IDEA GROUP PUBLISHING **ITJ3332**

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

This paper appears in the publication, **International Journal of Web Services Research**, Volume 3, Issue 3
edited by Liang-Jie Zhang © 2006, Idea Group Inc.

An Adaptive Access Control Model for Web Services

Elisa Bertino, Purdue University, USA
Anna C. Squicciarini, Purdue University, USA
Lorenzo Martino, Purdue University, USA
Federica Paci, University of Milano, Italy

ABSTRACT

This paper presents an innovative access control model, referred to as Web service Access Control Version 1 (Ws-AC1), specifically tailored to Web services. The most distinguishing features of this model are the flexible granularity in protection objects and negotiation capabilities. Under Ws-AC1, an authorization can be associated with a single service and can specify for which parameter values the service can be authorized for use, thus providing a fine access control granularity. Ws-AC1 also supports coarse granularities in protection objects in that it provides the notion of service class under which several services can be grouped. Authorizations can then be associated with a service class and automatically propagated to each element in the class. The negotiation capabilities of Ws-AC1 are related to the negotiation of identity attributes and the service parameters. Identity attributes refer to information that a party requesting a service may need to submit in order to obtain the service. The access control policy model of Ws-AC1 supports the specification of policies in which conditions are stated, specifying the identity attributes to be provided and constraints on their values. In addition, conditions may also be specified against context parameters, such as time. To enhance privacy and security, the actual submission of these identity attributes is executed through a negotiation process. Parameters may also be negotiated when a subject requires use of a service with certain parameters values that, however, are not authorized under the policies in place. In this paper, we provide the formal definitions underlying our model and the relevant algorithms, such as the access control algorithm. We also present an encoding of our model in the Web Services Description Language (WSDL) standard for which we develop an extension, required to support Ws-AC1.

Keywords: access control; authorization; security; trust negotiation; WSDL

INTRODUCTION

Web services are a key component of the emerging, loosely coupled, Web-based computing architectural paradigm. They represent the core element for building complex application services provided either by single companies or by a set of cooperating companies. The area of Web services today, thus, is an active area characterized by academic research, industrial developments as well as standardization efforts.

However, despite such intense research and development efforts, current Web service technology does not provide yet the flexibility needed to "tailor" a Web service according to preferences of the requesting subjects, thus failing to fulfil the mass-customization promises made at the beginning of the Web services era. At the same time, Web service providers demand enhanced adaptivity capabilities in order to adapt the provisioning of a Web service to dynamic changes of the Web service "environment" according to their own policies. Altogether, these two requirements call for policy-driven access controls model and mechanisms, extended with negotiation capabilities.

Models and languages to specify access and management control policies have been widely investigated in the area of distributed systems (Damianou, Dulay, Lupu, & Sloman, 2001). Standardization bodies have also proposed policy-driven, standard access-control models (Oasis XACML, 2004). The main goals of such models are to separate the access control mechanism from the applications and to make the access control mechanism itself easily configurable according to different, easily deployable access control policies.

The characteristics of the open Web environment, where interacting subjects are mostly unknown to each other, has led to the development of the *trust negotiation* approach as a suitable access control model for this environment (Yu, Winslett, & Seamons, 2003; Herzberg, Mihaeli, Mass, Naor, & Ravid, 2000; Bertino, Ferrari, Squicciarini, 2003). Trust negotiation itself has been extended with adaptive access control, in order to adapt the system to dynamically changing security conditions (Ryutov, Zhou, Neuman, Leithead, & Seamons, 2005). In such work, a framework is proposed that integrates trust negotiation techniques with a middleware (Ryutov & Neumman, 2002), providing access control and application-level intrusion detection and response. Automated negotiation is also being actively investigated in different application domains, such as e-business and Grid computing. However, a common key requirement that has been highlighted is the need of a flexible negotiation approach that enables the system to dynamically adapt to changing conditions. In addition, the integration of trust negotiation techniques with Semantic Web technologies, such as semantic annotations and rule-oriented access control policies, has been proposed (Gavriloaie, Nejdl, Olmedilla, Seamons, & Winslett, 2004). In this approach, the resource under the control of the access control policy is an item on the Semantic Web, with its salient properties represented as RDF properties. RDF metadata, managed as facts in logic programming, are associated with a resource and are used to determine which policies are applicable to the resource.

When extending a Web service with negotiation capabilities, the invocation of a Web service has to be managed as the last step of a conversation between the client and the Web service itself. The rules for such a conversation are defined by the negotiation protocol. Such a negotiation protocol should be described and made publicly available in a similar way as a Web service operation is publicly described through WSDL (W3C WSDL, 2005) declarations. An eXtensible Markup Language (XML)-based, machine-processable negotiation protocol description allows an electronic agent to automatically generate the messages needed to interact with the Web service. Of course, the client and the Web service must be equipped with a negotiation engine that evaluates the incoming messages, makes decisions and generates the outgoing messages according to the agreed-upon protocol.

The models already proposed for peer-to-peer negotiations assume that both parties are equipped with the same negotiation engine that implements the mutually understood negotiation protocol. This assumption, however, might not be realistic and may prevent the wide adoption of negotiation-enhanced access control models and mechanisms.

In this paper, we address the outlined requirements by proposing a Web service access control model and an associated negotiation protocol. The proposed model, Ws-AC1, is

based on a declarative and highly expressive access control policy language. Such a language allows one to specify authorizations containing conditions and constraints not only against the Web service parameters but also against the identity attributes of the party requesting the service and context parameters that can be bound, for example, to a monitor of the Web service operating environment. An additional distinguishing feature of Ws-AC1 is the range of object protection granularity it supports. Under Ws-AC1, the Web service security administrator can specify several access control policies for the same service, each one characterized by different constraints for the service parameters, or can specify a single policy that applies to all services in a set; to support such granularity, we introduce the notion of service class to group Web services. To the best of our knowledge Ws-AC1 is the first access control model developed specifically for Web services characterized by articulated negotiation capabilities. We believe that a model like Ws-AC1 has important applications, especially when dealing with privacy of identity information of users and with dynamic application environments.

To represent the negotiation protocol, we also propose an extension to the WSDL standard. The main reason of that choice is that, although the Web Services Choreography Description Language (WS-CDL) is the emerging standard for representing Web services interactions, WS-CDL is oriented to support a more complex composition of Web services in the context of a business process involving multiple parties.

The paper is organized as follows: We first present an overview of our approach to access control for Web services, and the formal model of Ws-AC1. We then describe how the Ws-AC1 service description and the Ws-AC1 policies can be represented in WSDL and WS-Policy (IBM, BEA Systems, Microsoft, SAP AG, Sonic Software & VeriSign, 2004), respectively. Finally, we discuss related works and present the conclusion and directions for future work.

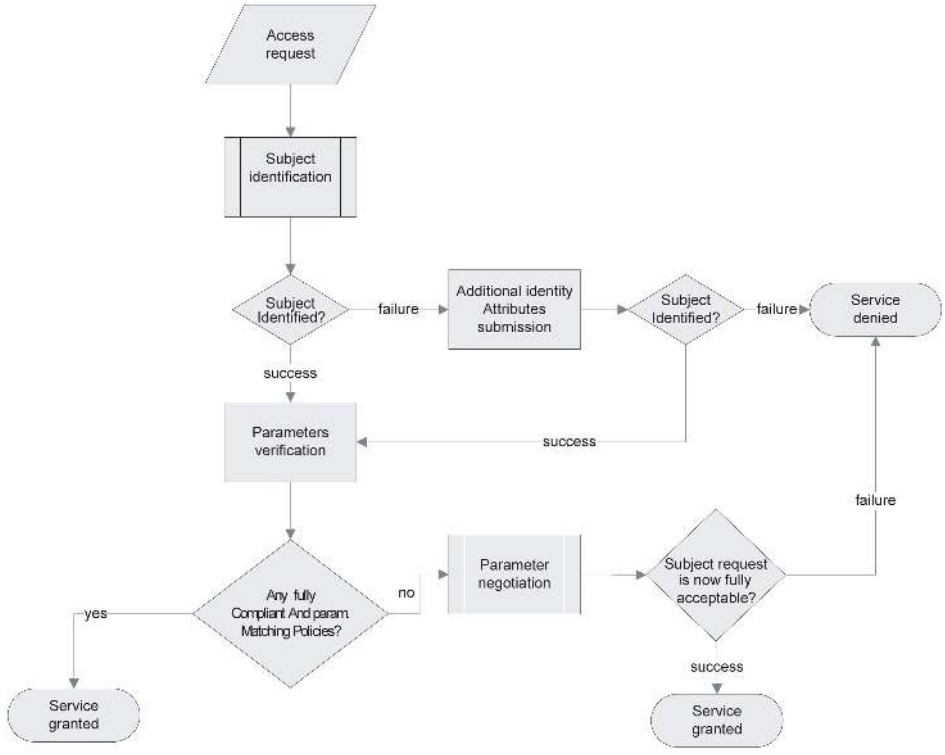
SYSTEM OVERVIEW

Ws-AC1 is an implementation-independent, attribute-based access control model for Web services, providing mechanisms for negotiation of service parameters. In Ws-AC1, the requesting agents (also referred to as *subjects*) are entities (human being or software agents), the requests by which have to be evaluated and to which authorizations (permissions or denials) can be granted. Subjects are identified by means of identity attributes qualifying them, such as name, birth date, credit card number and passport number. Identity attributes are disclosed within access requests invoking the desired service.

