Information Flow Control to Secure Dynamic Web Service Composition*

Dieter Hutter and Melanie Volkamer

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

Abstract. The vision of a landscape of heterogeneous web services deployed as encapsulated business software assets in the Internet is currently becoming a reality as part of the Semantic Web. When pro-active agents handle the context-aware discovery, acquisition, composition, and management of application services and data, ensuring the security of customers' data becomes a principle task. To dynamically compose its offered service, an agent has to process and spread confidential data to other web services demanding the required degree of security. In this paper we propose a methodology based on type-based information flow to control the security of dynamically computed data and their proliferation to other web services.

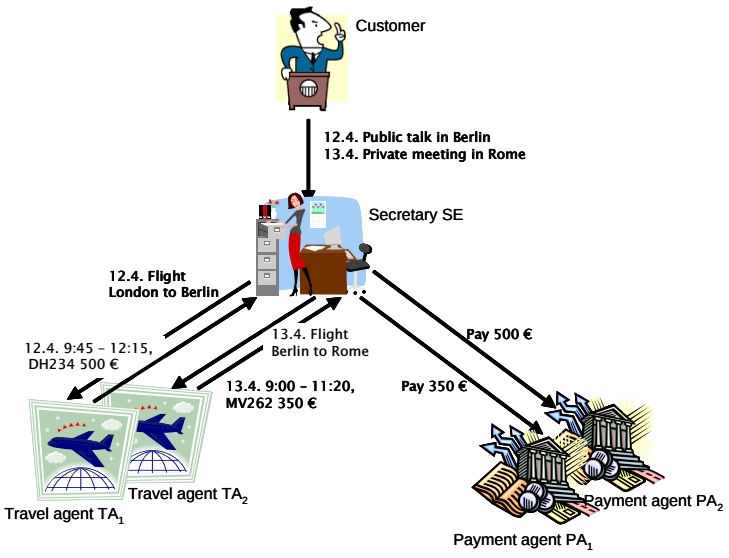
1 Introduction

The proliferation of web services as self-contained web accessible programs and the idea of the Semantic Web of making information *computer-interpretable* enables the dynamic composition of complex services assembled from various individual services and typically distributed over the web. Following the paradigm of pervasive computing, pro-active agents are installed on mobile phones or PDAs operating on the web and handle the context-aware discovery of appropriate services and use AI-based planning techniques to dynamically compose the retrieved service to solve complex tasks. Web services provide formal specifications of their well-defined semantics using languages based on description logics like OWL-S [19] or its predecessor DAML-S [7]. Based on these semantically annotated web services users who want to achieve a specific goal could be assisted by intelligent agents that automatically identify and compose the necessary services.

The introduction of web services in general and dynamic web service composition (e.g. [6, 24, 31, 24, 24, 14]) in particular requires appropriate security facilities to guarantee the security requirements of all participants. Web services have to be protected against misuse of their resources on the one hand, and on the other hand the customers of web services require guarantees with respect to the security (e.g. confidentiality or integrity) of their data. Security policies

* Parts of this work were sponsored by grants from the German Ministry for Technology and Education (BMBF) and the German Science Foundation (DFG)

are used to specify the security requirements of a system. The formalization of security policies is often guided by the idea to protect integrity or confidentiality by controlling the access to locations in which confidential information is stored. This notion of *access control* is typical for most of the approaches concerned with the specification of appropriate security policies for web services (e.g. [20, 8]). This results in a web service centered view of security. These policies describe access rights to web services and the delegation and withdrawal of these access rights. However, they are less suited to formulate the security needs of a customer, i.e. to formalize how web services have to deal with provided customer's information. In contrast, information flow control [17, 32, 16] relies on the idea of modeling confidentiality of data as restrictions on the flow of information between domains of a (global) system. Starting with the work of Goguen and Meseguer [9, 10], the restrictions on information flow have been formalized as independence properties between actions and observations of domains. In this paper we present an approach which uses information flow control techniques to control the distribution of customer's information in web services using security policies formulated and provided by the customer.



A Simple Example. As an example consider a future-oriented politician who uses intelligent web services to organize his traveling in Europe. Living in London, he has to give a public talk in Berlin and he will meet the Italian prime minister the day after in a secret get-together. He informs his web service 'secretary' implemented on his PDA about his intentions. This web service decomposes the problem into organizing the flights between London, Berlin and Rome and making arrangements for payment. The secretary contacts a travel agency

for the desired flights and obtains in return specific flight information and the corresponding prices. Afterward, the secretary selects a payment service which performs the payment of the selected flights.

