important next step in this work. We provided some preliminary results in one sample domain. However, a more extensive study is needed to strongly demonstrate that Trusty will generate useful trust information about service compositions in a real domain and that users will significantly prefer these compositions. This will require recruiting users who have expertise and strong preferences regarding web services and who can understand the resulting compositions. Furthermore, we will require some method of extracting trust relationships between users. This can be done by implementing or recruiting subjects from an existing social network, or by computing trust from profile similarity, which will require a dense set of service ratings. With this data, we can illustrate that our algorithm effectively improves the quality of service compositions for users.

Acknowledgments. This work was supported in part by the DARPA/Air Force Integrated Learning Program, through the contract # FA8650-06-C-7606, and by the National Science Foundation. The opinions expressed in this paper are those of authors and do not necessarily reflect the opinions of the funders.

References

[1] Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for web service composition using SHOP2. Journal of Web Semantics 1, 377–396 (2004)

[2] Kuter, U., Sirin, E., Parsia, B., Nau, D., Hendler, J.: Information gathering during planning for web service composition. Journal of Web Semantics (2005)
[3] McIlraith, S., Son, T.: Adapting Golog for composition of semantic web services. In: KR-2002, Toulouse, France (2002)
[4] Pistore, M., Barbon, F., Bertoli, P.G., Shaparau, D., Traverso, P.: Planning and monitoring web service composition. In: Bussler, C.J., Fensel, D. (eds.) AIMSA 2004. LNCS (LNAI), vol. 3192, pp. 106–115. Springer, Heidelberg (2004)
[5] Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 380–394. Springer, Heidelberg (2004)
[6] OWL Services Coalition: OWL-S: Semantic markup for web services, OWL-S White Paper (2004), http://www.daml.org/services/owl-s/1.1/owl-s.pdf
[7] Golbeck, J.: Computing and Applying Trust in Web-based Social Networks. PhD thesis, University of Maryland, College Park, MD, USA (2005)
[8] Levien, R., Aiken, A.: Attack-resistant trust metrics for public key certification. In: 7th USENIX Security Symposium, pp. 229–242 (1998)
[9] Ziegler, C.N., Lausen, G.: Spreading activation models for trust propagation. In: Proceedings of the IEEE International Conference on e-Technology, Taipei, Taiwan, IEEE Computer Society Press, Los Alamitos (2004)
[10] Avesani, P., Massa, P., Tiella, R.: Moleskiing.it: a trust-aware recommender system for ski mountaineering. International Journal for Infonomics (2005)
[11] Kuter, U., Golbeck, J.: Sunny: A new algorithm for trust inference in social networks, using probabilistic confidence models. In: Proceedings of the National Conference on Artificial Intelligence, AAAI (2007)
[12] Ziegler, C.N., Lausen, G.: Analyzing correlation between trust and user similarity in online communities. In: Proceedings of the Second International Conference on Trust Management (2004)
[13] Ziegler, C.N., Golbeck, J.: Investigating Correlations of Trust and Interest Similarity. Decision Support Services 1 (2006)
[14] Golbeck, J.: Trust and nuanced profile similarity in online social networks. In: Transactions on the Web (to appear, 2009)
[15] Chi, P., Nau, D.S.: Predicting the Performance of Minimax and Product in Game Tree Searching. In: Second Workshop on Uncertainty and Probability in Artificial Intelligence (1986)
[16] Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. JAIR 20, 379–404 (2003)
[17] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284, 34–43 (2001)
[18] Golbeck, J.: Generating predictive movie recommendations from trust in social networks. In: Proceedings of the Fourth International Conference on Trust Management, pp. 93–104 (2006)
[19] Kuter, U., Sirin, E., Nau, D.S., Parsia, B., Hendler, J.: Information gathering during planning for web service composition. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 335–349. Springer, Heidelberg (2004)
[20] Martinez, E., Lespérance, Y.: Web service composition as a planning task: Experiments using knowledge-based planning. In: ICAPS-2004 Workshop on Planning and Scheduling for Web and Grid Services (2004)
[21] Petrick, R.P.A., Bacchus, F.: A knowledge-based approach to planning with incomplete information and sensing. In: AIPS (2002)