Access requests to a Web service (also referred to as *provider agent*) are evaluated with respect to access control policies. Note that for the sake of simplicity, our model does not distinguish between the Web service and the different operations it provides; that is, we assume that a Web service provides a single operation. Our proposed access model can be applied to the various operations provided by a Web service without any extension.

Access control policies are defined in terms of the identity attributes of the requesting agent and the set of allowed service parameters values. Both identity attributes and service parameters are further differentiated in mandatory and optional ones. For privacy and security purposes, access control policies are not published along with the service description, but are internal to the Ws-AC1 system. Ws-AC1 also allows one to specify multiple policies at different levels of granularity. It is possible to associate fine-grained policies with a specific service as well with several services. To this end, it is possible to group different services in one or more classes and specify policies referring to a specific service class, thus reducing the number of policies that need to be specified by a policy administrator. A policy for a class of services is then applied to all the services of that class, unless policies associated with the specific service(s) are defined. We present in the following sections the conditions under which services can be grouped into classes and the

Figure 1. Conceptual organization of access control in Ws-AC1



criteria used by Ws-AC1 to select the policies to use upon a service request.

Moreover, to adapt the provision of the service to dynamically changing conditions, the Ws-AC1 policy language allows one to specify constraints, dynamically evaluated, over a set of environment variables, referred to as *Context*, as well as over service parameters. The *context* is associated with a specific service implementation, and it might consist of monitored system variables, such as the system load.

As illustrated in Figure 1, the access control process of Ws-AC1 is organized into two main sequential phases. The first phase deals with identification of the subject requesting the service. The second phase, executed only if identification succeeds, verifies the service parameters specified by the requesting agent against the authorized service parameters.

The identification phase is adaptive, in that the provider agent might eventually require the requesting agent to submit additional identity attributes in addition to those originally submitted. Such an approach allows the provider agent to adapt the service provisioning to dynamic situations: for example, after a security attack, the provider agent might require additional identity attributes to the requesting agents. In addition, to enhance the flexibility of access control, the service parameter verification phase can trigger a negotiation process. The purpose of this process is to drive the requesting agent toward the specification of an access request compliant to the service specification and policies. The negotiation consists in an exchange of messages between the two negotiating entities to limit, fix or propose the authorized parameters the requesting agent may use. The

negotiation of service parameters allows the provider agent to tailor the service provisioning to the requesting agent preferences or, at the opposite, to "enforce" its own preferred service provisioning conditions.

THE WS-AC1 MODEL: FORMAL DEFINITIONS

In this section we formally specify the main notions underlying the Ws-AC1 access control model. We refer to an example of a Web service, called *DrugStore*, supplying medicines and drugs to general customers and to private clinics and hospitals. We start by presenting the notion of *service description*, which specifies the information necessary to invoke the service. We then introduce, respectively, the notion of *Web service context*, *access request* and *access control policy*.

Definition 1 (Service Description)

A service description is a tuple of the form *Serv-descr* = <*s*; *Parameters*; *AuthAttrs*> where:

- s is a service identifier;
- Parameters = [$Pspec_1$,..,$Pspec_n$] where $Pspec_i$, i=1,…,n, is a triple of the form (P_i, $Domain_{Pi}$, $ParamType_{Pi}$) such that:
  - P_i is a parameter name;
  - $Domain_{Pi}$ denotes the set of values the parameter can assume;
  - $ParamType_{Pi} \in$ {mand, opt} specifies whether the parameter is mandatory or optional;
- AuthAttrs = [(A_1, $AttrType_{Ai}$), (A_2, $AttrType_{A2}$)….,(A_k, $AttrType_{Ak}$)] where (A_i, $AttrType_{Ai}$), i=1,…,k, represents an identity attribute; A_i is the name of the attribute, and $AttrType_{Ai}$ indicates if the attribute is mandatory or optional.

Given a service description of a service *s*, in the following we represent with *MandAtt* the set of mandatory attributes in *AuthAttrs* and *MandPar* the set of mandatory parameters in *Parameters.* Further, we refer to as *PN* the set of parameter names in *Serv-descr*.

A service description serves the following main purposes:

1. It allows the potential clients of the service to obtain the description of both the identity attributes (*AuthAttrs*) and the service parameters (*Parameters*) needed to submit a request to the service. Identity attributes are properties, such as name, birth date, credit card and passport, qualifying a requesting agent. Service parameters represent information the requesting agent has to provide to activate the operation supported by the service, and also information related to level of quality of service required by the requesting agent.
2. It conveys (and specifies) to potential clients of the service the following information:
   - which identity attributes are mandatory and optional;
   - which service parameters are mandatory and optional.

While mandatory identity attributes and service parameters *must* be assigned a value by the requesting agent as part of the initial request for the service, the optional ones do not have such a requirement. However, depending on their values, submission of the mandatory attributes by the requesting agent may not be enough for gaining access to the service. As such, values for the optional identity attributes may be required by the service agent during the subsequent negotiation process. We further elaborate on requesting agent authentication in the following section.

Accesses in Ws-AC1 are either granted or denied on the basis of access conditions referring to the identity attributes of the requesting agent and in terms of the parameters characterizing the service.

Example 1. *Serv-descr = <DrugStore; ((MedicineName, string, opt) (MedicineActivePrinciple,string,mand), (Price, {Lowest, Medium, High}, mand), (Quantity,{} ,mand));*

(CustomerId, mand)> is the DrugStore service description where:

- DrugStore is the service identifier
- MedicineName, MedicineActivePrinciple, Price and Quantity are the service parameters necessary to invoke the DrugStore service. MedicineName is an optional parameter and indicates the name of the medicine the customer wants to order. MedicineActivePrinciple specifies the active principle of the medicine the customer wants to purchase, Price represents customer preference about the medicine price and Quantity is the number of the medicine items required by the customer.
- CustomerId is the attribute used by the Ws-AC1 system to identify the service requester. CustomerId can be the name of a final user or the name of a Hospital or a private Clinic.

The Ws-AC1 system associates with the service a *context,* composed of a set of variables that can influence service provisioning. The formal definition is given in what follows.

Definition 2 (Web Service Context)

Let *s* be a service identifier. Let *SV* be set of names of the Ws-AC1 system variables. The context of *s* is a set $serv\_context = \{CP_1: cp_1, CP_2: cp_2, ..., CP_m: cp_m\}$, where CP_i is a variable name in *SV* and cp_i is the value assigned to the corresponding variable, i=1,…,m.

The context is evaluated by the Ws-AC1 system to enforce access control to the service as explained later in this section. The Ws-AC1 system updates the context variables each time an access request is received or the context changes. In what follows, the set of context variable names for a service *s* is abbreviated with *CVN*.

Example 2. *An example of context that can be associated with the DrugStore service is* $serv\_context_{DrugStore} = [UsersConnected:1000]$, *where UsersConnected records the number of users connected to the service during a given time period*.

The invocation of a service is formalized as an *access request* in which the requesting agent has to provide the information specified in the service description; that is, its qualifying attributes, the parameters of the Web service and the Web server identifier.

Definition 3 (Access Request)

An access request is a tuple *acc = (a, s, p)* where:

- $a = [A_1: a_1, A_2: a_2, ..., A_m: a_m]$ where A_i is an identity attribute of the requesting agent and a_i is the associated value, i=1,…,m
- *s* is a service identifier;
- $p = [P_1: p_1, P_2: p_2, ..., P_k: p_k]$ where P_i is a parameter characterizing the service and p_i is the associated value, i=1,…,k..

Example 3. *Referring to the service description introduced in Example 1, the access request must contain the identity attribute CustomerId and the service parameters MedicineActivePrinciple, Price and Quantity. An example of such an access request is the following:*

acc = ([CustomerId: Chicago Hospital]; DrugStore; [MedicineActivePrinciples: salicylic acid, Price:Medium; Quantity:20000]).

