Information Flow Analysis Based Security Checking of Health Service Composition Plans[1]

Dieter Hutter, Matthias Klusch, Melanie Volkamer

Deduction and Multiagent Systems
German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, 66123 Saarbrücken
{ hutter, klusch, volkamer}@dfki.de

Abstract: In this paper, we present an approach to solve the problem of provably 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. We demonstrate this approach by means of its application to a use case scenario of health service composition planning.

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 range from secure communication via protection of services against misuse to the preservation of user data privacy. 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 sub-services 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

[1] This work has been carried out within the basic research project SCALLOPS funded by the German ministry of education and research (BMB+F); www.dfki.de/scallops.

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 with 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?

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 programming 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 OWLS 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 section 4 describes the information flow based security check in more detail. We demonstrate this approach by means of its application to the motivating use case scenario in section 5. Related work is briefly discussed in section 6; and we conclude in section 7.

2 Use Case Scenario

Suppose Peter Miller is suffering from a cold and aids, and wants to keep this fact private as good as it gets, that is no one whom he does not trust shall know about his disease, in particular not that he is suffering from aids. Peter calls his personal medical agent (MED) via mobile phone, requesting it to purchase appropriate medicine online at a relevant online pharmacy in the web. The required drug for treatment of his disease has to be a special one in the sense that it has to take both, cold and aids into account at the same time. As to the provision of web services, we assume that one or multiple web services are encapsulated and deliberately provided by so called service agents. As a consequence, the Peter's personal agent MED first has to discover the set of available online pharmacy service and payment service agents. It then selects the most relevant pharmacy agent to obtain information on the appropriate drug and its price, and then to make the corresponding payment online.

However, the resulting service composition plan (Figure 1) to be jointly executed by the agents involved still has to be checked whether it complies with Peter's data security policies. For example, the pharmacy agent MA_1 could disclose the type of Peter's disease to other agents. If the price to be announced to the payment agent PA_2 depends on the drug and this drug discloses the disease, PA_2 could indirectly deduce Peter's disease from the order for payment including the price and the identity of Peter Miller as purchaser. Two solutions to this problem are obvious. Either the price of the drug does not reveal the type of the drug (resulting in some kind of "flatrate" for medicine) or Peter has to trust the payment agent. Both solutions result in the fact that the sensibility of the price information is appropriate to the trust we have in the payment agent. In order to verify this appropriateness we use information flow analysis to verify that sensible information will flow only to agents with appropriate clearance. We will explain this approach in more detail in subsequent sections, and then apply it to solve the privacy problem of this use case scenario.

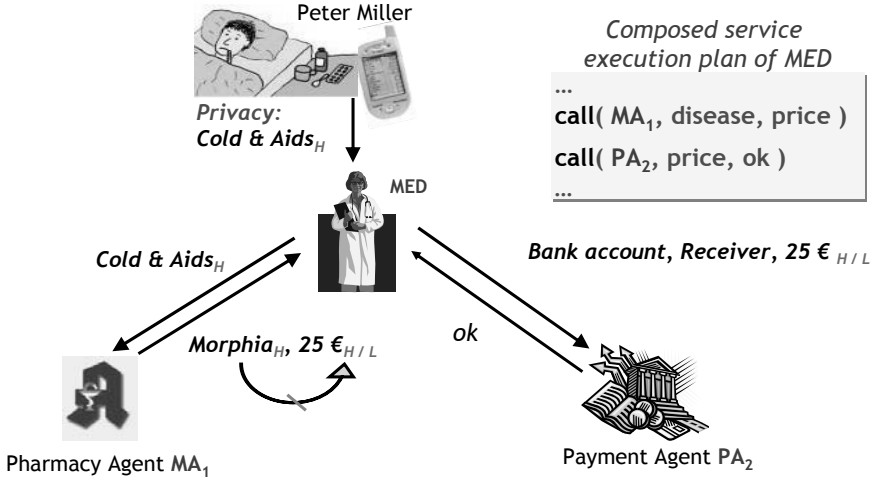

Figure 1: Composed service execution plan with service providing agents involved.

3 Conceptual Architecture

The basic conceptual architecture of the personal secure service composition planning and execution agent (SCPA), called MED in the use case scenario, 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 [8]. In addition, it collects the corresponding service security policies published by the respective service provider agents.

If the matchmaker module finds equivalent services to the query, it directly passes the top ranked one 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 [9]. The planner then converts both the request and all OWL-S services retrieved by the matchmaker into an initial state and goal ontology written in PDDXML, and generates a sequential service composition plan that satisfies the goal.

