Provably Secure Execution of Composed Semantic Web Services

Dieter Hutter, Melanie Volkamer, Matthias Klusch, Andreas Gerber

German Research Center for Artificial Intelligence (DFKI GmbH)
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
`[hutter, volkamer, klusch, agerber]@dfki.de`

Abstract. In this paper, we present an approach to solve the problem of secure execution of semantic web service composition plans. The integrated components of this approach include our OWL-S service matchmaker, OWLS-MX, the service composition planner, OWLS-XPlan, and the security checker module for formally verifying the compliance of the created composition plan to be executed with given data and service security policies using type-based information flow analysis.

1 Introduction

The composition of complex services available in the web, and the semantic web, at design time is a well-understood principle which is nowadays supported by, for example, service composition planners such as SHOP2, or OWLS-XPlan. However, ensuring the secure execution of composed services still remains to be a challenge. Related tasks to pursue range from secure communication, via protection of services against misuse, to the preservation of user data privacy and integriy. Standard approaches for secure execution of services such as those using REI [12] or Ponder [3] are based on the specification of access control policies that control the individual execution of services as actions on individual objects and thus focus on the first two tasks. With respect to privacy aspects, access control policy mechanisms suffer from the problem of Trojan Horses, or information leakage caused by hidden channels. The main reason for this is, that no security policy control is enforced on the use of provided data once it has been released to the authorized web service. Improper processing of confidential information, and subsequent calls of unauthorized subservices by an agent offering a composed service as part of the composition plan to be executed could lead to the revelation of private information even without intention.

In particular, we have to address the following questions: How can we represent and formalize privacy concerns of users, i.e. how to denote security *classification* of provided data and the user's security rating of web services (the *clearances* of web services)? How can we propagate classifications of input data to classify newly computed data in view of a dynamic composition of the program or plan? How then can we check automatically whether a web service call complies to a given security policy? How can a subsequently called web service enforce its security requirements on its own data (provided as a result of its call)? How can we guarantee the existence of an overall consistent security policy for the total encomplishment of a service.

Since access control mechanisms are obviously inappropriate to cope with these problems, we propose to use information flow techniques [4] in general, and techniques from program language security [17] in particular. Analogously to the concept of proof carrying code, we propose to add a (security) type checking mechanism to the implementation of web services that enables an agent to do an information flow analysis on dynamically generated plans or programs. This type checker is used to enforce the privacy requirements of a user by guarding calls of other web services and avoiding the execution of plans that would result in a prohibited leakage of information. In this paper, we apply this approach to the problem of privacy preserving execution of web services described in OWL-S as part of a given composition plan generated by our semantic web service composition planner OWLS-XPlan. Please note, however, that the integrated component for security checking of both individual services and the composition plan as a whole can be, in principle, applied to any kind of web services such as those described in WSDL, or WSMO.

The remainder of this paper is structured as follows. We motivate our research on the problem of provably secure execution of service composition plans by means of a brief use case description in section 2. Section 3 then provides an overview of our solution approach to this problem, while sections 4 to 6 then describe its main components, including the matchmaking, composition planning, and security check, respectively, in more detail. We compare our approach to existing ones in section 7, and conclude in section 8.

2 Use Case Scenario

Throughout the paper we will motivate our approach with the help of a use case which will be described in the following. Figure 1 illustrates this example. Living in London, a politician, named Susan Miller, wants to give an invited public talk in the international congress center of Berlin, Germany, and meet the Italian prime minister the day after in a secret get-together in Rome, Italy. She instructs her personal secretary agent running on her PDA to accomplish this particular task by entering the flight destinations, banking details for payments, personal data and security classification with respect to the category of location and payment.

As a result, the agent first discovers semantically relevant services, and then generates a composition plan that satisfies the given goal without checking any security policy. In this scenario, the goal is decomposed into subgoals of reserving flights from London to Berlin, with a connection flight to Rome, and execution of payment. The generated service composition plan proposes to subsequently call the service of travel agents TA_1 (AirBerlin) for the availability and costs of a flight to Berlin and TA_2 (Alitalia) for the availability and costs of a flight to Rome, and then to call the paypal service offered by agent PA_2 (PayPalService-