The Ws-AC1 system evaluates access requests with respect to the access control policies protecting the required service. The same service may be protected by several access control policies. Informally, an access control policy is expressed by means of three components: a component to authenticate the requester, a component for specifying the parameters to which the policy applies to and a component for specifying the parameter values allowed by the service. To authenticate requesters, policies may convey attribute conditions specifying

the conditions that each identity attribute of the requesting agent has to satisfy in order to access the service.

To enhance flexibility, the model allows one to specify for each service the set of *legal parameter values* that the service parameters can be assigned. Legal parameter values are defined by ad-hoc rules, referred to as *constraints*, defined over the set of the service parameters and/or the set of the service context variables. Constraints are evaluated dynamically. It is thus possible to adapt the access control policies to dynamically varying conditions. The formal definition of the constraints is given in what follows. The definition refers to the following sets: *PN*, the set of the parameters names; *CVN,* the set of context variable names.

Definition 4 (Constraint)

Let *serv-descr* be a service description of a service *s*, and let *serv_context* = (CP_1, CP_2, ..., CP_m) be the associated context. A *constraint* is represented by a logic rule of the form:

$$H \leftarrow L_1, L_2, \ldots\ldots\ldots, L_n, not\ F_1, not\ F_2, \ldots\ldots\ldots not\ F_m$$

where:

- *H* is the *head* of the rule and is an expression of the form *ArgName Op Values* where *ArgName* is an element in *PN*, *Op* is a comparison operator[1] or the $\in$ operator, and *Values* can be a set of values defined through enumeration or a range expression $[v_{begin}, v_{end}]$, or can be a single value;
- $L_1, L_2, \ldots L_n, not\ F_1, not\ F_2, \ldots., not\ F_m$ is the *body* of the rule; each L_i, i=1,…,n, or F_k k=1,…,m, in the body of the rule can be an expression of the form *ArgName op Values,* where *ArgName* is either an element in *PN* or in *CVN*. The body of a rule is empty to denote always true rules.

A constraint restricts the set of values associated with a parameter on the basis of the current values of the context variables and/or of the values assumed by other services' parameters. In the following, given a constraint $Constr_k$, we denote with $Legal\_Values_{Constrk}(P_i)$ the *Values* set of values assigned to the parameter P_i in the head of $Constr_k$ and with $Target_{Constrk}$ the service parameter name P_i in *H*.

Example 4. *With respect to the Web service presented in Example 1, following constraints can be specified:*

- *Quantity=10←MedicineActivePrinciple =salicylAcid; Price=Low*
- **Pr***ice=High*←StockLevel<**100**,MedicineActive**Pr**inciple=sildenafil citrato

The first constraint states that if the requesting agent wants to purchase a medicine, the active principle, which is salicyl acid, paying the lowest price, it can order only 10 items.

The second constraint specifies that when the stock level of the requested medicine is less than 100, and the user wants to order a medicine, the active principle of which is sildenafil citrate, it can only place the order paying a high price per medicine.

As already mentioned, access control policies in Ws-AC1 can be specified at different granularity levels. Precisely, a policy can govern access to either a single service (corresponding, in our model, to a Web service description) or a class of services. Services can be clustered in *classes* and be referred as a whole in a policy. In the following, we represent a class of services as a set of service identifiers WSClass=$\{s_1,..,s_k\}$, where s_i, i=1,…, k, denotes a service identifier.

Example 5. *The DrugStore service introduced in Example 1 is an element of the WSClass BuyOnline. BuyOnline is composed of three Web services: DrugStore, FoodStore and OnlineStore. FoodStore is a Web service allowing one to buy any kind of food online. OnlineStore is a Web service that allows one to buy different*

kinds of products belonging to different categories, like Books, Music, Electronics.

DrugStore, FoodStore and OnlineStore are characterized by the same mandatory identity attribute CustomerId and by the mandatory parameters Price and Quantity.

In the following, we use the dot notation to refer to a component of a tuple; that is, we use *R.a* to denote component *a* of tuple *R*.

For the sake of simplicity, we provide first the notion of access control policy for a single service and then, in Definition 6, we formalize the notion of access control policies for a class of services.

Definition 5 (Service Access Control Policy)

Let *s* be a service, and let *serv-descr* = <*s*; *Parameters*; *AuthAttrs*> be its description. An access control policy specified for *s* is defined as a tuple *pol* = <*st; C; ParamConstr; ParamSet*>, where:

- *st* denotes the identifier of *s*;
- *C* is a list of the form $\{CA_1, CA_2, ..., CA_n\}$, $n \geq 1$, where CA_i, i=1,…,n, is either an attribute condition or an attribute name;
- *ParamConstr* = $\{Constr_1, Constr_2, ..., Constr_k\}$, $k \geq 1$, is a (possibly) empty set of constraints defined over parameters in *ParamSet* such that for each $Constr_i$, $Constr_j$, $i \neq j$, $Target_{constr\ i} \neq Target_{constr}$;
- *ParamSet* is a set of parameter names referring to the description of *st*, such that *ParamSet*$\subseteq$ *Parameters.*

The above definition shows that the proposed access control allows one to specify fine-grained access control policies, in that one can associate a policy with a single service and even specify with which input parameters the service has to be invoked under a given policy. However, to simplify access control administration as much as possible, it is also important to support access control policies with coarse granularities. Such a requirement is addressed in our model by associating access control policies with classes of services. In other words, a single policy can be specified for all services belonging to a given class of services. However, to be regulated by a single policy, a service class has to include Web services satisfying the condition that the set of mandatory parameters and the set of mandatory attributes for all the services in the class be the same.

Definition 6 (Class Access Control Policy)

Let WSClass=$\{s_1,..,s_k\}$ be a class of services. Let s_i, i=1,…k, be a service identifier in WSClass and *serv-descr*$_i$ = <s_i; *Parameters*; *AuthAttrs*> be the corresponding service description. An access control policy specified for WSClass is defined as a tuple *pol* = <*class; C; ParamConstr; ParamSet* > where:

- *class* is the class identifier;
- *C* is a list of the form $\{CA_{1, ...}\ CA_n\}$ and each CA_i, i=1,…,n, is either an attribute condition or an attribute name. Each attribute name is a mandatory attribute for every service $s_i \in$ WSClass;
- *Paramset* is a set of parameter names. For each s_i in WSClass and for each *p* in *Paramset*, $p \in s_i.Parameters$ and $s_i.p.ParamType$=mand;
- ParamConstr = $\{Constr_1, Constr_2, ..., Constr_k\}$ is a (possibly) empty set of constraints defined over parameters in ParamSet such that for each $Constr_i$, $Constr_j$, $i \neq j$ $Target_{constr\ i} \neq Target_{constr}$.

The semantics of policies specified at class level is that they apply to any service in the class.

The advantage in supporting class policies for the service providers managing a large number of services is obvious. Service providers have the possibility of clustering as many services as they wish and specifying a unique policy while being able to refine policies for particular services, if required.

Example 6. *With reference to the DrugStore Web service introduced in Example 1, consider the following access control policies:*

pol_1 = < *DrugStore; {CustomerId* ∈*{Ann Meeker, John Smith}*, *PatientCardId* ∈*{AS128456, AX3455643}* }; { *MedicineActivePrinciple, PriceFare, Quantity* }; *Quantity=10*←*MedicineActive*Pr*inciple =salicyl acid,*Pr*iceFare=low*>

pol_2=<*DrugStore*; {*CustomerId = John Smith*, *PatientCardId = AS12345*}; {*MedicineActivePrinciple, Price , Quantity* }; { }>

pol_3= < *DrugStore; {CustomerId = ChicagoHospital , DoctorPrescriptionId = 34567*}; { *MedicineActivePrinciple, Price*};

Pr*ice=High*←*StockLevel<10,MedicineActive* Pr*inciple=sildenafil citaato*

Policy pol_1 states that users Ann Meeker and John Smith having a PatientCardId equal either to AS128456 or to AX3455643 can invoke the service. Specifically, the policy constraints limit the quantity that can be ordered to 10 items.

Policy pol_2 states that the user John Smith having a PatientCardId equal to AS123451 can access the service. The policy does not impose any restriction on the values the service parameters can assume.

Finally, policy pol_3 requires that the Chicago hospital providing a doctor prescription for a drug containing sildenafil citrato can only get the drug at a high price if the doctor submitted the request when the stock level of the medicine is less than 10 .

WS-AC1 IDENTITY ATTRIBUTE NEGOTIATION

Ws-AC1 evaluates access requests with respect to the access control policies protecting the corresponding services or, respectively, the service classes, if no ad hoc policies are specified for the required services. Each access request is first evaluated with respect to the identifying attributes submitted. If the attribute values specified by the requesting agent in the access request do not satisfy all the conditions of any corresponding access control policy, the access request is said to be partially compliant. The system can then require the requesting agent to provide the additional attributes specified by the policy but not appearing in the service description.

