A Framework for Building Privacy-Conscious Composite Web Services[*]

Wei Xu [*] V.N. Venkatakrishnan [†] R. Sekar [*] I.V. Ramakrishnan [*]

[*]Department of Computer Science
Stony Brook University
Stony Brook, NY 11790-4400
Email: {weixu,sekar,ram}@cs.sunysb.edu

[†]Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607
Email: venkat@cs.uic.edu

Abstract

The rapid growth of web applications has prompted increasing interest in the area of composite web services that involve several service providers. The potential for such composite web services can be realized only if consumer privacy concerns are satisfactorily addressed. In this paper, we propose a framework that addresses consumer privacy concerns in the context of highly customizable composite web services. Our approach involves service producers exchanging their terms-of-use with consumers in the form of "models". Our framework provides automated techniques for checking these models at the consumer site for compliance of consumer privacy policies. In the event of a policy violation, our framework supports automatic generation of "obligations" that the consumer generates for the composite service. These obligations are automatically enforced through a dynamic program analysis approach on the web service composition code. We illustrate our approach with the implementation of two example services.

1 Introduction

The past decade has witnessed a phenomenal growth in the number of users who routinely use the web to obtain information, conduct research, or carry out financial transactions. While the ability of the web to provide customized information and financial services has boosted personal and business productivity, it has raised significant concerns regarding consumer information privacy. Solutions such as providing an explicit verbal statement of the websites terms-of-use of the user's private data have not increased consumer confidence due to their imprecise and verbose nature.

To address consumer concerns on information privacy, web service providers have taken the steps to conspicuously identify and display their policies regarding the storage and use of personal user information. A first step in this direction is the P3P [2] (Platform for Privacy Preferences) approach, in which the website expresses its terms of use regarding privacy in a machine-readable format. The consumer checks if these terms of use match her privacy preferences via a P3P-enabled web browser and thereby is ensured that her preferences meet the website's terms of use.

The privacy concerns of end-users are exacerbated further as we progress towards a world of complex web services that involve multiple providers. Here our focus is on composite web services that provide advanced services to end-users based on simpler services provided by multiple providers. Illustrative examples of such composite web services include:

- A travel arrangement service that can, in one stop, make all travel arrangements, such as airline reservation, hotel accommodation, and local transportation arrangements.
- An electronics purchase service that integrates research, reviews, and comparison shopping into a single, easy-to-use service.
- Advanced over-the-counter (OTC) medicine procurement consulting medical histories, online databases and credit-card transactions.

Current trends indicate that the greatest potential for advanced web services lie in the context of such composite services. However, successfully realizing such advanced web services requires addressing the challenges raised in terms of consumer information privacy. In this case, personal information could be shared across the providers in ways that weren't intended by the owner of the information.

Consider the travel example given above, and let us say that there are airlines A_1 through A_{10}, hotels H_1 through H_{10} and rental car companies R_1 through R_{10} that participate in the composite service. In reality, a composite web service may be composed of several hundreds of individual web services. Consider that not all the individual web services are agreeable to the consumer's privacy preferences. In such cases, the consumer may prefer to have her personal information distributed only to some of these component web services by indicating this in her privacy preferences. For instance, she may choose to use airlines A_4, A_7, hotels H_1, H_2, H_3 and rental car agencies R_1 and R_4 out of the possible options. With the use of P3P, compliance checking involves matching the web-service's terms of use with the consumer's privacy preferences. This compliance checking

[*]This research was supported in part by Computer Associates, ONR grant N000140110967, and the Sensor CAT at Stony Brook (a NYSTAR Center for Advanced Technologies).

results in a number of conflicts ("hotels H_4 through H_{10} will receive your travel information") that are reported to the consumer. Such compliance violations do not suggest that the consumer disagrees with the composite service's privacy policy; they merely suggest that the consumer does not want to share her information with some subset of component services. (For instance, these may be the component services that the consumer has no prior transaction history.) Given the nature of highly, customizable composite web services, we claim that it is very much possible to offer *privacy-conscious* services to the consumer. Such privacy-conscious services can be accomplished by dynamically adapting the behavior of the web service to take into account the consumer privacy preferences.