This example illustrates how the confidentiality of the get-together in Rome enforces the confidentiality of information dynamically calculated by various web services when planning the traveling. To keep the get-together secret, the flight to Rome and the information provided by the travel agent about this flight have to be confidential. If the price of the flight depends on the flight information then also the price has to be confidential. This means that even if the payment service does not obtain any details of the booked flights, the politician has to trust or ensure that the payment service keeps the possible deduced information about his whereabouts confidential. Otherwise one might deduce flight information from the amount of money the service has to pay. The situation changes if the travel agency offers a flat rate for European flights. Then the price of the flights does not depend on the selected destinations and its calculation is independent of specific flight information. In this case the politician does not have to trust the discretion of the payment service with respect to his whereabouts. Furthermore, while our politician trusts his payment service with respect to financial information he may have made bad experiences with payment systems offered by some travel agencies and instructs his payment system not to pay by credit card any service offered by these agencies.

Summing up, our politician has his individual security policy with respect to the confidentiality of his whereabouts and his bank and credit card information. This policy has to be dynamically extended to cope with newly computed or generated information and also with subordinate web services not primarily known to our politician.

Access Control vs. Information Flow. Standard approaches (like, for instance, REI [20], Ponder [8]) are based on access control policies that control the execution of actions on individual objects. However, access control policies suffer from the problem of Trojan Horses or other information leakage using hidden channels. The reason for this is that no control is enforced on the use of the provided data once it is released to an authorized web service. Improper processing of confidential information and calls to subsiderary web services can disseminate secrets to web services that are not authorized to read this information. Mandatory access control might overcome some of these problems by labeling each information and service with some security level. Policies similar to Bell/LaPadula [2] could be used to control read/write operations on data depending on the labeling of data and web services. But still we are faced with two major problems:

First, there is no central authority in the web that is able to fix the security labels of all services and data. Both customers and web services will provide information to accomplish a requested service, and both sides will have individual perceptions about how the provided data may be used. Similar to the dynamic composition of web services there is a need for a dynamic and consistent composition of the related security policies of all participants.

Second, rather than providing statically stored data, web services will search for new ways to *compute* requested information from various data available on the net. The dynamic composition of web service results not only in a dynamic synthesis of programs (actions) for processing the data but also in a dynamic generation of new types of data which have to be dynamically classified according to their stored information.

In this paper we adopt techniques from language-based information flow control to control the secrecy of dynamically created data according to the policies of the involved web services. Each data is equipped with a type specifying its classification with respect to different (user-defined) security categories. In our example such a type would encompass for instance the degree of privacy of the whereabouts and the degree of confidentiality of payment information. Customers formulate their security requirements by attaching types to customer data and to individual web services or classes of web services.

The effects of this typing are twofold. First, web services can only be used for service composition if they provide the necessary clearance for the information they would receive when executing the service. Second, the computation of any low-security data has always to be independent of high-security data. Once a new data is synthesized, it is automatically classified according to the classifications of the data that have been used to compute it.

We start with a closer look on our approach in Section 2. In Section 3 we describe how to generate and specify security policies in our approach. Section 4 introduces a type calculus which is used to compute the security classification of both, newly computed data and composed services. We finish our paper with a comparison of our work with existing approaches in Section 5.

2 Dynamic Web Service Composition

Web services are software components distributed on various hosts and communicating over the web using standard protocols based on XML. Internet description languages are typically used to describe the interfaces. Following the idea of the semantic web [4], the idea of dynamic web service composition (cf. e.g. [30, 3]) is that web services provide their semantics as formal specifications such that other web services incorporating AI-planning algorithms can make use of these web services to solve complex (previously unknown) tasks. Different planning algorithms are proposed to implement dynamic web service compositions (cf. [21] for a survey). For our purposes we are not interested in the concrete planning process but we will operate on (partial) results of such planning processes just before the synthesized plans (or parts thereof) are executed. We consider the result of the planning process as (partial) programs formalized in some "standard" sequential programming language. In general they contain calls to other web services that are basically considered as procedure calls. Data is provided to web services via input parameters and returned by the web service via output parameters.

We distinguish between *atomic* and *composed* web services. While an atomic web service provides the service defined in its formal specification without requesting other web services, a composed web services will distribute subproblems to other web services. The execution of a web service can be considered as a tree: all leaves are atomic web services while inner nodes are compound web services. Plans have to be synthesized for each individual service request because in the setting of agent-based provision of web services there is no static global system but instead it is constantly changing by accretion and leave of individual agents and web services.