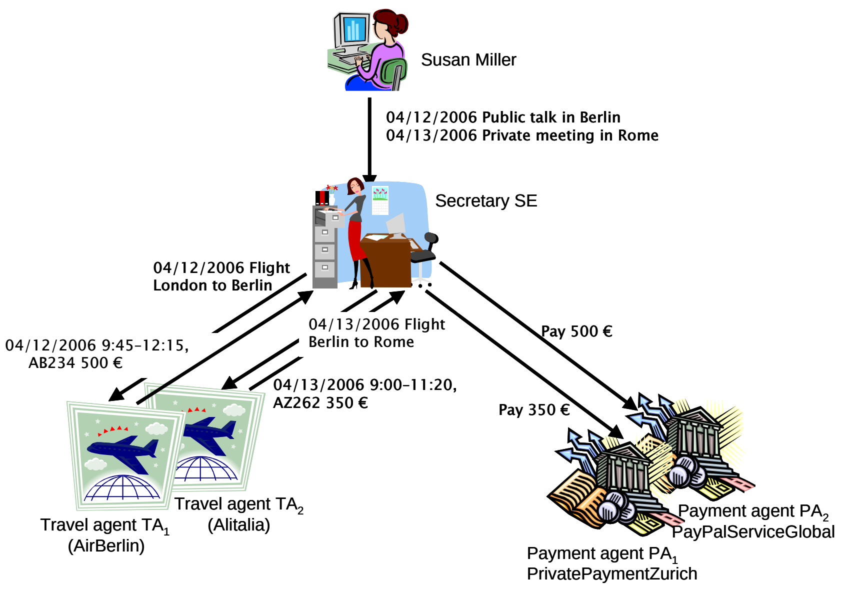

Susan Miller
04/12/2006 Public talk in Berlin
04/13/2006 Private meeting in Rome
Secretary SE
04/12/2006 Flight London to Berlin
04/12/2006 9:45-12:15, AB234 500 €
04/13/2006 Flight Berlin to Rome
04/13/2006 9:00-11:20, AZ262 350 €
Pay 500 €
Pay 350 €
Travel agent TA1 (AirBerlin)
Travel agent TA2 (Alitalia)
Payment agent PA2 PayPalServiceGlobal
Payment agent PA1 PrivatePaymentZurich

Fig. 1. Use case scenario: Execution of composed travel and payment services.

Global) to pay for both flights. The secretary agent formally checks the secure execution of this plan with respect to the security requirements of the politician and the services involved in the plan.

To keep Susan's flight to Rome private, the required input (04/13/2006 Flight Berlin to Rome) for the travel agent service of Alitalia (TA_2) have to be kept confidential. If the price (350 €) for the ticket of the selected flight from Berlin to Rome depends on the destination (Rome), then also the offered price has to be kept private to avoid any other agent to deduce the location Rome from the amount of payments to be done. In other words, even if the payment service agent PA_2 does not get any details of the booked flight, Susan would have to trust the payment service in keeping any static, or even dynamically generated data with respect to the price private.

Suppose Susan has made bad experiences with the proposed agent PA_2 since in the past information about her whereabouts were leaked to the press. Since the generated plan would provide PA_2 with the price of the flight to Rome, the plan violates Susan's security requirements. However, Susan trusts another payment service offered by agent PA_1 in this point. To fix the failed plan, a possible improvement would be to replace the payment agent PA_2 by the payment agent PA_1 in order to pay the flight to Rome. The situation changes if the travel agency Alitalia (TA_2) offers a flat rate for European flights. Then the price of a flight does not depend on the selected destination anymore. In this case, Susan does not have to trust the discretion of the payment service (PA_2) with respect to her whereabouts and the plan could be executed without security policy violations.

To evaluate the plan from a security point of view, Susan has to provide her secretary agent with her personal security requirements. She has to define which data could be provided to which web service. Technically, the service agent assembles a list of *clearances* for various web services and rates Susan's data according to her requirements. During the check of generated plans, her secretary agent also assembles the security assurances exported by the involved web services. Susan's individual security requirements (containing the clearances of services and the classification of data) will be dynamically extended to cope with newly computed or generated information, and subordinate services not primarily known.

In the following section, we provide an overview of Susan's secretary agent, where each of its main components is described in more detail in the following sections using this case scenario as a running example.

3 Architectural Overview

The basic architecture of our secure service composition planning agent (SCPA) is shown in figure 2. As input, the SCPA requires the request for some desired service in OWL-S 1.1 from the user, and her local security policies in terms of the security classification of personal data and clearance of known web services. The SCPA then attempts to discover OWL-S web services that are semantically relevant to the request using its service discovery and matchmaker module, named OWLS-MX. In addition it collects the corresponding security types published by the respective web service provider agents. If the matchmaker finds services detected as being equivalent to the requested one, it directly passes the top ranked one according to its QoS value to the security checking module to verify whether its published security policy complies with given local security policies and with the web service's security type. If no equivalent service is found, the OWLS-MX module passes the set of services to its composition planner, named OWLS-Xplan. The planner con-

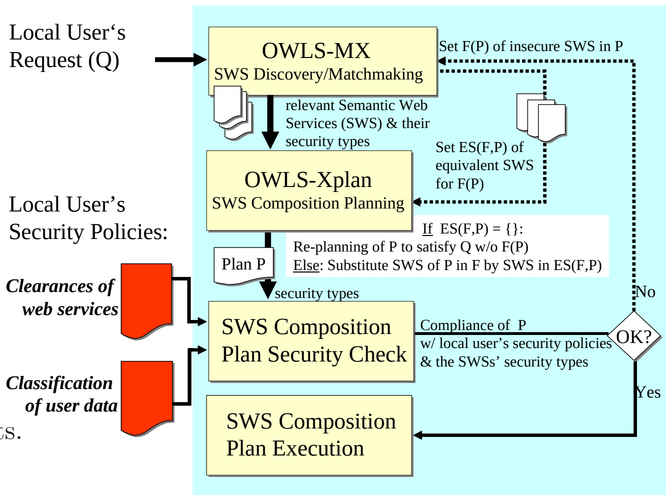

Local User's Request (Q)
OWLS-MX
SWS Discovery/Matchmaking
Set F(P) of insecure SWS in P
relevant Semantic Web Services (SWS) & their security types
Set ES(F,P) of equivalent SWS for F(P)
OWLS-Xplan
SWS Composition Planning
Local User's Security Policies:
If ES(F,P) = {}:
Re-planning of P to satisfy Q w/o F(P)
Else: Substitute SWS of P in F by SWS in ES(F,P)
Plan P
security types
No
Clearances of web services
SWS Composition Plan Security Check
Compliance of P w/ local user's security policies & the SWSs' security types
OK?
Classification of user data
Yes
SWS Composition Plan Execution