In this paper, we present a framework to realize such highly customizable privacy-conscious composite services. Our approach can be summarized as follows: When a consumer provides data to a service provider, she wants to ensure that the information she provides will be used in a manner consistent with her privacy preferences. To verify if this will be the case, the consumer requests a *model* of the service. The model summarizes the manner in which the composite service uses the consumer data (e.g. how the consumer data can flow to various component services.) In our approach, we provide automated techniques using which the consumer can check this model's compliance to her privacy preferences . If the check succeeds, the consumer forwards the request to be processed by the service. Otherwise, the consumer can forward this violation as an *obligation* to the composite service, and mandates the obligation to be enforced. Our past work [10] on dynamic analysis techniques provides the basic technology for such enforcement. This analysis ensures that these obligations are respected when the code for the service is executed. Although the techniques discussed in this paper are suitable for any web service framework, we have implemented our approach in the context of WinAgent system [5]. The WinAgent system offers web services APIs by creating open wrapper services around existing web sites. Composite services are built using such unit services. Two case studies have been developed in our implementation.

We want to point out that our approach (similar to P3P) does not address the problem of malicious service providers: there can be no technological solution that provides protection from providers that intentionally lie about their usage of consumer data. Even code analysis techniques are of no avail here: these techniques can ensure that a *given piece of software* satisfies a certain policy, but in a distributed environment, end-users have no authority (or even access) to the computers hosting the service. Therefore, they cannot prevent a malicious service provider from simply switching to a different piece of software that violates these policies. As such, technology cannot prevent malicious providers that misrepresent their terms of usage, or provide incomplete information about it. Indeed, our approach relies on the premise that due to market or societal forces, providers and consumers have already decided to collaborate in order to protect consumer privacy. Our approach simply provides a technological basis to facilitate this collaboration. We therefore assume some non-technological means (e.g., legal enforcement) can address the problem of such malicious providers.

In addition, several challenges need to be addressed in the area of security in composite web services. Some of them are authentication and trust management, confidentiality of communication, privacy, integrity, reliability and quality of service. Our focus in this paper is solely on privacy aspects, and we indirectly address aspects of quality of service through better privacy guarantees.

The rest of this paper is organized as follows: Section 2 gives our framework architecture. We then discuss the individual components in the framework in the following sections: composition infrastructure in Section 3, service models in Section 4, consumer privacy policies in Section 5, policy checking, obligation generation, and obligation enforcement in Section 6. In Section 7 we present two example services that we have implemented to demonstrate our framework. In Section 8 we discuss related work.

Some ideas from an earlier stage of this work were sketched in a poster presentation [13]. The paper adds notions of obligation generation and enforcement to the basic approach and provides details on the technical underpinnings of the high level ideas presented in the poster.

2 Framework Overview

Figure 1 shows the architecture of our framework, which consists of five major components: a) service composition code, b) service models, c) privacy policies, d) policy compliance checker and obligation generation, and e) obligation enforcer.

The component services are represented by ovals in the lower right part of the figure. Each component service accepts inputs from its caller, then interacts with other entities such as web sites, and returns requested results to the caller. These services are glued together using the *composition code*, resulting in a composite service which typically provides richer functionalities. Each composite web service is associated with a *service model* that captures how the input data provided to the service can flow into various entities such as its component services.

Consumers can express their privacy concerns using *privacy policies*. These policies capture what types of consumer data they are willing to share with which service providers ("principals"). When a consumer interacts with a composite web service, she first requests the model of the service, and check whether the model is compatible with