In order to satisfy a requested service, a web service receives sensitive data together with a security policy which is basically formulated by the customer and which guides the usage of data with respect to other web services. On the one hand it classifies the privacy of the provided data according to different information classes (like for instance traveling or payment information). For example, our politician rates his get-together in Rome as confidential while his talk in Berlin is public. On the other hand the security policy formalizes the trust a customer has in individual web services or families of web services by assigning corresponding clearances (split into the different classes of information) to web services. Then before actually calling a web service, it has to be checked that its clearance is sufficiently high to obtain the classified data. The called web service inherits the customer's security policy and extends it with respect to the clearance of previously unknown web services or the classification of data computed by the web service. A web service is not allowed to change the classification of provided data. Downgrading sensitive data would violate the security requirements of the customer but also upgrading data is typically useless because the data could have been already disseminated to web services which do not possess the newly required clearances.

Web services implement a bi-directed communication, which is important for policy descriptions: The web service receives a request including sensitive data from its caller and sends data back, which may also be sensitive. Hence, web services have to classify their newly computed data according to the security policies of web services and customers providing the processed information. A web service implementing an interface to a database may provide, for instance, its data only to web services with special clearances. Web services that process only data provided by the customer have to classify the computed data according to the classification of the used customer's data.

3 Security Policies for Dynamic Composition

As mentioned in the introduction we adopt the notion of non-interference to formulate confidentiality as independence between data. The low-security data must not depend on any high-security data. As a consequence, actions affecting low-security data are only allowed if they only access low-security data. More generally, the classification of any computed/synthesized information has to be at least as high as the classifications of all used data; i.e. no secret bit of information

must be disclosed in public information. Since the way information is assembled is dynamically planned in web services, also the classification of information has to be done dynamically during the plan construction and execution. In the following we adopt a type calculus developed by Volpano and Smith (e.g. [28, 29, 25]) to encode classifications as types and use rules of the type calculus to propagate the types of data along the composition of a given plan.

The classification of data determines the accessibility of this information by individual web services. In order to receive classified information, web services must provide a clearance that is at least as high as the classification of the data. However, in contrast to mandatory access control, there is no central authority that assigns clearances to individual web services but each individual customer has its own perception about the security categorization of individual or families of web services. Thus besides the classification of provided information, a customer has also to provide rules that allow invoked web services to assess the clearance of potential subcontractors and to decide whether this clearance is sufficiently high to deal with the data required to perform the service.

Clearances and classifications are formalized with the help of standard information flow policies. Such a policy is a lattice $(SC, \leq)$ where SC is a finite set of *security classes* that is partially ordered by $\leq$. In its simplest form SC might be a set containing two elements, e.g. H and L denoting secret and public, but we can easily encode also integrity classes (cf. [29] for an example) to differentiate between trusted or untrusted information.

As illustrated in our example we want to subdivide the clearance of a web service according to different types of information. For instance, we may want to classify a travel agency as being high with respect to our whereabouts but low with respect to our financial situation. Therefore, we combine different information flow policies $(SC_1, \leq_1), \ldots, (SC_n, \leq_n)$ to a *composed* flow policy $(SC, \leq)$ by $SC = SC_1 \times \ldots \times SC_n$ and $\langle \tau_1, \ldots, \tau_n \rangle \leq \langle \tau'_1, \ldots, \tau'_n \rangle$ iff $\tau_i \leq \tau'_i$ holds for all $1 \leq k \leq n$. Least upper bound (and greatest lower bound, respectively) are computed by the least upper bounds (and greatest lower bounds, respectively) of each component.

Once a customer charges a web service with some task he provides (partially) classified information to the web service. For instance, our politician tells his secretary about his confidential gathering in Rome and that it has to be kept secret to some extent. The potential distribution of this data to other web services is regulated by (i) the classification of the provided data (being, for instance, secret) and (ii) the clearances the customer assigns to the web services with respect to this type of information. Let $(SC, \leq)$ be a (compound) information flow policy, then we call a partial mapping σ_{ws} from the set of web services to SC a *web service clearing* wrt. $(SC, \leq)$.

The dynamic nature of web service composition results in the problem that typically a customer can neither know all subjects (i.e. web services) that will be involved in his request nor anticipate all types of information that will be communicated between the web services. As a consequence, his initially formulated security policy will cover only a fraction of the involved subjects and data

and has to be extended accordingly on the fly by involved web services. We will sketch this *conservative* extension of security policies in the following.