Fig. 2. Basic architecture of SCPA.

verts both the request and all OWL-S services retrieved into an initial state and goal ontology written in PDDXML, and generates a sequential service composition plan that satisfies the goal. In case a composition plan with more than one service is generated, the compliance of published security types of all web services involved in the plan is checked against the local security policies of the user. In contrast to usual access control mechanisms, the security checking of the SCPA relies on type-based information flow analysis. Thereby the approach also includes dynamically computed data of web services and their security classification, and its proliferation to other services.

In any case, the composition plan gets executed only if the security types of all web services meet the local security policies. So, the plan as a whole is formally verified as being secure. Otherwise the SCPA triggers a re-planning activity to be performed as follows. The security checker provides the matchmaker module with a set $F(P)$ of services of plan P that caused P to not comply with the local security policies, in order to select one semantically equivalent service with a different published security policy for each or at least some of them. If successful, the composition planner simply modifies the original plan considered by replacing each service in F by its substitute, and returns the modified plan to the security checker for verification. If there exists no services in $F(P)$ for which equivalent service can be found (and which are not yet tried), the composition planner generates a new plan by means of heuristic replanning [9]. In any case, if the modified plan is also provably insecure, the SCPA repeats the same procedure until a secure composition plan is generated, or it returns a failure otherwise.

The SCPA executes a secure plan sequence in joint collaboration with those agents that provide the services involved. For this purpose, it calls each of them by sending the required input data. In addition the SCPA transmits information on the clearance of other services with respect to the information category of the input data. For example, if a service is trusted by the user to preserve the privacy of data of the local information category “Location”, the SCPA also sends its actual list of clearances of other services for this particular category.

To summarize, the SCPA assists its user in service oriented computing tasks by means of automatically searching for, and composing individual or composed service. Moreover the SCPA ensures that plans are only executed if the web services are provably secure with respect to the security policies and security type. We acknowledge that the amount of security related information the user provides to her SCPA in terms of classified data and service clearances determines the degree to which the security of an automatically generated composition plan can be formally verified.

4 Service Discovery and Matchmaking

Our secure service composition planning agent (SCPA) uses both a service discovery and a service matchmaking module to discover services that are semantically relevant to the given request. Depending on the resources available, it can perform any combination of very fast keyword based search for relevant services in publicly accessible OWL-S service registries with a computationally more expensive semantic filtering of retrieved, or locally registered services.

The OWLS-MX matchmaking module takes any description of a desired service written in OWL-S 1.1 as an input, and returns an ordered set of relevant services that match, each of which annotated with its individual degree of matching, and syntactic similarity value. The user can specify the desired degree, and syntactic similarity threshold. Like in [7], the discovery module also stores the published security policies of the discovered services for further use by the security checking module, while the matchmaker focuses on its core functionality, that is semantic matching.

The OWLS-MX matchmaker first classifies the service query I/O concepts into its local service I/O concept ontology. For this purpose, it is assumed that the type of computed terminological subsumption relation determines the degree of semantic relation between pairs of inputs and concepts. Auxiliary information on whether an individual concept is used as an input or output concept by any registered service is attached to this concept in the ontology. The respective lists of service identifiers are used by the matchmaker to compute the set of relevant services that I/O match the given query according to its five filters.

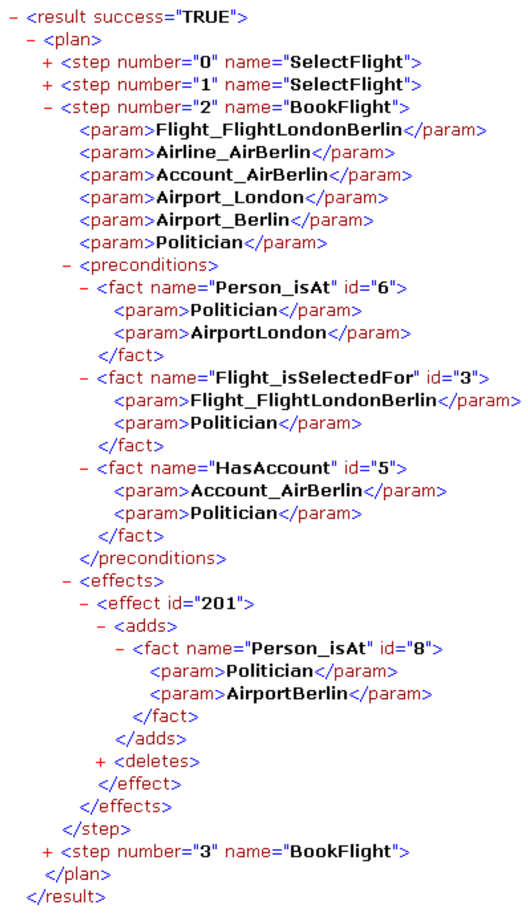
The OWLS-MX module does not only determine pairwise the degree of logical match but syntactic similarity between the conjunctive I/O concept expressions in OWL-Lite. These expressions are built by recursively unfolding each query and service input (output) concept in the local matchmaker ontology. As a result, the unfolded concept expressions are including primitive components of a basic shared vocabulary only. Any failure of logical concept subsumption produced by the integrated description logic reasoner of OWLS-MX will be tolerated if and only if the degree of syntactic similarity between the respective unfolded service and request concept expressions exceeds a given similarity threshold. For more detailed information on the OWLS-MX module of the SCPA, we refer the interested reader to [8].