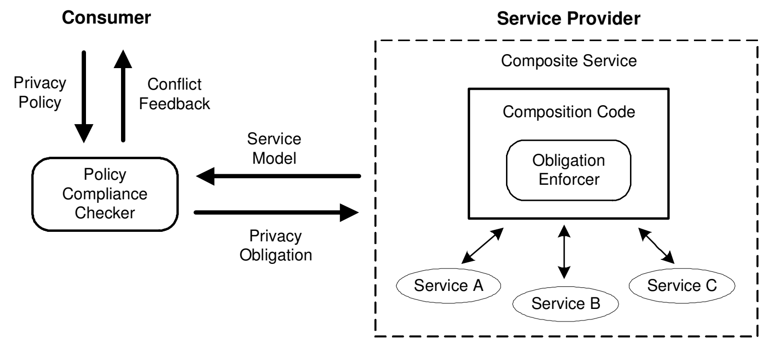

Figure 1. Framework architecture

her privacy policies. If the check succeeds, the service request is forwarded to the service provider. If the policy is violated, however, the consumer can then have two options:

- *Consumer policy refinement*. The consumer may relax her privacy policy so that the service can be used. In this case, the consumer is told the manner in which the policy is being violated, e.g., zip code is being sent to an Internet merchant through a comparison shopping service. The consumer may allow to reveal her zip code information to obtain the service.
- *Obligation generation*. The consumer may present this violation as an *obligation* and forwards it to the provider to see if the service can be provided without violating consumer policy, e.g., the comparison shopping service is informed that zip code should not be sent to any merchants. The comparison shopping site may still be able to provide most of its service, e.g. provide price quotes from those merchants that do not ask for zip code.

When the obligations are presented to the composite service, our approach ensures that these obligations are not violated during the execution of the composite service code.

To realize the above-mentioned approach, two important problems need to be addressed:

- *Policy compliance checking*. The consumer needs to be informed *a'priori* of the potential web services that may need access to the consumer's private information. The compliance checking needs to provide support (in the event of compliance checking violations) for generating obligations that will be passed on to the web service. In addition, it should provide guidance for making changes to the consumer's privacy policies if the consumer decides to relax her policies.
- *Obligation enforcement*. Once the web service is given a set of obligations that it needs to enforce, support has to be provided for meeting these obligations. This would entail tracking how the consumer's information flows to and from the composite service, and prevent any of such information flow that is not authorized by the consumer.

In our approach, compliance checking is addressed by *automated techniques* that check whether the service's model complies with the consumer's privacy policies, and generation of any violations. Such violations can be used to either guide the consumer to relax her policies, or automatically generate obligations to be satisfied by the web service. *Obligation enforcement* involves tracking the flow of information to and from the component web services, and within the composite service, as and when it happens.

Running example. To illustrate the main ideas behind our approach, we shall use a running example which will be revisited in later sections when we discuss each component of the framework.

Suppose that the composite service in Figure 1 is a travel management service, which is composed from a JetBlue airline ticket booking service (Service A), a Hotels.com hotel reservation service (Service B), and a MapQuest driving direction service (Service C). Consumers can make use the composite service as a one-stop shop for their travel planning and reservations. To use this service, a user provides her travel information (e.g. the origin/destination cities) and her payment information (e.g. the credit card number), phone contact information (e.g., emergency contact) to the composite service. The composite service will invoke Service A with the origin/destination addresses and payment and contact information to book an air ticket; then request Service B to reserve a hotel near the destination address, and execute Service C to retrieve the driving directions from the destination airport to the hotel.

To protect users' privacy in the composite service, the user needs to define her privacy policy to specify her own privacy concerns. One example privacy policy (in English) would be that "both origin/destination addresses and payment information are allowed to be sent to JetBlue and Hotels.com, but the destination address is the only piece of information (neither the billing information nor the mobile number information) that is allowed to be sent to MapQuest." Each composite service is required to provide a model that describes the manner in which consumer data is handled by component services that are invoked by this composite service. For instance, the model of the composite service with regard to the JetBlue air ticket booking service might describe itself as “origin/destination addresses and payment information are sent to JetBlue”. The model of the composite service with regard to the MapQuest system may say “destination address and mobile phone information (if present) are sent to MapQuest” (for maps sent by mobile phone). These models are then used by the policy compliance checker to verify if the services violate the consumer’s privacy policy. In this case, the consumer policy specifies that only her address information be sent to MapQuest and not her phone numbers. Here, the consumer can suggest that the “mobile phone information be used for JetBlue” (only for emergency contact information purposes) and generate an *obligation* to the composite service that mobile phone information “not be provided to MapQuest”. The composite service receives the request and proceeds with the airline reservation and hotel reservation, and the obligation enforcer makes sure that the service from MapQuest is obtained without providing the mobile number.