In the following subsection, we present the formal definitions of *total and partial compliance* of an access request with respect to an access control policy and the notion of *parameter matching* of an access request with respect to an access control policy. Then, we describe the main steps of the identification process of the user requesting the service, based on the negotiation of attributes.

Formal Definitions

An access request in Ws-AC1 can be either totally or partially compliant with a (single or class) access control policy.

Definition 7 (Total Compliance of Identity Attributes)

Let *acc* = (*a, s, p*) be an access request, and let *pol* = <*st; C; ParamSet; ParamConstr*> be an access control policy.

Acc **totally complies** (denoted as ***TC***) with *pol* if both the following conditions hold:

1. if st= s then acc.s = pol.s else acc.s ∈ pol.Class;
2. *for each attribute condition C of the form* $C = A\ op\ k$, $C \in pol.C$, $\exists j : A = acc.\underline{a}.A_j \wedge C$ *is true* according to value $acc.\underline{a}.a_j$ assigned to A_j.

In case no access control policy for the required service is specified, a class policy referring the class the required service belongs to has to be evaluated. In such a case, condi-

tion 1 is formulated in terms of a Web service class policy.

By definition, an access request is totally compliant with an access control policy if all the attribute conditions specified in the policy are evaluated true, according to the attribute values specified in the access request.

If no access control policy exists for which the access request is totally compliant, the request is rejected. However, Ws-AC1 gives the requester the possibility of providing additional attributes to fully comply with an access control policy. To this end, we introduce the concept of partial compliance of an access request.

Definition 8 (Partial Compliance of Identity Attributes)

Let *acc* = (*a, s, p*) be an access request, and let *pol*=<*st; C; ParamSet; ParamConstr*> be an access control policy. *Acc* **partially complies** (denoted as ***PC***) with *pol* if both the following conditions hold:

1. If *st*= *s* then *acc*.s = *pol*.s else *acc.s* ∈ *pol.Class*;
2. An attribute A_j in acc.a and an element AC in *pol.C* such that either *C*= *A op k* and is evaluated true according to the value of *Aj* or C is a attribute name equal to A_j.

As stated by the definition, an access request for a specified service is *partially compliant* with an access control policy if a subset of the attribute names of the policy appears in the access request, or if some attribute of the access request appears in some condition of the policy and the condition evaluates to true according to the attribute value submitted in the access request. In the case of partial compliance of the attributes, Ws-AC1 asks the requesting agent to disclose the attributes not provided in the submitted request, but specified in the access control policies the access request partially complies with. Moreover, an access request may be totally compliant with respect to a policy, but it may specify parameter values not allowed by the service. In this case also, the access request cannot be accepted as-is. This leads us to introduce another form of partial compliance with respect to a policy. Such a notion is based on the definition of *parameter matching* for an access request given next.

Definition 9 (Parameter Matching)

Let *acc* = (*a, s, p*) be an access request, and let *pol* = <*st*; *C*; *ParamSet*; *ParamConstr*> be an access control policy. *Acc is* **parameter matching** (denoted als as ***PM***) with respect to *pol* if both the following conditions hold:

1. If st= s then acc.s = pol.s else acc.s ∈ pol. Class
2. each parameter acc.p.P in acc ∈ Legal_Values $_{Constrk}$ (P) where Constr is a constraint in pol. ParamConstr or if does not exist a constraint Constr ∈ pol. ParamConstr such that Target$_{Constr}$ = $P_{i,}$ acc.p.P must belong to Domain$_P$

As shown in the definition, an access request is parameter matching if each parameter value requested is acceptable; that is, it either satisfies a policy constraint (if applicable) or falls in the corresponding parameter domain. In the end, access to a Web service can only be granted if an access request *fully satisfies* an access control policy. This requires both the successful identification of the requesting agent and the agreement on the parameter values for invoking the service. The next section details the requesting agent identification process through attribute negotiation.

THE NEGOTIATION PROCESS FOR IDENTITY ATTRIBUTES

As mentioned in the previous discussion, upon receiving an access request, the system determines whether any access control policy exists for the required service or for the class the service belongs to for which the access request is totally compliant for the identity attributes. If such a policy is found, the pending request is further evaluated for parameter matching, to check if access can be granted. If no policy for which the access request is totally compliant

is found, instead of rejecting the request, the Ws-AC1 system checks if the access request is partially compliant with any of the enforced policies. If this is the case, the system asks the requesting agent to provide additional attributes. In particular, the requesting agent has to submit the attributes not provided in the request, specified in the access control policies the access request partially complies with. Such attributes are requested by server to the requesting agent with an ad hoc message, referred to as *request for attributes* or *rfa*.

Definition 10 (Request for Attributes)

Let *acc*= (a, s, p) be an access request and *PartialCompliantPolSe*t = (pol_1 ,… pol_k) be the set of access control policies *acc* partially complies with. An *rfa* is a disjunction of attributes sets $rfa = AttrSet_1 \vee AttrSet_2 \vee \ldots\ldots \vee AttrSet_k$, where each $AttrSet_i = (A_1, A_2, \ldots, A_m)$, i=1,…,k, is the set of attributes names specified in pol_i.C not contained in *acc*.a.

If more than one access control policy is found, Ws-AC1 selects among these policies the ones the access request parameter is matching with. If the result of the selection is not empty, then the *rfa* will contain an attributes set for each selected access control policy. Otherwise, *rfa* will contain an attribute set for each access control policy the access request partially complies with. The requesting agent, thus, has the freedom to decide which set of attributes to reveal. The message used by the requesting agent to reply to an *rfa* sent by the server is referred to as *response for attribute*s or *rsfa.*

Definition 11 (Response for Attributes)

Let $rfa = AttrSet_1 \vee AttrSet_2 \vee \ldots\ldots \vee AttrSet_k$ be a request for attributes. A *response for attributes* is a tuple $rsfa = [Attr_1:a_1, Attr_2: a_2, \ldots, Attr_m:a_m]$ where each $Attr_i \in AttrSet_i$, *i*=1,…,n, is one of the attributes sets specified in the corresponding *rfa*, and a_i is the associated value.

After receiving the *rsfa* message, the Ws-AC1 system verifies if the access request updated with the attributes just submitted is now totally compliant with one of the access control policies the original access request was partially compliant with. In case the access request now provides all attributes required by one of the access control policies, the system evaluates if the requesting agent can access the service on the basis of the parameters' values specified. It is important to note that the identification process is not iterative: It lasts two rounds in the worst case — one round for sending attributes request and another one for the reply. In case no fully compliant policy can be found, the request is rejected without possibility of further negotiation.

Algorithm 1 describes the negotiation process for identity attributes submitted by a requesting agent. In Figure 2, the main steps of the algorithm are described. The algorithm accepts in input two different types of messages: an access request *acc* or a response for attributes *rsfa*.

If the input message is an access request, the algorithm builds *TotalCompliantPolSet;* that is, the set of access control policies the request *acc* totally complies with (**step 7**). If the set is empty, the algorithm builds a so-called *PartialCompliantPolSet;* that is, the set of access control policies the request *acc* partially complies with (**step 10**). If there are no policies, the access to the service is denied to the user (**steps 11-12**). Otherwise, the request for attribute message *rfa* is created by invoking function *Generate-RFA()* (**steps 14-20**). To generate the request for attributes message, the algorithm can adopt two strategies. First, it checks if *acc* is parameter matching with some of the policies in the set *PartialCompliantPolSet.* In this case, the algorithm builds the set *SelectedPol,* a subset of *PartialCompliantPolSet* containing the policies with which acc is parameter matching (**steps 14-16**). If *SelectedPol* is not an empty set, the function *Generate-RFA()* is activated and generates a request for an attribute message containing an attributes set for each policy in *SelectedPol*: the set of attributes specified in *rfa* contains the attributes specified in the policy, not provided by the user in *acc* (**steps 17-18**).