5 Service Composition Planning

The OWL-S service composition planning of the agent is performed by OWLS-XPlan, whenever no appropriate single service can be found during matchmaking. OWLS-XPlan takes a set of available OWL-S services, related OWL ontologies, and an OWL-S query as input, and returns a plan sequence of composed services that satisfies the query goal. For this purpose, it first converts the domain ontology and service descriptions in OWL and OWL-S, respectively, to equivalent problem and planning domain descriptions in the standard language PDDL. The problem description contains the definition of all types, predicates and actions, whereas the domain description includes all objects, the initial state, and the goal state. Both descriptions are then used by the AI planner XPlan to create a composition plan that solves the given problem in the actual domain.

XPlan is a heuristic hybrid FF (FastForward) planner which combines guided local search with graph planning, and a simple form of hierarchical task networks to produce a plan sequence of actions that solves a given problem. It uses an enforced hill-climbing search method to prune the search space during planning, and a modified version of relaxed graph-planning. This version allows the use of (decomposition) information from hierarchical task networks during the efficient creation of the relaxed planning graph, if required, such as in partially hierarchical domains. Information on the quality of an action (service) are utilized by the local search to decide upon two or more steps that are equally weighted by the used heuristic. In addition, XPlan includes a re-planning component which can be called by the security checking module on demand.

Figure 3 shows a part of the travel and payment service composition plan generated by OWLS-XPlan for our use case scenario, encoded in PDDXML.



```
- <result success="TRUE">
  - <plan>
    + <step number="0" name="SelectFlight">
    + <step number="1" name="SelectFlight">
    - <step number="2" name="BookFlight">
        <param>Flight_FlightLondonBerlin</param>
        <param>Airline_AirBerlin</param>
        <param>Account_AirBerlin</param>
        <param>Airport_London</param>
        <param>Airport_Berlin</param>
        <param>Politician</param>
      - <preconditions>
        - <fact name="Person_isAt" id="6">
            <param>Politician</param>
            <param>AirportLondon</param>
          </fact>
        - <fact name="Flight_isSelectedFor" id="3">
            <param>Flight_FlightLondonBerlin</param>
            <param>Politician</param>
          </fact>
        - <fact name="HasAccount" id="5">
            <param>Account_AirBerlin</param>
            <param>Politician</param>
          </fact>
        </preconditions>
      - <effects>
        - <effect id="201">
          - <adds>
            - <fact name="Person_isAt" id="8">
                <param>Politician</param>
                <param>AirportBerlin</param>
              </fact>
            </adds>
          + <deletes>
          </effect>
        </effects>
      </step>
    + <step number="3" name="BookFlight">
    </plan>
  </result>
```

Fig. 3. Part of service composition plan generated by OWLS-XPlan.

For more detailed information on the OWLS-XPlan composition planning module of the secure planning agent, we refer the interested reader to [9].

6 Plan Security Checking

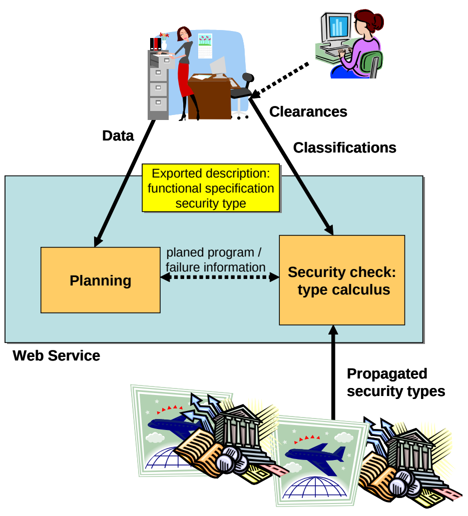
Once the service composition plan has been created, the agent checks whether it complies with given classifications, clearances and published security types of local user data and web services involved in the plan.

6.1 Privacy of user data

To protect the privacy of user related information, the data used in web services is always *classified* according to its confidentiality. Web services require the corresponding *clearances* to deal with confidential data. Both classifications and clearances are denoted by a so-called *security rating*. There is a partial ordering $\leq$ on security classes which allow us to compare them. The set of all security classes together with $\leq$ forms a lattice, i.e. for two arbitrary security classes there is always a least upper bound. In the simplest case we may have H and L as the set of security classes denoting confidential (H or "high") and public (L or "low") data, respectively.