3 Service Composition

Service composition is a technique in which an advanced service can be built by composing a number of smaller services that implement a relatively simpler task. Though several automated approaches have been proposed for service composition, we note that our focus here is not on service composition itself but on a different goal: in providing consumer data privacy in service composition.

3.1 Composition Language

We shall use some formalism to introduce the composition framework. A natural formalism involves construing every component service as a function that maps a set of inputs to a set of outputs. Let $(i_1, i_2, i_3, \ldots, i_n)$ denote a set of inputs to an individual service A, and the corresponding outputs be $(o_1, o_2, o_3, \ldots, o_m)$. Each of the individual inputs can come from primitive types such as `int`, `float`, `string`, or aggregated types. Now each composition can be understood as a function that maps the outputs of service A to inputs of other services.

Several standard languages have been proposed for web services composition, notably BPEL4WS [3] and WSCI [1]. We used a simple imperative language to write composite code in our prototype implementation. This language is similar to the above languages, but with constructs to allow us to directly refer to service models and to facilitate the automated policy enforcement process detailed later in this paper. In the future, we shall explore ways to implement these functionalities in a standard web service composition language.

Each composition code consists of three sections: *service declaration*, *variable declaration*, and *composition sequence*. Service declaration defines simple services that the composite service is based on, while variable declaration defines variables used in the composition. The last section, composition sequence, expresses the actual composition logic through a sequence of composition actions, which include service invocations (`INVOKE`), variable assignments (`ASSIGN`), conditional branches (`SWITCH`), and loops (`WHILE`). Though the composition code is responsible for the control flow of the composition logic and handling data flows among participating component services, we note that it does not directly manipulate any data values other than propagating these values from one component service to another. All the actual data processing is performed within component services themselves.

A compiler for the composition language has been developed, which translates the composition code of each composite service into a Java program that can be linked into executables with service libraries as well as other libraries.

Services are central to composition, and the APIs for accessing services in Java composition code are defined in service libraries. In our prototype implementation of the framework, we make use of the WinAgent system [5] to create open wrapper services around existing web sites.

Each component service is represented as an object of class `Service`. A `Service` object can be constructed from a service definition XML file, and defines the following important methods:

- `int setInput(String nam, Object val)`. Sets the value of service input named `nam` to `val`.
- `Object getOutput(String nam)`. Returns the value of service output named `nam`.
- `int execute()`. Executes the service.

Composition example. Below is the composition code snippet of the travel management service example described in Section 2.

```
PARTNERS {
  SERVICE jb("jetblue_service.xml");
  SERVICE htl("hotels_service.xml");
  SERVICE mq("mapquest_service.xml");
};
VARIABLES { ... };
SEQUENCE {
  INVOKE jb
    IN  ["sourceCity": srcCity;
         "destinationCity": destCity;
         "creditCardNo": ccNo ]
    OUT ["price": price;
         "confirmNo": confirmNo];
  INVOKE htl
    IN  ["address": destCity;
         "creditCardNo": ccNo]
    OUT ["hotel": hotel];
  INVOKE mq
    IN  ["fromAddress": destCity;
         "toAddress": hotel]
```

```
        OUT ["Direction": directions]; };
```

The generated Java code snippet is shown as follows:

```
Service jb = new Service("jetblue_service.xml");
jb.setInput("sourceCity", srcCity);
jb.setInput("destinationCity", destCity);
jb.setInput("creditCardNo", ccNo);
status = jb.execute();
price = jb.getOutput("price");
confirmNo = jb.getOutput("confirmNo");
...
```

4 Service Model

The model of a web service is built by the service producer and encodes the data flows among its inputs, outputs, and interacting principals. In the service composition, the model is specified as an additional argument to the service definition construct `SERVICE`. Using the model, the policy compliance checker can verify whether the web service satisfies the consumer's privacy policy.