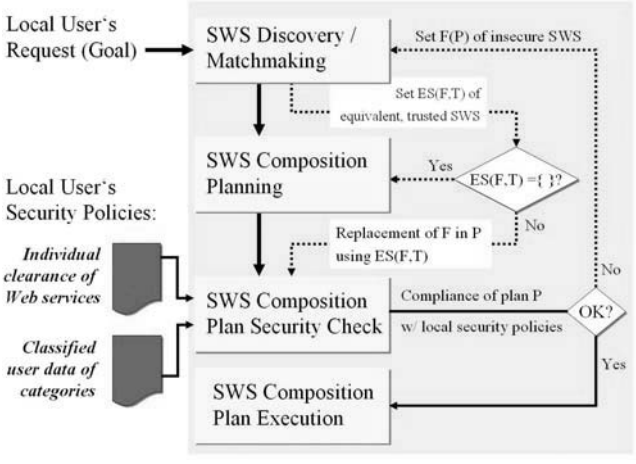

Figure 2: Conceptual architecture of the secure service composition planning agent

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, or not.

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 the plan 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(P) 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 re-planning. 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. If some service is not trusted with respect to keeping data on location data private, hence does not have a clearance for receiving it, it does neither obtain the private data, nor information on the clearance of other services with respect to this particular category. As a consequence, untrusted service agents cannot even know which agent to persuade of revealing private data at all, or of certain 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 Security Checking of Service Composition Plans

4.1 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 *H* and *L* denoting high confidentiality and public release, respectively - with respect to a particular category, like location or payment information.

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 information categories of data used when invoking a web service may change while it does process the request. That is, any service may introduce a new category and classify the new data also with respect to this new category. This way, any 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.

4.2 Type-based information flow analysis

Web services provided by service agents 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 these restrictions for web services, we adopt a security type calculus developed by Volpano and Smith [17, 18, 14] in order 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 assignment 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 is of 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 modelled 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, each web service *WS*(*x, y*) with input parameter *x* and output parameter *y* propagates a security type τ *proc*(τ_1, τ_2). τ_1 is the classification of the input *x*, i.e. WS promises to send or assign *x* only to services or variables which possess a clearance higher or equal than τ_1. τ_2 describes the resulting classification of the output of *WS*. On the one hand τ_2 is determined by the flow of information from *x* to *y* and the type τ_1. If *y* contains information about *x* then τ_2 has to be at least as high as τ_1. On the other hand τ_2 incorporates also the requirements of *WS* on potential receivers of *y* with respect to any information originally contributed by *WS* in order to compute *y* (for instance, the pharmacy shop might not want that its special prices become publicly known and encodes this restriction in an additional role of τ_2). To prevent indirect information flow, i.e. altering publicly available data depending on the value of confidential data, a web service also exports a type τ specifying the minimal clearance a web service must possess to be able to observe the run of *WS*. 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 τ *proc*(τ_1, τ_1)

would simply indicate that the output requires the same clearance as the classification of the input. An example would be a web service that simply copies its input to the output.

4.3 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 some trusted web service to fix the clearance of web services unknown to the user. 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.

5 Application to the Use Case Scenario

Let us return to our use case of ordering and paying Peter's medicine. To guarantee Peter's privacy in this example, Peter has to classify his personal data he provided to his personal agent MED. This is done with respect to different information categories (or roles). One role "Medicine" is related to the privacy of his disease while another one "Payment" regulates the confidentiality of his payment account data. In the same way Peter has to assess the clearance of known web services with respect to different roles. Comparing the clearance of a web service with the classification of a particular data determines whether the web service gets access to this data. For example, since the pharmacy agent MA_2 has only a clearance *L* ("low") in the role "Medicine", a call of MA_2 with the input data "Cold & Aids" being classified as *H* ("confidential") is prohibited. Figure 3 illustrates Peter's settings in our example.