Similar to the approach presented by Bell and LaPadula [1], the idea is that a web service is only entitled to obtain a specific datum if its clearance is at least as high as the classification of the data. For example, in order to receive data classified as H, a web service needs a clearance H while web services with clearance L are not entitled to get any H-classified data.

However, in practice we would like to select the clearances of web services and also the classification of data with respect to individual categories or types of information. For instance, while we trust the travel agent to keep our travel routes confidential we might have mixed feelings when providing the same agent with a direct debit authorization for our bank account. Analogously, a given datum may allow us, for instance, to infer confidential information about the location of a person or confidential information about his bank account. Hence, both classifications of data and clearances of web services are described by a vector of security ratings. Each entry in the vector denotes the classification or clearance - such as, for instance, H and L denoting high confidentiality and public release, respectively - with respect to a particular category (like, for instance, location or payment information).


Data
Clearances
Classifications
Exported description: functional specification security type
Planning
planed program / failure information
Security check: type calculus
Web Service
Propagated security types
Fig. 4. Overview of security checking

The clearances of web services used for a web service request are assessed by the original provider of the data (typically the user) or, in case of a delegation, by a web service acting on behalf of the provider. In other words, an information provider may specify which web service it trusts to keep data private with respect to given categories, or not. Analogously, the provider determines the classification of the data that will be provided to web services. Web services classify data they provide as an output by integrating the classifications of the input data used to compile the result and the requirements of their own security policies.

The set of categories used to invoke a web service may change while processing the request. When a called web service provides new information, it may introduce a new category and classify the new data also with respect to this new category. In this way a subsequently called web service can formulate its security requirements for its provided data by defining also the clearances of all web services with respect to the new category. In this case, all data provided by the calling web service have to be classified as public with respect to the newly introduced categories. This is to avoid the blocking of external data by classifying them as confidential for a new category but providing no clearances for any web service.

6.2 Type-based information flow analysis

Web services deal with confidential input and in general their answers will also include confidential bits, i.e. knowing the output of a web service call (but not the input) we might be able to deduce constraints on the confidential input. If the knowledge of the output of a web service call would disclose information about a confidential input, the output itself has to be confidential as well. In order to assess the classification of data computed by web services we use information flow techniques in general and program language security techniques in particular.

Clearances and classifications are formalized by means of standard information flow policies [4, 11, 10]. Obviously, the output of a web service call does not contain any information about confidential input data if it does not *depend* on the concrete values of input data. Low-security data must not depend on any high-security data. More generally, the security classification of any computed or synthesized data has to be at least as high as the classification of all used data. That is, no secret bit of information must be disclosed in public information.

To ensure this restrictions on web services, we adopt a security type calculus developed by Volpano and Smith [17, 18, 14] to analyse synthesised plans and formally encode security classifications as types. Their approach secures information flow in a simplified programming language. They distinguish security classifications and security clearances. Security ratings τ, like H or L, are used to describe the classification of data (or expressions in a program). τ *acc* denotes the clearance (e.g. of a program variable) to store or keep information up to a classification τ. For instance, a variable of type H *acc* is entitled to store confidential data. Its security rating is H. Besides storing confidential information into low-security variables, a program may leak confidential information if confidential information causes the program to move into different branches of the program that cause different settings of low-security variables. For example, let `x` be a low-security and `y` be a high-security Boolean program variable, then `if y = true then x := true else x := false` implicitly copies the value of `y` to `x`. Thus, Volpano and Smith introduce a security type τ *cmd* for program statements or fragments denoting that the execution of this fragment can be only noticed by observers with clearance higher or equal than τ. Obviously, a program fragment is of type τ *cmd* if there are only assignments to variables that possess a clearance higher or equal than τ. Calculus rules are used to formally reason on security classifications and to propagate the types of data along the program to newly computed data. For instance, an assigment like `x := c` is secure if `x` has type τ *acc* and `c` the type τ or a type $\tau' \leq \tau$. Then, the statement itself has type τ *cmd*. Such rules are defined for all expressions and commands of the programming language. The programming language used in [18] comprises a notion of procedures as well.

The applied programming language and the web service composition plans are very similar. The only command that has to be added and modeled is the web service call. The underlying idea of our approach (compared to [5]) is that web service calls can be treated like remote procedure calls, while encoding global states as global variables common to various web services. However, in contrast to procedures, web services have to be first class citizens. In our approach web services possess an individual clearance. As a consequence, each service call has to be guarded by a check whether the input to be provided to this service lies within its clearance.

In particular, a web service $WS(x, y)$ with input parameter x and output parameter y has a security type $\tau\ proc(\tau_1, \tau_2)$ if the input variable x of security type τ_1 will not influence any (global) variable of a security type less than τ, and the clearance of the resulting output is of type τ_2; i.e. any variable storing the result of the web service call has to provide a clearance higher or equal then τ_2. Each web service will export its security type as a part of the interface description. Roughly speaking, it is a promise that the input x is kept confidential to all agents with a clearance less then τ_1 and that all changes to the outside only affects information rated τ and higher. Furthermore it is a requirement that the output has to be kept confidential to all agents with a clearance less than τ_2. If there is no observable global "world" state (i.e. there are no side effects of executing web service on the global state) then τ would be always the maximal upper bound (e.g. H). In general, web services are polymorphic in their types, i.e. τ_1 and τ_2 may be type variables rather than fixed values. Let τ_1 be a type variable then a type $\tau\ proc(\tau_1, \tau_1)$ would simply indicate that output require the same clearance as the classification of the input. An example would be a web service that simply copies its input to the output.