4.1 Model Language

In a model for a component service, the relations *input* and *output* define data items that flow into and out of service principals, respectively. A model also captures two kinds of data flow dependences that happen in a service: from an input to an output, and from an input to a principal. These data flows are specified by the relation *depend* defined below (Here we do not distinguish principals and outputs, and just mention sources and destinations):

The relation *depend* is defined as a subset of the cross-product of destinations and sources. Any member (Y,X) of this relation is a tuple that suggests that there is a flow of information from source X to destination Y. Note that *input(Y,X)* automatically implies *depend(Y,X)*.

The model of a composite web service is basically a collection of the models of its component services. However, the data item names used in the individual models are *canonicalized* such that they are unique across the entire composite service model. This renaming is necessary to reflect the data flow relationship among the component services due to the composition operation.

Model example. We present part of an example model for the JetBlue service in the travel management composite service as follows:

```
        input(Jetblue,srcCity).
        input(Jetblue,destCity).
        input(Jetblue,creditCardNo).

        output(Jetblue,price).
        output(Jetblue,confirmNo).

        depend(price,srcCity).
        depend(price,destCity).
```

This model states that the output `price` depends on the inputs `srcCity` and `destCity`, and the output `confirmNo` does not depend on any inputs. Because the *input* relation implies *depend*, the model also (implicitly) states the values of the three inputs, `srcCity`, `destCity`, and `creditCardNo`, flow into the principal entity `Jetblue`;

Currently our model representation captures only data flow relationships between inputs and outputs. But the model can be extended to support more expressive representations such as those carrying temporal control-flow based relationships. ("Input to principal B or principal C is given strictly after output is received from principal A"). Consequently, the algorithm described in Section 6 that is used to check the model against a policy needs to be augmented to handle these additional features.

Finally, we mention a few words about model construction. The model can be constructed manually by the web service provider (in a manner similar to P3P) by formalizing its terms of use, to our language. Automated source code analysis techniques can also be used to extract models, however, a discussion of such techniques is beyond the scope of this paper.

5 Privacy Policy

A privacy policy describes the privacy requirements of a user by defining constraints on how her data could flow between different entities. Because privacy concerns vary across users, privacy policies are defined by users and checked at the user end in our framework. Also user-side policy checking naturally allows for policy refinement in case of non-compliance. Policy specifications are also neutral to services, i.e., they express a user's privacy concern independent of the services that might be executed.

5.1 Policy Language

To specify privacy policies, our policy language uses an important concept, *label*, which is described as follows.

Labels. In our approach, a piece of user's data (data item), can flow into various entities (principals), such as service providers. Both data items and principals are grouped using *labels* to ease policy specification and management.

Labels are classified as *data labels* and *principal labels*, which are used to group data and principals, respectively. Each label is a vector of label attributes, each representing a particular privacy-related aspect of data or principals. Each label attribute can have a finite number of possible values, and is also partially ordered. In our implementation, we have defined two label attributes (*sensitivity level* and *information category*) for data labels, and one label attribute (*trust level*) for principal labels.

Data label attributes. The attribute *sensitivity level* describes how sensitive a piece of data is, and the possible sensitivity levels can be *highly sensitive*, *sensitive*, *less sensitive*, and *public*. For instance, one might consider her credit card number and bank account number to be highly sensitive; the credit card expiration date or the bank name to be sensitive; and data such as which country she lives in to be less sensitive. Then these data items can be assigned to sensitivity levels *highly sensitive*, *sensitive*, and *less sensitive*, respectively.

Data items can also be categorized into different information categories. A user might classify her social security number (SSN) and date-of-birth as personal information, her credit card numbers as financial information, and her residence street name and number as address information. Such categories are described by the label attribute *information category*. Figure 2 presents example data labels based on sensitivity levels and information categories.

	Highly sensitive	**Sensitive**	**less sensitive**
Personal	SSN, DoB	age	name, gender
Address	apt. no.	street no.	city, zip code, state, country
Financial	account no.	routing no.	have checking?
Payment	CC no.	CC exp date	CC type
Health	medical history	blood pressure	height

Figure 2. Data labels based on sensitivity levels and information categories.