Algorithm 1. Generation of a request for attributes

INPUT:
Message: an access request *acc*= (a, s, p) or a *rfsa* = [$Attr_1$: a_1, $Attr_2$: a_2,...., $Attr_m$: a_m]

OUTPUT:
"Access denied" ***or***
A request for attributes message *rfa* = $AttrSet_1$ v $AttrSet_2$ v ..... v $AttrSet_k$ ***or***
TotalCompliantPolSet , the set of access control policies, the request *acc* totally complies with.

```
1)      If message is an access request acc
2)        msg-type:= false
3)      Else
4)        If message is a rsfa
5)          acc = Update-acc(acc, rsfa)
6)          msg-type:= true
7)      endif
8)      Let TotalCompliantPolSet be the set of access control policies, the request acc totally complies with.
9)      If TotalCompliantPolSet = ø
10)       If  msg-type:= false
11)                 Let PartialCompliantPolSet be the set of access control policies, the request acc partially complies
                    with.
12)         If PartialCompliantPolSet = ø
13)           return "Access denied"
14)         Else
15)           Foreach pol_i ∈ PartialCompliantPolSet
16)             If acc is parameter matching with pol_i
17)               SelectedPol = SelectedPol ∪ pol_i
18)           If SelectedPol <> ø
19)             rfa = Generate-RFA( SelectedPol, acc)
20)           Else
21)             rfa =  Generate-RFA(PartialCompliantPolSet, acc)
22)           return rfa
23)       Else
24)         return "Access denied"
25)     Else
26)       return TotalCompliantPolSet
```

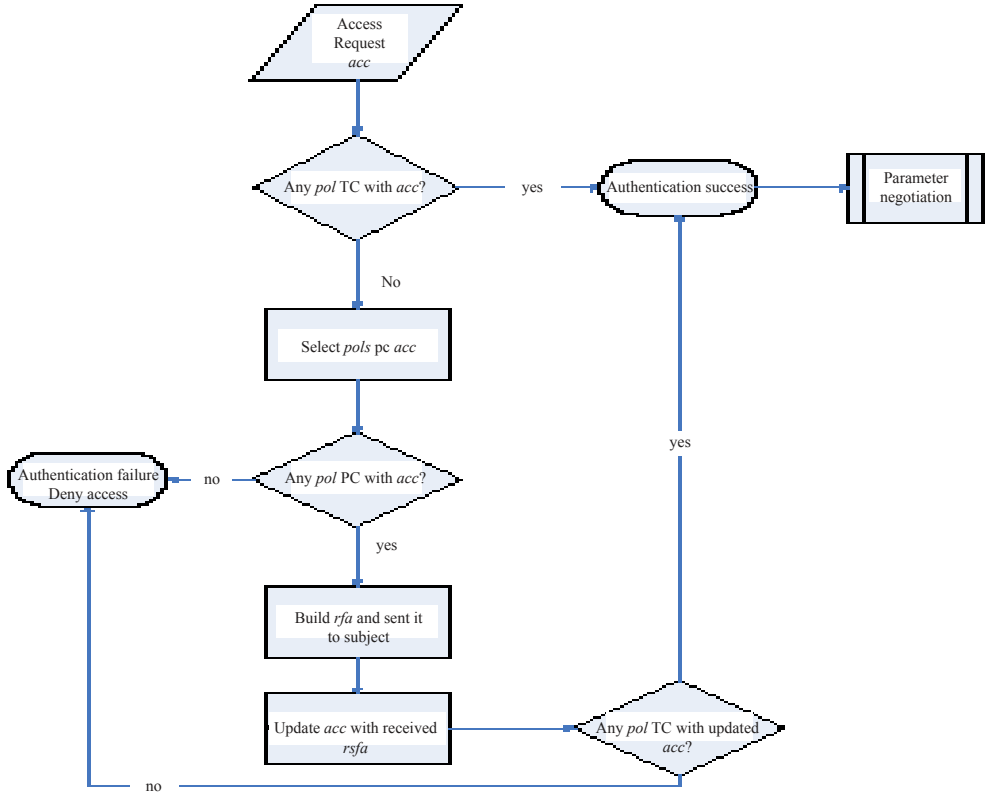
Instead, if *SelectedPol* is empty, *Generate-RFA()* is activated and generates a request for attribute message containing an attributes set for each policy in *PartialCompliantPolSet* (**step 20**).

If set *TotalCompliantPolSet* is not empty, the algorithm returns this set, as it represents the input for the service parameter negotiation process (**step 25**).

In case the input message is a response for attributes, the algorithm first updates the access request *acc* previously received, invoking *Update-acc()*.*Update-acc()* adds to *acc* the attributes the service requester has specified in the request for attributes. Then, it builds the set of access control policies the updated *acc* totally complies with (**step 7**), referred to as *TotalCompliantPolSet*. If such set is empty, the algorithm ends by denying the service access (**step 23**).

Example 7. *Consider the access request "acc" introduced in Example 3 and the access control policies specified in Example 5. Acc does not totally comply with* pol_1, pol_2, pol_3, *but it partially complies with* pol_3. *In fact, the attribute condition CustomerId = ChicagoHospital is evaluated to true according the CustomerId attribute value specified in acc. Hence, the*

Figure 2. The negotiation of identity attributes in Ws-AC1



Ws-AC1 system asks the service requester for attribute rfa = {DoctorPrescriptionId} where DoctorPrescriptionId is the attribute name specified in pol$_3$, not provided in acc.

Parameter Negotiation

In the following sections, we describe the other relevant negotiation process of Ws-AC1; that is, the parameter negotiation. First, we provide formal definitions of request acceptance. Then, we illustrate the conditions triggering a negotiation process and formalize the type of messages and the protocol to follow.

Access Request Acceptance

Given a set of policies totally compliant with the requesting agent request, first the Ws-AC1 system checks whether an access control policy exists that makes the access request fully acceptable.

Definition 12 (Full acceptance)

Let *acc* = (*a, s, p*) be an access request, and let *pol* = (*s; C; ParamSet; ParamConstr*) an access control policy. *Acc is* **fully acceptable** (denoted also as ***FA***) by *pol* if both the following conditions hold:

1. *acc* is totally compliant with respect to *pol,* according to Definition 7.
2. *acc* is parameter matching with respect to *pol* according to Definition 9.

In case the access request is *fully acceptable*, the server grants the requesting agent access to the service.

Algorithm 2. Function Update_acc()

INPUT:
An access request *acc*= (a, s, p) and a *rsfa* = [$Attr_1$: a_1, $Attr_2$: a_2,…., $Attr_m$: a_m]

OUTPUT:
Update access request *acc'* = (a', s, p)

1) *acc'* = *acc*
2) **for** i = 0 to | *rsfa* |
3) *acc'*. a = *acc'*. a ∪ *rsfa*.$Attr_i$ ∪ ": " ∪ *rsfa*.$Attr_i$.a_i
4) **return** *acc'*

Algorithm 3. Function Generate-RFA()

INPUT:
A set of access control policies *PolSet*
An access request *acc*= (a, s, p)

OUTPUT:
A request for attributes *rfa* = $AttrSet_1$ v $AttrSet_2$ v ..... v $AttrSet_k$

1) *rfa* = ø
2) *PolAttrSet* = ø
3) **Foreach** *acc*.a.A_i
4) *AttrSet* = *AttrSet* ∪ A_i
5) *rfa* = *rfa* ∪ " < "
6) **Foreach** pol_i ∈ *PolSet*
7) **Foreach** pol_i.C.C_k = A_k *op k*
8) *PolAttrSet* = *PolAttrSet* ∪ A_k
9) *PolAttrSet* = *PolAttrSet* – *AttrSet*
10) **Foreach** A_j ∈ *PolAttrSet*
11) *rfa* = *rfa* ∪ A_j ∪ " , "
12) *rfa* = *rfa* ∪ " v "
13) *PolAttrSet* = ø
14) *rfa* = *rfa* ∪ " > "
15) **return** *rfa*

An access request is not fully acceptable and can be negotiated if it is totally compliant with a policy but is not parameter matching. Precisely, one of the following conditions occurs:

1. The access request is specified using all the parameters appearing in one or more access control policies, and contains parameter values that are not legal under these policies.
2. The access request is specified using a subset of the parameters provided by the policies enforced for the required service. Therefore, the requesting agent has to provide the missing parameters.

The requesting agent, thus, is given the possibility of negotiating with the incorrect parameters, as illustrated in the following section.

Example 8. *With respect to our running example, consider the following policies:*

pol_1 = <*DrugStore*; {*CustomerId* = *John Smith*, *DoctorId*}; {*Quantity*}; { *Quantity* ∈ *[1, 5000]*←}>

pol_2 = <*DrugStore*; {*CustomerId* = *John Smith*}; {*Quantity* }; {*Quantity* ∈ *[1, 1000]*←}>

acc = <CustomerId: John Smith; DrugStore; MedicineActivePrinciples: salicylic acid, Price:High; Quantity:2500 >

The access request above partially complies with policy pol1 and fully complies with policy pol2. The requesting agent can then opt for negotiating parameters and purchase the drugs in the quantity allowed, or it can also disclose its DoctorId and obtain authorization to buy up to 500000 items.