6.3 Propagation of clearances

The security type of a web service tells us about the propagation of confidential inputs to the outputs and to the global state. However, we also have to propagate the clearances a customer is willing to issue to individual web services or classes of web services. Therefore, each web service provides an additional input parameter to receive the clearances assigned to web services by the customer.

However, both the user and her agent do not necessarily know all web services that are involved in a particular composition plan, depending on the granularity of the respective process model specifications, or black-box views on subordinated services. Thus, the user may also specify delegation rules that allow trusted web service to fix the clearance of web services unknown to the user (and thus not specified by the existing clearance of web services). A web service may only *add* clearances to new web services but it must not change existing clearances. Once a web service has added a clearance it will be fixed till the end of the complete service. To communicate the addition to the clearances there is also an additional output parameter that propagates any increments of the list of clearances to the calling web service.

6.4 Trust

Once a customer provides a web service with a specific clearance she agrees that this service will be provided with confidential data if the classification of the data is less or equal than the clearance of the web service. However, she has also to trust that the web service's plans will always satisfy the propagated security types, i.e. that the web service will not betray her by sending confidential data to some unauthorised web service. One way to solve this problem is to certify web services if they include a security type checker that will automatically control generated plans. In this case a web service would export its security type combined with a certification of a trusted third party that its realization will enforce the proposed security types.

6.5 Application to the use case scenario

In the following we illustrate the sketched security checking mechanism within the use case scenario introduced in section 2. In a first step our politician Susan Miller has to select appropriate categories to classify her data and to formulate the clearances of the web services. To simplify matters, we assume that she chooses two categories. The first category "Location" is concerned with the confidentiality of her locations, while the second category "Payment" refers to the privacy of payment informations like details about her bank accounts or credit cards. Figure 5 illustrates the concrete security types for the data and web services occurring in our example. Being in London or Berlin is publicly known

Personal Data	Classification	
	Location	Payment
London, 04/11/2006	L	L
Berlin, 04/12/2006	L	L
Rome, 04/13/2006	H	L
...		

Web services	Clearance	
	Location	Payment
TA_1	H *acc*	L *acc*
TA_1	H *acc*	L *acc*
PA_1	H *acc*	H *acc*
PA_2	L *acc*	H *acc*

Fig. 5. Classification and clearances of user data and web services.

and thus, she rates this information as L in the category "Location". Since her stay in Rome is confidential, this information is classified as H. All these data contain no information about payments, and therefore they are L-rated in the category "Payment". Next she rates travel agencies and trusts all of them that they do not leak any information about her future whereabouts (rated as H). She does not trust these travel agencies with respect to any payment information (hence rated as L) but wants to receive an invoice that can be paid by a special payment service. She has two payment services at hand in which she trusts (thus rated as H). While the first payment service is her private bank that will not disclose any information related to the payment (therefore also H-rated wrt. the location), the second payment service engages untrusted employees that might have close relationship to some newsmen. Using these settings, Susan instructs her secretary agent to organize a round trip from London to Berlin on April 12, and to Rome on the day after. The request Q is of the form Is(London, 04/11/2006), Goal(Berlin, 04/12/2006), Goal(Rome, 04/13/2006), and Susan's security policies for personal data (cf. figure 5). As a consequence, the agent first generates a service composition plan (cf. figure 3) that would enable Susan's agent to accomplish the general flight reservation task by subsequent calls of services of the corresponding execution plan as follows:

$\mathbf{call}(\mathit{TA}_1 : \mathit{FlightReservation}, [\mathit{London}, \mathit{Berlin}, 04/12/2006], \mathit{price}_{\mathit{Berlin}});$
$\mathbf{call}(\mathit{TA}_2 : \mathit{FlugReservierung}, [\mathit{Berlin}, \mathit{Rome}, 04/13/2006], \mathit{price}_{\mathit{Rome}});$
$\mathbf{call}(\mathit{PA}_2 : \mathit{PayPalServiceGlobal}, \mathit{price}_{\mathit{Berlin}}, \mathit{okP});$
$\mathbf{call}(\mathit{PA}_2 : \mathit{PayPalServiceGlobal}, \mathit{price}_{\mathit{Rome}}, \mathit{okP}');$

The notation **call**(*agtID:WS*, In, *Out*) means that the web service *WS* of agent *agtID* should be called with the input value of variable In, and the returned answer be stored in the variable *Out*.