Using the label attributes sensitivity level and information category, a consumer can to express more expressive privacy policies. Let us take the case of a consumer who considers that both her SSN and credit card number as sensitive information. When she is shopping in an on-line store, she may be willing to give her credit card number to the store, but not her SSN. By classifying SSN and credit card number into different categories, we can differentiate the information flow requirements for these two applications.

Principal label attributes. A user may trust different web sites or web services in varying ways. A well established on-line retailer might be deemed as "highly trusted", a less known web site might be treated as "less trusted", and a random web site might be considered as "untrusted". The principal label attribute *trust level* is used to express such users' privacy preferences.

Policy rules. A user's privacy policy $\mathcal{P}$ is formalized as a set of policy rules. A policy rule is of the form:

```
pref(X,Y,Action).
```

which either grants a flow of data with label `X` to a principal with label `Y` when `Action` is "allow," or restricts such a flow when `Action` is "deny." As an example, if a user does not want to release any sensitive financial information to a less trusted site, she can specify a policy rule:

```
pref((sensitive,financial),(less_trusted),deny).
```

The default action (when a rule for a piece of data is unspecified) is to deny.

6 Policy Enforcement

Once the model of the composite service is received by the user, her privacy preferences need to be enforced. Our approach for enforcement involves two stages: the first is a (static) policy compliance checking stage, which proceeds to check the model against the consumer specified policy $\mathcal{P}$, detect any violations and possibly generate obligations to the composite service. The second stage is when these obligations are enforced through a dynamic analysis approach on the composite code. In this section, we first describe the algorithm for static checking used to detect violations, followed by the dynamic analysis for obligation enforcement.

Label initialization. As mentioned earlier, the policy is specified in terms of labels (e.g. sensitivity), while the model (after the variable renaming step during composition), is described in terms of data item names. So as the first step in policy enforcement, the labels described using the consumer preferences in the form of a relation such as `pref(X,Y,allow)`, where X and Y are labels, are now converted to actual principal and data item names present in the model for the composite service. This results in `cpref`, a new concretized version of the relation `pref`. This new relation specifies if a particular principal has access to a particular data item. For instance, consider a user's privacy policy expressed as `pref(highly_sensitive, trusted, allow)`. If the data item `ccNo` represents the credit card number, and is assigned the label `(highly_sensitive)`, and if the principal `JetBlue` is trusted, then through label initialization the new relation `cpref` contains the following rule: `cpref(Jetblue,ccNo, allow)`. This is shown in lines 1 to 13 in the privacy policy checking procedure given in Figure 3.

Dependency propagation. Recall that the dependency relationships expressed in the service models denote the relationship of the inputs received by the model to its outputs. If these relationships were not taken into account in the policy checking procedure, information flows that arise due to these dependencies will remain ignored, and will affect the correctness of the approach. For instance, if the model contains the terms `input(Jetblue, srcCity)` and `depend(price, srcCity)`, the latter expressing the dependency between data items `srcCity` and `price`, then any web service that receives the price may be able to infer the value of `srcCity`. Hence, our approach propagates the user's policy across these dependencies. This is done by computing a transitive closure of the `cpref` relation, with respect to the `depend` relation. This is shown in steps 14 to 19 in Figure 3.

Policy compliance checking and obligation generation. The third and final step (lines 20 to 26 in Figure 3) involves checking the user's privacy preferences as given in `cpref` with that of the `input` to various web services. Any

```
$input$: models $(input, depend)$ relations, policy $pref(X, Y, Z)$ relation
$begin$
for all data labels $(\ell_c, \ell_s)$ and for all principal labels $l_p$ do
    if $pref((\ell_c, \ell_s), \ell_p$, allow) then
        for all data items $d$ with label $(\ell_c, \ell_s)$, and for all principals $p$ with label $l_p$ do
            $cpref(d, p, allow)$
        end for
    else
        for all data items $d$ with label $(\ell_c, \ell_s)$, and for all principals $p$ with label $l_p$ do
            $cpref(d, p, deny)$
        end for
    end if
end for/* end label initialization */
$newdepend = transitive\text{-}closure(depend)$
for all $d, p$ such that $cpref(d, p, deny)$ do
    for all $d_1$ such that $newdepend(d, d_1)$ do
        add-rule $cpref(d_1, p, deny)$
    end for
end for/* end dependency propagation */
for all $d, p$ such that $cpref(d, p, deny)$ do
    if input(d,p) then
        raise violation /* for refinement */
        $generate\text{-}obligation(d, p)$ /* if necessary */
    end if
end for/* end policy check */
end
```