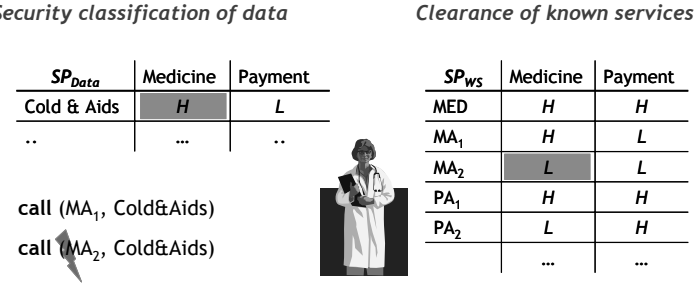

Figure 3: User data security classification, and clearance of known services

Typically, a service composition plan created by MED may also contain services that are unknown to Peter in advance. Hence, Peter can delegate individual web services the right to assess the clearance of an unknown web service with respect to a particular role. Depending on the needs we can model either non-transitive or transitive delegation rights. Once the service plan has been composed, the web service agent MED has to check this plan with respect to the classification and clearances of data and involved services. In order to decide whether a web service call in a plan is admissible, it has to compare the classification of the parameter *x, y* of an intended call *WS(x, y)* with its own security requirements for *WS* specified by the clearance for WS. While the parameter "Cold & Aids" sent to the pharmacy agent has been already rated by Peter himself, the agent MED has to deduce the classification of the data used as input for the call to the payment agent by itself. This classification reflects the confidentiality of the information used to compute the data. Our agent makes use of the security type calculus to compute the classification of such data according to the way it was computed.

For example, Figure 4 presents two different security policies of the pharmacy agent. In the right case the price does not reveal any information about the ordered medicine and thus the required output clearance is *L* while in the left case the price reveals information about the medicine and thus the price is rated as *H* with respect to the role "Medicine".

SP_{Data}	Medicine	Payment
Cold & Aids	*H*	*L*
Price	*H*	*L*
..	...	..

Price depends on the disease (H)

SP_{Data}	Medicine	Payment
Cold & Aids	*H*	*L*
Price	*L*	*L*
..	...	..

Price is a flat rate, hence does not depend on the disease (L)

Figure 4: Security classification of user data based on their dependencies

The type calculus propagates this requested clearance for the output ("price") of the pharmacy agent as a required clearance of the input of the call to the payment agent. This means that in the right case of Figure 4, the MED agent can call any payment agent with a *L* clearance while in the left case an agent with *H* clearance is required. In the same way MED makes uses the type calculus to compute from Peter's classification of the initially given data and the security policies of the involved web services the resulting classifications of all data which will be computed within the composed web service plan. Based on the resulting types it can check the admissibility of the incorporated web service calls.

Figure 5 and 6 illustrate the resulting security type check of a composed plan to buy the medicine from pharmacy agent MA1 first and then to instruct payment agent PA1 to pay the bill. While in Figure 5 the plan fails the security requirements because the price includes information about the medicine (and thus implicitly also about the disease), Figure 6 shows the case in which the same plan gets accepted because the price does not reveal the information about the medicine. Another solution would be to exchange the payment agent. Using PA_1 instead of PA_2 would allow the MED agent to transfer

information rated as *H* with respect to "Medicine" to the payment agent since PA_1 possesses a *H* clearance with respect to this category.

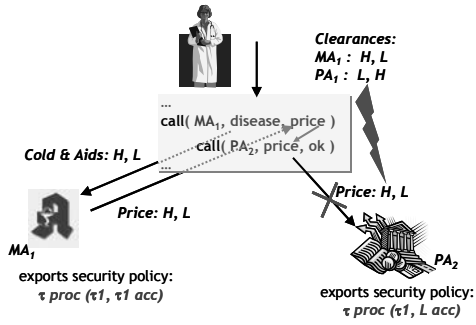

Figure 5: Security check (*H*-prices)

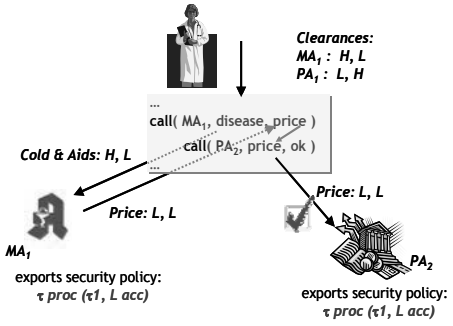

Figure 6: Security check ("flatrate")

6 Related Work

Starting with the work of Goguen and Meseguer [4] 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 not decidable. 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 the usage of 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. [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 absence 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].

7 Conclusions

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. 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 in part complementary 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*, 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, 2002.

[7] L. Kagal, M. Paoucci, N. Srinivasan, G. Denker, T. Finin, and K. Sycara. Authorization and Privacy for SemanticWeb 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 owlsxplan. 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 *IEEE Computer Security Foundations Workshop*, Cambridge, UK, IEEE Computer Society, 2000.

[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. Ant´on, 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*. 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.