Since a customer usually does not know about all available web services, the web service clearance σ_{ws} provided by the customer may be undefined for some unknown web services WS. Suppose that another web service WS' acting on behalf of the customer creates a plan involving the provision of confidential data to WS. Since the plan violates the security requirements (we assume that unknown services have no clearences at all), it would be rejected when checked by the type calculus presented in Section 4. However, we can incorporate a delegation mechanism that allows the customer to delegate a web service WS' the right to extend σ_{ws} to a new mapping σ'_{ws} that coincides with σ_{ws} in all its defined values (i.e. $\sigma_{ws}(x) = \sigma'_{ws}(x)$ for all $x \in \mathit{domain}(\sigma_{ws})$). Therefore, the customer provides a *delegation classification* that is again a partial mapping σ_{del} from the set of web services to SC. $\sigma_{del}(WS)$ denotes the maximal clearance a web service WS may allocate to an unknown web service. Let $\bot$ be the bottom element of SC then $\sigma_{del}(WS) = \bot$ denotes that WS is not allowed to classify any previously unknown web service. Let $\top$ be the top element in SC, then $\sigma_{del}(WS) = \top$ gives WS full discretionary power with respect to the classification of previously unknown web services. Notice however, that once the customer or some web service with appropriate delegation rights has fixed the clearance of a web service it cannot be changed anymore.

As mentioned before, we consider SC as a set of tuples $\tau_1, \ldots, \tau_n$. Each tuple reflects the classification or clearance with respect to different security categories or aspects like, for instance, location information or payment details. Web services, like web interfaces to data bases, may act as data sources and formulate their own security requirements for the newly provided data. Besides classifying the new data according to the given categories they can introduce new security categories by introducing a new lattice $(SC', \leq')$ operating on tuples $\langle \tau_1, \ldots, \tau_n, \tau_{n+1}, \ldots, \tau_{n+m} \rangle$ such that for all tuples in SC': $\langle \tau_1, \ldots, \tau_{n+m} \rangle \leq' \langle \tau'_1, \ldots, \tau'_{n+m} \rangle$ implies $\langle \tau_1, \ldots, \tau_n \rangle \leq \langle \tau'_1, \ldots, \tau'_n \rangle$ and additionally, $\langle \tau_1, \ldots, \tau_n \rangle \leq \langle \tau'_1, \ldots, \tau'_n \rangle$ implies $\langle \tau_1, \ldots, \tau_{n+m} \rangle \leq' \langle \tau'_1, \ldots, \tau'_n, \tau_{n+1}, \ldots, \tau_{n+m} \rangle$.

Analogously, the web service conservatively extends the mappings σ_{ws} and σ_{del} to σ'_{ws} and σ'_{del} which now operate on SC' instead of SC but are identical to the former mappings with respect to all previously existing categories, i.e. $\exists \tau_{n+1}, \ldots, \tau_{n+m} : \sigma'_{ws}(WS) = \langle \tau_1, \ldots, \tau_{n+m} \rangle$ iff $\sigma_{ws}(WS) = \langle \tau_1, \ldots, \tau_n \rangle$ for all web services WS; and $\exists \tau_{n+1}, \ldots, \tau_{n+m} : \sigma'_{del}(WS) = \langle \tau_1, \ldots, \tau_{n+m} \rangle$ iff $\sigma_{del}(WS) = \langle \tau_1, \ldots, \tau_n \rangle$ for all web services WS. All data provided by the call of such a web service are classified as $\bot$ with respect to newly introduced categories. Otherwise the web service could easily block information provided freely by the customer if it introduces a new category, classifies the provided data as high with respect to this category, and gives no clearance for this category to any web service. Similarly, we can *restrict* mappings σ_{ws} and σ_{del} by removing particular categories.

Example revisited. Let us consider the example we presented in the introduction. The first step in this example is the specification of σ_{ws} by the customer. The

customer selects two categories *location info* and *payment info* with H and L as potential values to formulate his security policy. Location info relates to informations about his whereabouts while payment info covers information about the details of his bank accounts or credit cards. He rates the web services as follows. Since he trusts in his secretary with respect to both categorizes, he chooses $\sigma_{ws}(SE) = \langle H, H\rangle$. However, he does not trust the travel agencies with respect to all his payment informations: $\sigma_{ws}(TA_i) = \langle H, L\rangle$ for $i \in \{1, 2\}$. Contrarily, he trusts both payment agents, paying for him the bills with respect to different accounts, in all his payment information. However, he only trusts the first agent in keeping also his whereabouts secret. The reason might be that the corresponding bank account belongs to some public agency and cash audits may reveal the details of the trip. Thus, we obtain $\sigma_{ws}(PA_1) = \langle H, H\rangle$ and $\sigma_{ws}(PA_2) = \langle L, H\rangle$.

4 Type Calculus To Enforce Security Policies

In the last section we described the mechanism to synthesize a common security policy for all participating web services that is consistent with the initially formulated policy of the customer. This section illustrates how a web service can prove whether the execution of its synthesized plan would violate the requested security policy and how it can classify newly computed data according to the given information flow policy.