The Negotiation Process for Parameters

The process of parameter negotiation consists of message exchanges between the two parties. The provider agent starts the negotiation by sending to the requesting agent a message proposal, containing a combination of admitted values for the parameters of the required service. We call such a message *negotiation access proposal* (NAP). Given a totally compliant access request *acc* and an access control policy, a NAP is a tuple of the form *nap* = < *NegId; ap, end*> where:

- *NegId* is the negotiation identifier denoting the current negotiation.
- ap = {$P_{1:} p_1$,..., $P_{n:} p_n$} is a list of pairs where P_i is a parameter name belonging to *ParamSet* and p_i is the corresponding value, i=1,...,n.
- *end* ∈ *{yes, no}* is a flag denoting whether or not the NAP is the last one in the negotiation process.

The parameters included in a NAP depend on the misplaced values in the submitted access request. If the access request is specified using non-admitted parameter values (case a) of the previous section, the generated NAP will suggest legal values for the incorrect parameter values. The current version of Ws-AC1 does not provide any inference engine for checking conflicts among the enforced access control policies. Therefore, policies may overlap or may subsume one another. Hence, the same access request may be negotiated against several policies. If this case holds, the requesting agent will receive as many NAPs as the policies having all the parameter names *p* appearing both in *ParamSet* and in the *p* component of the access request. Of course, although this approach maximizes the success chances of the negotiation process, it has the drawback that in case of a large number of fully compliant policies, the requesting agent will be flooded by alternative NAPs. We will explore alternative approaches to better deal with this issue in our future work.

If the required service parameters are not specified at all in the access request (case b), the policies to be considered are the access control policies having at least one parameter name in common with the received access request. Here, the NAP will be composed by the parameter values chosen by the requesting agent whenever possible, and parameter values set by the system for the remaining parameters. As in the previous case, only policies having a parameter *p* appearing both in *ParamSet* and in the *p* component of the access requests are selected. Note that the criteria adopted for defining parameter values in a NAP are based on a "user-oriented" criteria. This means that given a non-fully acceptable access request, the fewest number of modifications on the original access request are applied. In other terms, all the acceptable parameter values are kept, while the non-acceptable ones are replaced with legal values. The replacement might be executed according to different approaches. A straightforward solution consists of specifying constant default parameter values to be used for filling

Algorithm 4. Negotiation algorithm

INPUT:
Message: an access request *acc*= (a, s, p) or a *nap* = [P_1: p_1, P_2 : p_2,…., P_m : p_m]
Policies of the form

OUTPUT :
"Access denied"
"Access Granted",
NAPList={nap$_1$*,…,nap*$_n$*}*, negotiation access proposal lists, each of the form < NegId ; *ap*,, end>

1) **If** message is an access request *acc*
2) **If** ¬ ∃ a policy pol s. t. acc.s = pol.s
3) Let tWSclass be the class service s belongs to
4) PolSet= {pol$_1$, …pol$_k$} be the policies for WSclass
5) **else**
6) PolSet={pol$_1$, …pol$_k$} be the set of policies such that acc.s= pol.s
7) Let accPNames be the set of parameter names in acc. $\overline{p}$
8) **Foreach** pol$_i$ in PolSet s.t. Pol$_i$.ParName = AccReqPNames
9) Case1= Case1 ∪ Pol$_i$
10) i=1;
11) **Repeat**
12) **If** acc.p.p$_i$ in Pol$_i$ParName s.t. acc.p.p$_i$ ∉ Legal_Values($\underline{p_z}$)
13) Case1= Case1 - Pol$_i$
14) Case2= Case2 ∪ Pol$_i$
15) exit=true
16) **else**
17) i=i+1,
18) **until i=**|accPNames| or exit=true
19) **If** Case1 ≠ ø
20) Let pol$_k$ be a randomly chosen policy in Case1
21) **Return** "Access granted"
22) **else**
23) **If** Case2 ≠ ϕ
24) *CreateProposal(*PolSet, acc*)*
25) **Foreach** pol$_i$ in PolSet s.t. Pol$_i$.ParName ∩ AccReqPNames ≠ ϕ
26) **If** ∃ acc.p.p$_i$ ∈ Legal_Values($\underline{p_z}$)
Case3= Case3 (Poli
27) CreateProposal()
28) endfor
29) If Case3
"Access denied"
30) end for

the missing or wrong ones. A more sophisticated approach is to determine such values on the fly while the proposal is generated. A possible solution in this sense is to make use of scripts, as proposed by Bertino et al. (2005).

Basically, the idea is to represent parameter values and context variables in a relational form and query them with ad hoc scripts. Scripts, in turn, may or may not be parametric. In the current work, we do not rely on a relational representation of service parameters and context variables. Parameters might also be dynamically determined by invoking ad hoc procedures having as input parameter names and returning legal

Table 1. Actions to be taken. Key: Acc-access request, pol-access control policy

Type of Acc compliance to pol	#pol	Action
Acc *is Totally Compliant* with pol	**0**	Verify if exists some pol such that *acc* ***PC*** *pol* holds
	≥1	For each *pol*, verify if *acc* ***PM*** *pol*
Acc *Partially Compliant* with pol	**0**	Deny access
	≥1	Request missing attributes for all ***PC*** *pol's*; then verify if *acc* ***PM*** *pol* holds
Acc *Parameter Matching* with pol	**0**	Negotiate parameters
	1	Grant access, if totally compliant, too
	> 1	Negotiate by sending NAPS for each of the parameter matching policies.
Acc *Fully Acceptable* by pol	**0**	Deny access
	1	Grant access
	> 1	Grant access randomly selecting a policy

values for those names. How these procedures are actually encoded depends, however, on the specific Web service implementation. As this aspect goes beyond the scope of this work, we do not further elaborate on it.

The negotiation algorithm is reported in Algorithm 4. The process is iterative and the NAP exchanges are carried on until the requesting agent, based on the received NAP, submits a fully acceptable request or the process is interrupted by one of the parties. The wish of closing the negotiation is explicitly notified to the counterpart, and it is represented in the algorithm by setting to *yes* the flag *end* in the NAP message.

Table 1 summarizes the actions taken by Ws-AC1 upon receiving an access request.

ENCODING WS-AC1 USING WS STANDARDS

In this section, we illustrate how the main components of the Ws-AC1 model can be represented according to the existing Web services standards. In particular, we provide the WSDL encoding of the Ws-AC1 *service description* of DrugStore service introduced in Example 1 and the specification of Ws-AC1 access control policies according to WS-Policy standard.

Encoding Ws-AC1 Services with WSDL

WSDL 2.0 is an XML language for describing Web services as a set of network endpoints that operate on messages. The WSDL description of a Web service consists of two parts. The first, called the abstract part, describes a Web service in terms of messages it sends and receives. In particular, the *types* clause specifies the structure of the exchanged messages using XML Schema. The sequence and cardinality of exchanged messages is described by message exchange patterns (MEP). An *operation* associates MEP with one or more messages. An *interface* groups these operations in a transport and wire independent manner. In the concrete part of the description, *bindings* specify the transport and wire format for interfaces. A *service endpoint* associates network address with a binding. Finally, a *service* groups the endpoints that implement a common interface.

The main issue in describing in WSDL a Web service supporting the Ws-AC1 model is to

specify the interactions between the requesting agents and the service provider corresponding to the identity attributes and parameter negotiation phases. Indeed, in Ws-AC1, the service provider and the requesting agent can be involved in different interactions, consisting each of a message exchange. For example, to an initial access request message, the service provider can reply either with a request for attributes message, if the access request is partially compliant with access control policies; with a NAP message if the requesting agent has not specified correct values for the service parameters; or with an access denied message. Further, during the identity attribute negotiation phase, the requesting agent can answer to a RFA message with a RSFA or with an access denied message, while during parameter negotiation phase, it can reply to a NAP message with another NAP message or it can simply refuse the last received NAP. Since we assume that the Web service supports only one operation, the operation can have multiple input and output messages, corresponding to Ws-AC1 Access Request, RSFA, RFA, NAP messages. Hence, it is necessary to specify which are the messages in input and output and the order according to which they are exchanged between the requesting agent and the service provider.