Figure 3. Algorithm for policy compliance checking and obligation generation

policy violations indicate that the web service's input requirements do not match the user's privacy policy. In this case, a set of violations ("service MapQuest is given the data item phone number") is generated. The user can then decide to relax her policy (by allowing MapQuest to have her phone number) or generate an obligation that the composite service needs to respect. The obligation is also a relation `obligation(Principal,DataItem)` that suggests to the enforcement mechanism that `Principal` should not be given `DataItem`. In the above example, the obligation will be `obligation(Mapquest,phone-number)`. Such obligations are forwarded to the composition code for enforcement, which is described next.

Obligation enforcement. Once enforcement obligations are generated at the consumer site, they are passed on to the composite service. To enforce these obligations, the composite service needs to track whether the flow of consumer's data inputs (through various actions of the composite service) respect these obligations. For instance, when `obligation(Mapquest,phone-number)` is provided to the composite service, the tracking needs to check whether data related to the consumer's phone number is provided to MapQuest during code execution. As the obligations are only available during service execution, our framework performs a dynamic analysis of the actions of the composition code to perform such tracking. The monitoring code that performs dynamic analysis itself is introduced as part of the composition code through a program transformation operation. The details on the program transformation technique and proofs of correctness are available in [10].

Below we show an example of transforming part of the travel management service composite code.

```
Obligation jb_oblg; /* obtained from consumer */
Service jb=new Service("jetblue_service.xml");
Model jb_model=new Model("jetblue_model.xml");
...
jb.setInput("creditCardNo", ccNo);
jb_model.setInput("creditCardNo",ccNo_lb);
if (jb_oblg.verifyObligation(jb_model)!=SUCCESS)
    /* perform alternate actions */
    raise PolicyViolationException;
status = jb.execute();
price = jb.getOutput("price");
price_lb = jb_model.getOutput("price");
```

7 Example Services

We implemented a prototype of the framework and built two example composite services in the framework. Each component service is implemented as a WinAgent agent.

Travel management service. This is similar to our running example. The inputs to this service are the origin, the destination of travel, the zip code of the destination and the credit card number. Agents are built for crawling and searching a number of leading airline websites, whose results are put together in a consistent order for the user to choose the most ideal itinerary. After the consumer chooses the appropriate air travel, hotel and car reservations are done and detailed driving directions are also obtained. Based on the level of security assigned to each site that is explored, the details such as destination zip code and credit card number are shared. Thus the privacy of the user and the details of her travel are preserved appropriately.

Electronics shopping service. This composite service queries different websites for a particular product, presents reviews, and queries the best possible prices for the chosen item from a set of trusted web sites. A buying agent is then invoked to complete the purchase. During all these operations, the privacy policies of research, price search and buy are verified for every agent that is invoked.

8 Related work

Access control based approaches (including *sandboxing* [6] approaches) do prevent information from leaking from a given resource (such as a user's computer or a web site), but they do not address the problem of controlling access to a piece of information once it leaves a given host with the consent of the user. An approach that addresses privacy of information must address the problem of control to be effective. Anonymity based approaches [7, 4] also do not help, as disclosure of personal information is required in situations such as making airline reservations.

Several program analysis techniques for securing programs from information leaks [9, 12]. These approaches are not particularly suitable in the context of services provided by various web sites, as users privacy information is only available when the service actually executes and hence dynamic tracking approaches such as ours are needed.

Solutions such as P3P [2] (Platform for Privacy Preferences) have aimed at web resources such as cookies and files. P3P has been developed for individual web sites and not for composite web services, consequently is challenged in its ability to describe data flows within a composite webservice. Also, P3P framework does not provide any support for the consumer to provide any feedback to the service. Our framework supports this through obligation generation and enforcement.