Our approach is based on the work of D. Volpano and G. Smith [29, 25] on using type systems to secure information flow. Its underlying idea is as follows. We assume that a program has low-level and high-level inputs and computes some low-level and high-level outputs. Such a program is considered secure if the low-level output of the program does not depend on the high-level input. The basic idea is to monitor the data flow of high-level input with the help of a type system and prove that no high-level input was used to compute any low-level output.

Therefore, types are attached to all constructs in the programming language that store or provide information (like, for instance, locations, parameters or variables). Typing rules are formulated that specify the consequences to the involved types once any particular statement of the programming language is executed.

The programming language used in [29] is a simplified sequential programming language which provides conditionals, loops, and procedures. We extend this language by the additional feature of calling web services WS (in a blocking mode) by $\mathbf{call}(WS, x, y)$. Figure 1 presents the syntax of this language. Op and R are just place holders for the various functions and relations built into the language. Examples are $+$, $-$, $<$, and $=$. Metavariables x and P ranges over identifiers, l over locations and n over numerical literals.

The security classes SC of the underlying information flow policy $(SC, \leq)$ constitute the set of so-called data types which are used to classify data. Data are represented by expressions in the programming language. Hence, all expressions $Expr$ possess a data type τ denoting their classification. Intuitively, the type of

(Expr) $e ::= x \mid P \mid n \mid l \mid Op(e, e') \mid eRe' \mid \mathbf{proc}(\mathbf{in}\ x, \mathbf{out}\ y)\ c$
(Comm) $c ::= x := e' \mid c; c' \mid P(e, e') \mid \mathbf{call}(WS, x, y) \mid \mathbf{while}\ e\ \mathbf{do}\ c$
$\mathbf{if}\ e\ \mathbf{then}\ c\ \mathbf{else}\ c' \mid \mathbf{letvar}\ x := e\ \mathbf{in}\ c \mid$
$\mathbf{letproc}\ P\ (\mathbf{in}\ x, \mathbf{out}\ y)\ c\ \mathbf{in}\ c'$

Fig. 1. Syntax of the language

Expr has to be equal to or higher than the least upper bound of the security types of all information we have used to evaluate *Expr*. For instance, given two expressions e and e' then $Op(e, e')$ obtains the least upper bound of the security types of e and e' as its security type.

Type calculus rules are used to propagate data types along expressions. The following rule illustrates this for expressions constructed with the help of a function *Op*. γ represents the assignment of the variables and is used to map occurring variables to their types.

$$\frac{\gamma \vdash e : \tau, \gamma \vdash e' : \tau}{\gamma \vdash Op(e, e') : \tau}\ Op$$

The rule makes use of an implicit type coercion: $\gamma \vdash e : \tau$ and $\tau \leq \tau'$ implies $\gamma \vdash e : \tau'$ ("upgrading data"). Hence we can always upgrade implicitly the type of e and e' to its least upper bound τ in order to apply the rule.

Variables are used to store information. They need a sort of clearance to hold classified information. This is denoted by a phrase type $\tau\ acc$. $\gamma \vdash x : \tau\ acc$ says that x has the clearance to house data with a classification up to τ. Again there is a contravariant type coercion by $\gamma \vdash e : \tau\ acc$ and $\tau' \leq \tau$ implies $\gamma \vdash e : \tau' acc$ to simplify the following typing rule for assignments.

$$\frac{\gamma \vdash x : \tau\ acc, \gamma \vdash e : \tau}{\gamma \vdash x := e : \tau\ cmd}\ Assign$$

A phrase type $\tau\ cmd$ is assigned to a program fragment, like a statement or a block of statements, to store information about the security types of changed variables inside the fragment. If a fragment has type $\tau\ cmd$ then only variables with clearances τ and higher will be changed inside the block. The idea is that the execution of the fragment does not assign to variables of type $\tau'\ var$ for τ' lower than τ. It is thus permissible for the fragment to be executed conditionally depending on the values of variables labelled τ or lower without leaking information from the context into lower graded variables.

Variables play two roles: first, they provide data and have a classification τ and second, they store information and need a clearance $\tau\ acc$. Both notions are integrated into a phrase type $\tau\ var$ combining the classification $\tau\ cmd$ and the clearance $\tau\ acc$.