Here, we propose an extension to the WSDL language that allows one to specify the interactions between a requesting agent and the service provider. We have defined a new message exchange pattern and we extended the syntax of the WSDL **interface** element. Since WSDL 2.0 supports only MEP with at maximum one input message and one output message, we defined a new MEP called multi-in-multi-out. This MEP allows one to specify for an operation exposed by the Web service and multiple input and output messages, and allows one to specify which is the first input message to the operation. The input element representing the first message in input to the operation has an attribute **initial** set to the value true. Furthermore, to specify all the possible interactions, we added an **interaction_protocol_reference** subelement to the interface element, pointing to an XML document defining the possible interactions between the requesting agent and the service provider. The document consists of two components: the first defines all the possible interactions specifying the messages involved in the interaction; the second lists the allowable sequences of interactions. All interactions are included in an element **Interactions.** Each interaction is represented by an **Interaction** element having two child nodes, **InboundMessage** and **OutboundMessage**, and attributes **type** and **Id**. An InboundMessage is a message that the service accepts in input, an OutboundMessage is a message sent by the service. The attribute type specifies the type of interaction and can be one of the following: *Receive* for an inbound message, *ReceiveSend* for receiving an inbound message and then sending an outbound message as reply. The possible sequences of interactions are collected in a **Protocol** element. Each alternative sequence of interactions is represented by an **Interactions_Sequence** element containing an Interaction subelement for each interaction composing the sequence.

Another approach to specify interactions, quite similar to ours, has been recently proposed by Paurobally and Jennings (2005). This approach is based on the combined use of two Web services languages, WS-Conversation Language (WSCL) and WS-Agreement; it allows one to specify non-trivial interactions in which several messages have to be exchanged before the service is completed and/or the interaction may evolve in different ways depending on the state and needs of the requesting agents and of the service provider. With respect to such an approach, our method has the advantage that the interactions are referred in the WSDL service description and, hence, the requesting agent knows not only the information necessary to invoke the Web service but also how to communicate with the service provider without retrieving any further information.

Example 9. *To illustrate the proposed extensions to WSDL language, we consider the*

DrugStore service introduced in Example 1. In Figure 3 is reported a sketch of the DrugStore service description in WSDL, while Figure 4 represents the related XML document describing the possible interactions.

The types element specifies the structure of the messages characterizing Ws-AC1: AccessRequest, ResponseforAttributes, RequestforAttributes, NAP, NapRefused and AccesDenied. The interface element, called DrugStoreInterface, is constituted by only one operation, DrugStoreOnline. Since this operation can have more than one input and output message (AccessRequest, NAP, etc.), it supports the proposed MEP multi-in-multi-out. To specify that AccessRequest is the first message in input to DrugStoreOnline, its attribute initial is set to true.

The interface element contains an that is reference for the interaction_protocol and pointing to the XML document protocol_description.xml defining the interactions between the requesting agent and the service.

For example, the Interaction element AccRequest-to-Nap represents the interaction involving the messages AccessRequest and NAP. This interaction is of type ReceiveSend: It means that the service receives an AccessRequest message from the requesting agent and replies with a NAP message. According to the interaction protocol, AccRequest-to-Nap can be followed by Nap-to-Nap and Nap-to-AccGrant interactions or by Nap-to-AccGrant interaction, or by Nap-to-Nap and Nap-to-AccDen interactions or by a NapRefud-to-AccDen.

Specifying WS-AC1 Access Control Policies in WS-Policy

Ws-AC1 access control policies can be implemented in a format compliant to WS-Policy, the current standard for Web service policy specification. The main motivation for using WS-Policy to represent WS-AC1 policies is that, although access control policies are private to the Web service — and therefore should not in principle be made publicly available — representing them according to a standard might make it possible to exchange them among different Web services sites (end-points) where the same Web service is deployed.

WS-Policy is a specification that defines a general framework to describe a broad range of Web service policies. WS-Policy defines a policy as a collection of alternatives. Each alternative is a collection of assertions. Generally speaking, a policy assertion represents an individual requirement, a capability and so forth. For instance, a policy assertion can specify a particular authentication scheme, a transport protocol selection, a privacy policy, quality of service characteristic and so forth. The normal form schema of a policy according to WS-Policy is shown in Figure 5. In this schema, * indicates 0 or more occurrences of an item, while the [ ] indicates that the contained items must be treated as a group.

The **<wsp:Policy >** element is used as a policy container. The **<wsp:ExactlyOne>** element is used to define a collection of policy alternatives. The **<wsp:All>** element instead is used to define a collection of policies assertions, each of which must be satisfied. A policy alternative can be considered as a particular scheme of interaction that the requester of the service must be able to satisfy. Note that a requester can choose only a single policy alternative among the alternatives provided by the policy. Moreover, if a policy alternative is chosen, the requester must be able to satisfy all the policy assertions included in that policy alternative. Figure 6 reports an example of policy that adheres to WS-Policy specification. This example, taken from the WS-Policy specification, shows two policy alternatives, each composed by a single policy assertion. The policy has to be interpreted in the following way: If the first alternative is selected, only the Kerberos token type is supported; conversely, if the second alternative is selected, only the X.509 token type is supported.

The assertions used in a policy expression can be defined in public specifications, like WS-SecurityPolicy, WS-PolicyAssertion, or

Figure 3. DrugStore service description in WSDL

```
<?xml version="1.0"?>
<description name="DrugStore"
  targetNamespace=http://example.com/DrugStore.wsdl
     .....
  <types>
    <schema xmlns="http://www.w3.org/2000/10/XMLSchema">

      <element name="RequestforAttributes" type=" RequestforAttributesType">
      <element name="ResponseforAttributes" type=" ResponseforAttributesType">
      <element name="Nap" type=" NapType">
      <element name="AccessDenied" type=" xs:string" default=" Access Denied">
      <element name="NapRefused" type=" xs:string" default=" NapRefused">

       <complexType name="AccessRequest Type
          " .....
       </complexType>

       <complexType name="RequestforAttributesType ">
          .....
       </complexType>

       <complexType name="ResponseforAttributes Type ">
          .....
       </complexType>

       <complexType name="NapType ">
          .....
       </complexType>

    </schema>
  </types>

  <interface name=" DrugStoreInterface">
    <interaction_protocol_reference xlink: type= "simple" xlink:href="protocol_description.xml"/>
    <operation name="DrugStoreOnline" pattern="multi-in-multi-out"/>
      <input messageLabel="In" element="AccessRequest" initial="yes">
      <input messageLabel="In" element=" ResponseforAttributes "/ >
      <input messageLabel="In" element="Nap" />
      <input messageLabel="In" element="NapRefused"/>
      <output messageLabel="Out" element="RequestsforAttributes " />
      <output messageLabel="Out" element="DeniedAccess" />
      <output messageLabel="Out" element="Nap"/>
    </operation>
  </interface>

<binding name=" DrugStoreOnlineBinding "  interface=" DrugStoreInterface "
     type="http://www.w3.org/2005/05/wsdl/soap"
     wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">
     <operation ref=" DrugStoreOnline " /></binding>

<service name="DrugStoreService" interface=" DrugStoreInterface ">
   <endpoint name=" DrugStorePort " binding=" DrugStoreOnlineBinding "
         address ="http://example.com/DrugStoreOnline"/>
   </endpoint>
 </service>
</description>
```

Figure 4. protocol_description.xml

```
<Protocol_Description>
 <Interactions>
  <Interaction type = "ReceiveSend " Id= "AccRequest-to-AccDen" >
    <InboundMessage  type = "AccessRequest "/>
    <OutboundMessage type = "DeniedAccess "/>
  </Interaction>

  <Interaction type = "ReceiveSend " Id="AccRequest-to-Nap ">
    <InboundMessage  type = "AccessRequest "/>
    <OutboundMessage type = "Nap" />
  </Interaction>

  <Interaction type = "ReceiveSend " Id="AccRequest-to-ReqForAttr" >
    <InboundMessage   type = "AccessRequest "/>
    <OutboundMessage type = "RequestsforAttributes " />
 </Interaction>

  <Interaction type = " ReceiveSend" Id="RespForAttr-to-Nap">
    <InboundMessage  type = "ResponseforAttributes" />
    <OutboundMessage type = "Nap" />
  </Interaction>

  <Interaction type = " Receive" Id="RespForAttr-to-ACCGran">
    <InboundMessage  type = "ResponseforAttributes" />
   </Interaction>

     <Interaction type = "ReceiveSend " Id="Nap-to-Nap" max_num_
act="unbounded">
            <InboundMessage  type = "Nap" />
            <OutboundMessage type = "Nap" />
  </Interaction>

  <Interaction type = "Receive " Id="Nap-to-AccGran" >
   <InboundMessage  type = "Nap" />
  </Interaction>

  <Interaction type = "ReceiveSend " Id= "NapRefus-to AccDen">
    <InboundMessage  type = "NapRefused "/>
    <OutboundMessage type = "DeniedAccess "/>
  </Interaction>

  <Interaction type = "ReceiveSend " Id=" ResponseForAttr-to-AccDen">
    <InboundMessage  type = "ResponseforAttributes "/>
    <OutboundMessage type = "DeniedAccess "/>
  </Interaction>

  <Interaction type = "ReceiveSend " Id=" Nap-to-AccDen" >
    <InboundMessage  type = "Nap"/>
    <OutboundMessage type = "DeniedAccess "/>
  </Interaction>
 </Interactions>

<Protocol>
 <Interactions_Sequence Id="SQ1">
     <Interaction name= "AccRequest-to-AccDen"/>
 </ Interactions_Sequence >
```