The secretary agent passes this execution plan to the security checking module together with the classifications of the personal data and the clearances of the web services as illustrated in Figure 5. By inspecting the service descriptions of the web services, the security checking module can extract their security types. Let us assume that both travel agents TA_1 (AirBerlin) and TA_2 (Alitalia) publish a security type $H\ \mathit{proc}(X, X\ \mathit{acc})$ (X being a variable) for their corresponding web services. This type reflects the fact that the output of the service, i.e. the price of the flight, depends on the destination and date of the flight. If an adversary knows about the price of the flight he might deduce possible destinations of the flight. Hence the classification of the price has to be as high as the classification of the whereabout; i.e. the price is public if and only if the whereabout is public. In addition, both payment agents, PA_1 and PA_2, specify $H\ \mathit{proc}(X, L\ \mathit{acc})$ as their service security type, since their output, which is the approval of payment (*okP*), does not depend on the amount of the payment.

In a next step the plan security checking module uses the type calculus to check security of the generated plan. Since the plan is a plain sequence of web service calls, the check is a simple propagation of security types. The first call of the plan is $\mathbf{call}(\mathit{TA}_1 : \mathit{FlightReservation}, [\mathit{London}, \mathit{Berlin}, 04/12/2006], \mathit{price}_{\mathit{Berlin}})$. According to the Susan's ratings, $[\mathit{London}, \mathit{Berlin}, 04/12/2006]$ is rated as L in both categories. Since TA_1 has type $H\ \mathit{proc}(X, X\ \mathit{acc})$, also the variable $\mathit{price}_{\mathit{Berlin}}$ has to have at least the clearance $L\ \mathit{acc}$ in both categories. Since L is the bottom element of the lattice, there are no restrictions to

$price_{Berlin}$ arising from this call. In the next call **call**(TA_2 : *FlugReservierung*, $[Berlin, Rome, 04/13/2006]$, $price_{Rome}$) the input $[Berlin, Rome, 04/13/2006]$ is rated as H in the category "Location" since the location $Rome$ is confidential. Since Susan grants TA_2 a H-clearance with respect to locations the call is admissible. TA_2 publishes the security type $H\ proc(X, X\ acc)$. As a result, $price_{Rome}$ has to have a H-clearance with respect to locations while a L-clearance is sufficent for the category "Payment". Also the third web service call **call**(PA_2 : *PayPalServiceGlobal*, $price_{Berlin}$, okP) is admissible when fixing the classification of $price_{Berlin}$ to $[L, L]$. Since the whereabout in Berlin is publicly known, it does not matter that Susan does not trust the service of PA_2 with respect to locations. The variable okP can have an arbitrary clearance as the output of the call demands a clearance higher or equal than $[L, L]$.

Analysing the fourth call **call**(PA_2 : *PayPalServiceGlobal*, $price_{Rome}$, okP), the security checking module reveals the problem that $price_{Rome}$ has to have a H-clearance with respect to locations but Susan does not trust PA_2 with respect to locations (cf. Figure 5). Thus, a call of this web service with the designated parameters would violate the security requirements and therefore the generated plan fails due to security issues. Analysing the failure we see that the constraints on the clearance of $price_{Rome}$ obtained by step 2 and 4 are inconsistent. In order to get a secure plan we have to change step 2 or 4 or both of them. Suppose the called OWLS-MX matchmaker module returns a service *PrivatePaymentZurich* offered by agent PA_1 which is semantically equivalent to the service *PayPalServiceGlobal* of PA_2. Replanning the last step of the plan might result in calling this other payment service to pay the flight to Rome: **call**(PA_2 : *PayPalServiceGlobal*, $price_{Rome}$, okP). Since PA_1 has a H-clearance with respect to locations the call is admissible and the changed plan would pass the security check. As an alternative, suppose that the travel agent TA_2 would offer a special deal that all flights within Europe would cost the same price. Then the price would not depend on the particular destination of the flight and thus the web service would propagate a security type $H\ proc(X, L\ acc)$. No confidential information is revealed by knowing the price of the flight. In this case, $price_{Rome}$ would only require a clearance $[L, L]$ and the original call of PA_2 to pay the flight to Rome would be admissible.

This time the security constraints are all satisfied, and the new service composition plan can be executed as proposed with proven guarantee of privacy preservation.

7 Related Work

Starting with the work of Goguen and Meseguer in the domain of security, information flow control has been subject of a large variety of different approaches introducing different formal notions of independence. Most prominent, McLean [11], Zakinthinos and Lee [19] and Mantel [10] proposed frameworks to embed these different notions in a uniform framework. Our work is based on language-based information flow. The general problem whether a program leaks information from high-level to low-level is undecidable. Thus, type calculi as they are proposed, for instance, in [18] are incomplete. Meanwhile following Volpano and Smith's work, more refined type calculi (e.g. [13]) have been developed that are able to recognize more programs as secure. Since dynamically composed web services are rather simple programs, we decided to use a less refined type calculus, which requires less resources.

Various aspects of security of web services have been investigated. Some aspects were concerned with how to specify a policy in a machine readable and user friendly way at the same time (see e.g. IBM and Micosoft's Web Services security specification [6], especially the WS-Policy part, or REI [12, 7]), how to compose different policies, and how to prove that the web services involved enforce their policy specification for each request. Most of the current approaches to secure service execution concentrate on the proper use of access control mechanisms. As a consequence, any generated service composition plan gets executed anyway, while checking just during execution whether the given access control matrix prohibits any access. If that is indeed the case, the whole execution process is stopped, and a new composition plan has to be created.