[29] provide also types and typing rules for procedures which we simplified for our purposes (since we omit in-out parameters). A phrase type $\tau\ proc(\tau_1, \tau_2\ acc)$ indicates that the call of the procedure has type $\tau\ cmd$, the input parameter has the classification τ_1 and the output must have the clearance $\tau_2\ acc$. $[e'\backslash P]e$

denotes the capture-avoiding substitution of e' for all free occurrences of P in e. Then the rules for declaring (polymorphic) procedures are as follows.

$$\frac{\begin{array}{l}\gamma \vdash \mathbf{proc}(\mathbf{in}\ x_1, \mathbf{out}\ x_2)c : \pi, \\ \gamma \vdash [\mathbf{proc}(\mathbf{in}\ x_1, \mathbf{out}\ x_2)c\ /\ P]c' :\ \tau\ cmd\end{array}}{\gamma \vdash \mathbf{letproc}\ P(\mathbf{in}\ x_1, \mathbf{out}\ x_2)\ c\ in\ c' : \tau\ cmd}\ LetProc$$

$$\frac{\gamma[x_1 : \tau_1,\ x_2 : \tau_2\ acc] \vdash c : \tau\ cmd}{\gamma \vdash \mathbf{proc}(\mathbf{in}\ x_1, \mathbf{out}\ x_2)c :\ \tau\ proc(\tau_1, \tau_2\ acc)}\ Proc$$

The rule for typing the application of procedures is straightforward:

$$\frac{\begin{array}{l}\gamma \vdash P :\ \tau\ proc(\tau_1, \tau_2\ acc), \\ \gamma \vdash e_1 : \tau_1, \\ \gamma \vdash e_2 : \tau_2\ acc\end{array}}{\gamma \vdash P(e_1, e_2) : \tau\ cmd}\ ApplyProc$$

The idea of our approach is that web service calls can be treated like procedures. We consider web services as external procedures. We can encode global states as global variables common to various web services. In contrast to procedures, web services have to be first class citizens in our approach possessing their individual clearances. The call of a web service has to be guarded by a check whether the input to be provided to the service lies within its clearance. Thus we introduce a new phrase type $\tau\ proc(\tau_1, \tau_2)\ \tau'$ to type web services. First, the phrase denotes that the call of the web service has type $\tau\ cmd$. This information is only required in the presence of common variables (global states) of calling and called web services. Second, the input has the classification τ_1 and the resulting output requires the clearing τ_2. Additionally, τ' defines the clearance of the web service from the caller's point of view. It is crucial to understand the difference between the classification τ_1 of the input parameter and the clearance τ' which the calling web service assigns to the called web service. In general, web services are polymorphic in their types, for example, τ_1 and τ_2 are type variables that are related by the constraint $\tau_1 = \tau_2$. In this case the output would 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. Thus τ_1 and τ_2 reflect the constraints on the security types considering the computation inside the called web service while τ' reflects the trust the calling web service has in the called web service.

The rule for typing the call to the web services is similar to the corresponding rule for procedure calls:

$$\frac{\begin{array}{l}\gamma \vdash\ WS :\ \tau\ proc(\tau_1, \tau_2\ acc)\tau', \\ \gamma \vdash e_1 : \tau_1, \\ \gamma \vdash e_2 : \tau_2\ acc \\ \gamma \vdash \tau_1 \leq \tau' \\ \gamma \vdash \tau_2 \leq \tau'\end{array}}{\gamma \vdash \mathbf{call}(WS, e_1, e_2) : \tau\ cmd}\ ApplyWS$$

There is, however, one difference between a sub web service call and a procedure call: while we know the body of a procedure being part of the program itself, we do not know the interior of called web services because firstly, there is no fixed program so no fixed body can be exported as part of the specification of a web service, and secondly because in general owners of web services do not want to disclose the source code of their web services. Instead a web service will export its security type (considered as a procedure) $\tau\ proc(\tau_1, \tau_2\ acc)$ as part of its overall specification. Given the inherited security policy σ_{ws} of the customer and the published security type $\tau\ proc(\tau_1, \tau_2\ acc)$ of a suitable web service WS' to be called by a web service WS, WS can assemble the type of WS' to $\tau\ proc(\tau_1, \tau_2\ acc)\ \sigma_{ws}(WS')$. Contrarily, the called web service requires the type of each parameter. Hence, the calling web service provides the security type of each parameter as intrinsic part of a call.

Example revisited. Let us consider the example we presented in the introduction. In Section 3 we illustrated already how he customer chooses his security policy. In a next step the customer instructs his secretary agent SE to organize his trip to Berlin and Rome. The trip to Berlin is not classified at all, i.e. $\langle L, L\rangle$ while the second trip contains confidential location information: $\langle H, L\rangle$. The secretary agent constructs a plan of how to decompose the task into a sequence of web service calls and comes up with the following program:

$$\begin{aligned}
&\mathbf{call}(TA_1, \mathit{flight}_{Berlin}, \mathit{price}_{Berlin});\\
&\mathbf{call}(TA_2, \mathit{flight}_{Rome}, \mathit{price}_{Rome});\\
&\mathbf{call}(PA_1, \mathit{price}_{Berlin}, okP);\\
&\mathbf{call}(PA_1, \mathit{price}_{Rome}, okP);
\end{aligned}$$