One of the first known approaches for privacy in the context of web services was presented by Rezgui et al [8]. Their work involves private information stored and retrieved from databases through web services (e.g., governmental web sites) and credential based access control techniques to prevent unauthorized access to this information. By allowing users to describe their privacy preferences, the system provides mechanisms that control access to this information both at the client and server side. However, as mentioned above, access control mechanisms only prevent unauthorized access to information. They do not put limits on how this information can be used after the initial access is granted. Reasoning the information flows through the composite web service is critical for providing guarantees about further information leakage, and our work addresses this aspect of privacy.

Our earlier work [11] on privacy policy enforcement used a similar program transformation approach for enforcement of consumer privacy policies. This work was done in the context of enforcing privacy policies in an enduser application. However, in the context of composite web services the same approach suffers from two main weaknesses. If code for component services is not available the approach is not applicable. In this paper, models bridge this gap and provide the relationships between inputs and outputs in component services. Secondly, a user's privacy policy was entirely provided to the application for enforcement, which is not desirable in the context of web services. In our current approach, obligations (if chosen by the consumer) provide the right level of feedback based on the service's violation of the consumer privacy policy.

9 Conclusion

In this paper, we have proposed a framework for preserving privacy in web-based services. In our framework, consumers can have facilities to specify their privacy concerns through use of privacy policies, while service providers express their terms of use (of private data) through models. The compatibilities between privacy policies and service models can be verified automatically at the consumer end using the techniques proposed in this paper. Any conflicts can result in obligations that are provided to the producer, who can enforce these obligations using dynamic analysis techniques that we have proposed. We believe that addressing consumer privacy is an important area in composite web services, and the framework presented in this paper is a significant step in that direction.

References

[1] A. Arkin and S. A. et al. Web services choreography interface. Technical report, W3C consortium, 2002.

[2] L. Cranor. *The Platform for Privacy Preferences 1.1 (P3P1.1) Specification*. W3C working draft, July 2004.

[3] F. Curbera and Y. G. et al. Business process execution language for web services. Technical report, IBM Developerworks, 2002.

[4] E. Gabber, P. B. Gibbons, D. M. Kristol, Y. Matias, and A. Mayer. Consistent, yet anonymous web access with lpwa. *Communications of the ACM*, 42(2), February 1998.

[5] A. Gandhre, P. Santhanagopalan, P. Singh, D. Ramavat, I. Ramakrishnan, and H. Davulcu. Creating and managing personal information assistants via a web browser: The WinAgent experience. In *Workshop on Information Integration on the Web*, Toronto, August 2004.

[6] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A secure environment for untrusted helper applications: confining the wily hacker. In *USENIX Security Symposium*, 1996.

[7] M. K. Reiter and A. D. Rubin. Anonymous web transactions with crowds. *Communications of the ACM*, 42(2), February 1999.

[8] A. Rezgui, M. Ouzzani, A. Bouguettaya, and B. Medjahed. Preserving privacy in web services. In *Workshop on Information and Data Management*, 2002.

[9] A. Sabelfeld and A. C. Myers. Language-based informationflow security. *IEEE J. Selected Areas in Communications*, 21(1), Jan. 2003.

[10] V. N. Venkatakrishnan, D. C. DuVarney, W. Xu, and R. Sekar. A program transformation approach for enforcement of information flow properties. Technical Report SECLAB-04-01, Department of Computer Science, Stony Brook University, 2004.

[11] V. N. Venkatakrishnan, W. Xu, I. V. Ramakrishnan, and R. Sekar. A secure composition framework for trustworthy personal information assistants. In *IEEE conference on Integration of Knowledge Intensive Multi-Agent Systems (KIMAS)*, April 2005.

[12] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security (JCS)*, 4(3):167–187, 1996.

[13] W. Xu, R. Sekar, I. V. Ramakrishnan, and V. N. Venkatakrishnan. An approach for realizing privacy preserving webbased services. In *World Wide Web conference (WWW) (poster presentation)*, May 2005.