These approaches can be classified according to the type of policy they work with. For example, both KAoS [16] and Ponder [3] handle security policies for authentication and obligations, while REI [12] copes with the specification and reasoning with security policies for rights, prohibitions, obligations and dispensations. REI, in particular, is a rich logic-based policy language using rules and constraints to formulate security and privacy policies. However, the REI based approach presented in [7] does not take any information flow aspects into account. As a result, the proposed enforcement of privacy policies is simply a matter of secure communication between web services in terms of agreed encryption protocols. In other words, privacy aspects are assumed to be dealt with by means of cryptographic techniques only. However, this is not enough to ensure the absense of hidden channels, or unintended leakage of information in compiled data. However, the description of our service security policies in terms of logical type calculus expressions could be translated to equivalent but more natural language like expressions to be of use for annotating OWL-S service profiles with corresponding policies. That could be done, for example, by adapting the RDFS syntax of REI as proposed in [7]. However, the details have to be explored in future work.

Finally, we would also like to refer to related approaches that are concerned with the theory of composing security policies independent of the type of the policy [2], and practical extensions resulting in IBM's algebra for composing policies, which is based on their Enterprise Privacy Authentication Language (EPAL) [6], [15].

8 Conclusion

In this paper, we presented an approach to solve the problem of provably secure execution of semantic web service composition plans by means of type based information flow analysis prior to, and during, execution of the plan. While we concentrated on privacy aspects to illustrate our approach it is worth to mention that the same approach can be also used to ensure the integrity of data (based on Biba instead of Bell/LaPadula). Non-repudiation or availability issues are orthogonal to our approach. The integrated components of this approach include our OWL-S service matchmaker, OWLS-MX, the service composition planner, OWLS-XPlan, and the security checker module for formally verifying the compliance of the created composition plan to be executed with given data and service security policies of both service consumer and provider. Our approach can be, for example, considered complementary in part to the one presented in [7] with respect to the abstraction of specification of policies and used means of their enforcement.

References

1. D. E. Bell and L. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report MTR-2997, MITRE, 1976.
2. P. A. Bonatti, S. D. C. di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Trans. Inf. Syst. Secur.*, 5(1):1–35, 2002.
3. N. Dulay, N. Damianou, E. Lupu, and M. Sloman. A policy language for the management of distributed agents. In *Agent Oriented Software Engineering (AOSE-2001)*, pages 84–100. Springer, LNCS 2222, 2001.
4. J. A. Goguen and J. Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy*, pages 11–20, Oakland, CA, USA, 1982. IEEE Computer Society.
5. D. Hutter and M. Volkamer. Information flow control to secure dynamic web-service composition. In *3nd International Confernce on Security in Pervasive Computing*. Springer, LNCS, 2006.
6. IBM and Microsoft. *Security in a Web Service World: A proposed architecture and roadmap*. www-106.ibm.com/developerworks/ webservices/library/ws-secmap, April 2002.
7. L. Kagal, M. Paoucci, N. Srinivasan, G. Denker, T. Finin, and K. Sycara. Authorization and Privacy for Semantic Web Services. *IEEE Intelligent Systems (Special Issue on Semantic Web Services)*, 19(4):50–56, July 2004.
8. M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX. In *5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. ACM Press, 2006.
9. M. Klusch, A. Gerber, and M. Schmidt. Semantic web service composition planning with owls-xplan. In *1st International AAAI Fall Symposium on Agents and the Semantic Web*. AAAI Press, 2005.
10. H. Mantel. Possibilistic definitions of security – an assembly kit. In *2000 IEEE Computer Security Foundations Workshop*, pages 185–199, Cambridge, UK, 2000. IEEE Computer Society.
11. J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1994.
12. A. Patwardhan, V. Korolev, L. Kagal, and A. Joshi. Enforcing policies in pervasive environments. In *Mobile and Ubiquitous Systems, MobiQuitous-2004*, pages 299–308. IEEE Computer Society, 2004.
13. A. Sabelfeld and A. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1), 2003.
14. G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Conference Record of POPL 98: The 25th Symposium on Principles of Programming Languages*, pages 355–364, New York, NY, 1998.
15. W. H. Stufflebeam, A. I. Antón, Q. He, and N. Jain. Specifying privacy policies with P3P and EPAL: lessons learned. In *Workshop on Privacy in the Electronic Society, WPES-2004*, Washington DC, USA, 2004.
16. A. Uszok, J. M. Bradshaw, R. Jeffers, A. Tate, and J. Dalton. Applying KAoS services to ensure policy compliance for semantic web services workflow composition and enactment. In *International Semantic Web Conference*, pages 425–440. Springer, LNCS 3298, 2004.
17. D. M. Volpano and G. Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.
18. D. M. Volpano and G. Smith. A type-based approach to program security. In *TAPSOFT'97: Theory and Practice of Software Development*, pages 607–621. Springer, LNCS 1214, 1997.
19. A. Zakinthinos and E. S. Lee. A general theory of security properties. In *1997 IEEE Symposium on Security and Privacy*, pages 94–102, Oakland, CA, USA, 1997.