Both travel agents publish $H\ proc(X, X\ acc)$ (X being a type variable) as their specification of their security types. This means that the output (the price of the flight) requires the same security class as the input (the flight requirements). It reflects the fact that the price is calculated with the help of the flight information. Suppose, a travel agent would offer a flat rate for European flights, i.e. each flight in Europe would cost the same amount of money, then he could publish a security type $H\ proc(X, L\ acc)$. Both payment agents specify $H\ proc(X, L\ acc)$ as their security types since we assume that their acknowledgments okP do not depend on the prices.

Based on these typings, the type calculus rates price_{Berlin} as L wrt. location information and price_{Rome} as H. As a consequence, the secretary agent cannot use PA_2 to pay the flight to Rome because it only has a L-clearance with respect to location information. If we suppose that the secretary makes use of a flat rate-offer then price_{Rome} would be rated as L which would enable the use of PA_2 to pay the flight.

5 Related Work

The dynamic composition of web services is a quite new research area coming up within the last four years. [21] gives a survey about the different AI-planning approaches to tackle dynamic composition. Since our approach is independent of the way a plan is generated we will not go into the details of these approaches.

With the advent of pervasive computing, a lot of difficulties arouse with respect to the transmitted, stored, used and computed sensitive data. There are various approaches concerned with trust management (e.g. [12, 13]) and authentication techniques, i.e. how to ensure that a web service does not try to cheat [22] because cheating would not gain any benefits in the underlying economic model.

Various aspects of security policy 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 [11], especially the WS-Policy part), how to compose different policies and how to prove that the web service does ensure its policy specification with each request. Current approaches concentrate on access control. So the plan is executed in any case and if the access control matrix forbids any access during the execution, it stops and a new plan has to be created. The approaches can be distinguished with respect to the type of policy they work with: e.g. KAoS [27] and Ponder [8], which handle security policies for authentication and obligations, or REI [20], which works with security policies for rights, prohibitions, obligations and dispensations. There are also two approaches which concentrate on the composition of security policies independent of the type of the policy: The main idea, Samarati et al. present in [5] and a practical extension is introduces with IBM's algebra for composing policies based on their Enterprise Privacy Authentication Language (EPAL) [11], [26].

Starting with the work of Goguen and Meseguer, information flow control has been subject of a large variety of different approaches introducing different formal notions of independence. Most prominent, McLean [18], Zakinthinos and Lee [32] and Mantel [16] 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 [29] are incomplete. Meanwhile following Volpano and Smiths work, more refined type calculi (e.g. [23]) have been developed that are able to recognize more programs as secure. They are, for instance, able to detect if different program paths will result in the same low-level results. In this case the condition whether to take the one path or the other can freely use high-level inputs. However, the power of these approaches result in a more expensive computation and propagation of types. Since dynamically composed web services are rather simple programs, we decided to use a less refined type calculus, which require less resources.

6 Conclusion

We presented an approach to use language-based information flow control to ensure the confidentiality (as well as integrity) of user's data provided to dynamically composed web services. The data flow of confidential data is monitored by a type calculus which propagates the security requirements of the user along the planning and execution of dynamically composed web services.

Future work concern the development of an appropriate language to encode the security requirements σ_{ws} of a customer. Possible solutions are based on the use of appropriate description logics [1]. Moreover, we will analyze the application of negotiation mechanisms in case the web service does not fit with the policy of a web service which should be called. Here the SLAng approach [15] might be helpful.

Another problem in this context arises if the web service executes its plan but one of the chosen web services is not available anymore. An easy solution would be to try to find another web service with the same policy and functional specification. But if the web service cannot find such a web service, we do not want to start from scratch again by creating a new plan for the whole task, but we would like to patch the existing plan by creating only a plan for the changed sub tasks. In order to come up with such a solution we have to investigate how the change of the security types of some variables or parameters will effect existing proofs of the type calculus.

Acknowledgements

We are grateful for many fruitful and inspiring discussions that we had with our former colleague Axel Schairer on this work in general and on earlier versions of this paper in particular. We would also like to thank the anonymous referees for their helpful feedback on this work.

References

1. Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the semantic web. In Dieter Hutter and Werner Stephan, editors, *Mechanizing Mathematical Reasoning, Festschrift in Honor of J.H.Siekmann*. Springer-Verlag, LNCS 2605, 2005.
2. D. E. Bell and L. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report MTR-2997, MITRE, 1976.
3. V. Richard Benjamins, Enric Plaza, Enrico Motta, Dieter Fensel, Rudi Studer, Bob Wielinga, Guus Schreiber, and Zdenek Zdrahal. Ibrow3 - an intelligent brokering service for knowledge-component reuse on the world wide web. In *11th Knowledge Acquisition for Knowledge-Based System Workshop (KAW98)*, 1998.
4. T. Berners-Lee, J. Hendler, J., and O. Lassila. The semantic web. *Scientific American*, May 2001.
5. P.A. Bonatti, S. De Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security*, 5(1):1–35, 2002.

6. J. Bryson, D. Martin, S.I. McIlraith, and L.A. Stein. Agent-based composite services in DAML-S: The behavior-oriented design of an intelligent semantic web. In Ning Zhong, Jiming Liu, and Yiyu Lao, editors, *Web Intelligence*. Springer Verlag, 2002.
7. DAML-S DARPA agent markup language for services, version 0.9. http://www.daml.org/services/daml-s/0.9/daml-s.html.
8. Naranker Dulay, Nicodemos Damianou, Emil Lupu, and Morris Sloman. A policy language for the management of distributed agents. In *Agent Oriented Software Engineering, AOSE*, pages 84–100. Springer, 2001.
9. J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20, Oakland, CA, USA, 1982.
10. J. A. Goguen and J. Meseguer. Inference Control and Unwinding. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 75–86, Oakland, CA, USA, 1984.
11. 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.
12. L. Kagal, T. Finin, and A. Joshi. Trust based security for pervasive computing enviroments. *IEEE Computer*, 24(12):154–157, December 2001.
13. L. Kagal, T. Finin, and A. Joshi. Developing secure agent systems using delegation based trust management. In K. Fischer and D. Hutter, editors, *Security of Mobile MultiAgent Systems (SEMAS 02) held at Autonomous Agents and MultiAgent Systems (AAMAS 02)*, 2002.
14. M. Klusch, A. Gerber, and M. Schmidt. Semantic web service composition planning with owls-xplan. 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web, 2005.
15. D. Davide Lamanna, James Skene, and Wolfgang Emmerich. Slang: A language for defining service level agreements. In *IEEE Workshop on Future Trends of Distributed Computing Systems, FTDCS*, pages 100–, 2003.
16. H. Mantel. Possibilistic Definitions of Security – An Assembly Kit. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 185–199, Cambridge, UK, 2000.
17. J. D. McLean. Proving Noninterference and Functional Correctness using Traces. *Journal of Computer Security*, 1(1):37–57, 1992.
18. J.D. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1994.
19. OWL ontology web language, w3c standard technical recommendation. http://www.w3.org/TR/2003/WD-owl-ref-20030331/.
20. Anand Patwardhan, Vlad Korolev, Lalana Kagal, and Anupam Joshi. Enforcing policies in pervasive environments. In *MobiQuitous*, pages 299–308, 2004.
21. Joachim Peer. Web service composition as ai planning - a survey. Technical report, University of St. Gallen, March 2005.
22. S.D. Ramchurn, D. Huynh, and N.R. Jennings. Trust in multi-agent systems. *The Knowledge Engineering Review*, 19(1):1–25, 2004.
23. A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1), 2003.
24. M. Sheshagiri, M. desJardins, and T. Finin. A planner for composing services described in DAML-S. In *Proceedings of AAMAS 2003 Workshop on Web Services and Agent-Based Engineering*, 2003.

25. Geoffrey Smith and Dennis Volpano. Secure information flow in a multi-threaded imperative language. In *Conference Record of POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California*, pages 355–364, New York, NY, 1998.
26. William H. Stufflebeam, Annie I. Antón, Qingfeng He, and Neha Jain. Specifying privacy policies with p3p and epal: lessons learned. In *Workshop on Privacy in the Electronic Society, WPES*, page 35, 2004.
27. Andrzej Uszok, Jeffrey M. Bradshaw, Renia Jeffers, Austin Tate, and Jeff Dalton. Applying kaos services to ensure policy compliance for semantic web services workflow composition and enactment. In *International Semantic Web Conference*, pages 425–440, 2004.
28. Dennis M. Volpano and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.
29. Dennis M. Volpano and Geoffrey Smith. A type-based approach to program security. In *TAPSOFT*, pages 607–621, 1997.
30. R. Waldinger. Deductive composition of web software agents. In *NASA Workshop on Formal Approaches to Agent-Based Systems*. Springer-Verlag, LNCS 1871, 2000.
31. D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S web services composition using SHOP2. In *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, pages 20–23, Sanibel Island, Florida, USA, October 2003.
32. A. Zakinthinos and E. S. Lee. A General Theory of Security Properties. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 94–102, Oakland, CA, USA, 1997.