(continued on the following pages)

Figure 4. protocol_description.xml (cont.)

```
 <OR>
<Interactions_Sequence Id="SQ2">
    <Interaction name="AccRequest-to-Nap"/>
    <Interaction name="Nap-to-Nap"/>
    <Interaction name="Nap-to- AccDen "/>
</ Interactions_Sequence>
 <OR>
<Interactions_Sequence Id="SQ3">
    <Interaction name="AccRequest-to-Nap"/>
    <Interaction name="Nap-to-Nap"/>
            <Interaction name="Nap-to-ACCGran"/>
</ Interactions_Sequence>
 <OR>
<Interactions_Sequence Id="SQ4">
    <Interaction name="AccRequest-to-Nap"/>
    <Interaction name=" Nap-to-AccGran "/>
</ Interactions_Sequence>
 <OR>
<Interactions_Sequence Id="SQ5">
    <Interaction name="AccRequest-to-Nap"/>
    <Interaction name="NapRefus-to-AccDen"/>
</ Interactions_Sequence>
 <OR>
<Interactions_Sequence Id="SQ6">
    <Interaction name="AccRequest-to-ReqForAttr"/>
      <Interaction name="ResponseForAttr-to-OK AccGran
"/>
</ Interactions_Sequence>
 <OR>
<Interactions_Sequence Id="SQ7">
    <Interaction name="AccRequest-to-ReqForAttr"/>
    <Interaction name="ResponseForAttr-to-AccDen "/>
</ Interactions_Sequence>
 <OR>
<Interactions_Sequence Id="SQ8">
    <Interaction name="AccRequest-to-ReqForAttr"/>
    <Interaction name="ReqForAttr-to-Nap/>
    <Interaction name=" Nap-to- AccGran "/>
</ Interactions_Sequence>
 <OR>
</ Interactions_Sequence>
   <OR>

<Interactions_Sequence Id="SQ9">
  <Interaction name="AccRequest-to-ReqForAttr"/>
  <Interaction name="RespForAttr-to-Nap"/>
  <Interaction name="Nap-to-Nap"/>
  <Interaction name="Nap-to- AccGran "/>
</ Interactions_Sequence>
 <OR>
<Interactions_Sequence Id="SQ10">
  <Interaction name="AccRequest-to-ReqForAttr"/>
  <Interaction name="RespForAttr-to-Nap"/>
  <Interaction name="NapRefus-to-AccDen"/>
</ Interactions_Sequence>
 <OR>
<Interactions_Sequence Id="SQ11">
```

Figure 4. protocol_description.xml (cont.)

```
<Interaction name="AccRequest-to-ReqForAttr"/>
<Interaction name="RespForAttr-to-Nap"/>
<Interaction name="Nap-to-Nap"/>
<Interaction name="NapRefus-to-AccDen"/>
        </ Interactions_Sequence>

</Protocol>

</Protocol_Description>
```

Figure 5. Normal form schema of a policy according to WS-Policy

```
<wsp:Policy ... >
 <wsp:ExactlyOne>
 [<wsp:All> [<Assertion ...> ... </Assertion> ]* </wsp:All> ]*
 </wsp:ExactlyOne>
</wsp:Policy>
```

they can be defined by the entity owning the Web service. The assertions of the first type are named standard assertions and are understandable potentially from any client. The assertions defined by the entity owning the Web service instead can be understood only from those clients to which the entity has already released the specifications.

To encode Ws-AC1 access control policies, we define a new type of policy assertions, since no public specification we are aware of defines assertions suitable for expressing attribute conditions and parameter constraints required by WS-AC1 policy formalism (see Definitions 5 and 6).

All the WS-AC1 policy components are suitable to be represented as policy assertions. The associations between WS-AC1 policy components and the WS-Policy assertions are summarized in Table 2.

Figure 5 reports an example of a simple Ws-AC1 policy represented in a format compliant to WS-Policy.

Example 10. *Consider the policy pol_3 introduced in Example 5. The representation of pol_3 in a format compliant to WS-Policy is reported in Figure 7.*

Algorithm 5 formalizes the steps necessary to represent one or more Ws-AC1 policies in a format compliant to WS-Policy. Basically, the algorithm merges all the Ws-AC1 policies associated with the same service into a single policy WS-Policy compliant. Given a service name, each Ws-AC1 policy that applies to that service or to its related class becomes a WS-Policy alternative. Note that because all the Ws-AC1 policies referring to the same service are merged into a single policy conforming to WS-Policy, such policy can be uniquely identified by the service name (see, for instance, Example 10).

Functions AttributeName(CA_i), Operator(CA_i) and AttributeValue(CA_i) extract, respectively, the name of the identity attributes, the comparison operator and the costant K appearing in an attribute condition CA_i. Function Body($Constr_i$) returns the set of conditions in the body of constraint $Constr_i$, while, function Head($Constr_i$), extracts the head of $Constr_i$

Figure 6. Example of policy

```
<wsp:Policy xml:base=http://dico.unimi.it wsu:Id=MyPolicy>
<wsp:ExactlyOne>
<wsp:All>
<wsse:SecurityToken>
<wsse:TokenType>wsse:Kerberosv5TGT</wsse:Token-
Type>
/wsse:SecurityToken>
</wsp:All>
<wsp:All>
<wsse:SecurityToken>
<wsse:TokenType>wsse:X509v3</wsse:TokenType>
</wsse:SecurityToken>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
```

Table 2. WS-AC1 policy assertions

WS-AC1 access control component	**Assertion Type**
St	<ServiceIdReference></ServiceIdReference>
C	<AttributeConditions> <AttributeCondition>+ <AttributeName></AttributeName> <Operator></Operator>? <AttributeValue></ AttributeValue>? </AttributeCondition> </AttributeConditions>
ParamSet	<ParameterSet> <ParameterName></ParameterName>+ </ParameterSet>
ParamConstr	<ParameterConstraints> <Constraint Id>+ <Conditions> <Condition></Condition>+ </ Conditions> <Consequence></Consequence> </Constraint>

SYSTEM IMPLEMENTATION

The Ws-AC1 implementation consists of a client program and service system. The client is a Web-based application, developed using JSP. WS-AC1 service infrastructure has been developed using Java, Tomcat and AXIS. Tomcat is a servlet/JSP container, while AXIS is a SOAP engine, which makes transparent to the developer the management of SOAP messages.

The service architecture is composed of different Java classes that manage the exchange of the different type of messages supported by Ws-AC1, such as RFA and NAP and the identity attributes and service parameters negotiation protocol. In particular, RFA and NAP composi-

Figure 7. A Ws-AC1 policy represented in a format compliant to WS-Policy compliant

```
pol= < DrugStore; {CustomerId = ChicagoHospital , DoctorPrescriptionId = 34567};
      {Price}; {Price = High ← StockLevel < 10,
MedicineActivePrinciple = sildenafil citrato, Price = Low} >
<wsp:Policy >
 <wsp:ExactlyOne>
  <wsp:All>
   <ServiceIdReference>
    DrugStore
   </ServiceIdReference>
   <AtributeConditions>
    <AttributeCondition>
     <AttributeName>CustomerId</AttributeName>
     <Operator>=</Operator>
     <AttributeValue>ChicagoHospital</AttributeValue>
    </AttributeCondition>
    <AttributeCondition>
     <AttributeName>DoctorPrescriptionId</AttributeName>
     <Operator>=</Operator>
     <AttributeValue>34567</AttributeValue>
    </AttributeCondition>
   </AttributeConditions>
   <ParameterSet>
   <ParameterName>Price</ParameterName>
   </ParameterSet>
   <ParameterConstraints>
    <Constraint Id="1">
     <Conditions>
      <Condition>StockLevel < 10</Condition>
      <Condition>MedicineActivePrinciple=sildenafil citrato</Condition>
      <Condition> Price = Low </Condition>
     </ Conditions>
     <Consequence> Price = High </Consequence>
    </Constraint>
  </wsp:All>
 </wsp:ExactlyOne>
</wsp:Policy>
```

tion is executed by two different classes: class *CreateRFA* to generate request for attributes, and class *CreateNAP* to generate NAP. The system also uses a database managed by Oracle 9.2 to store the access control policies and the context variables on the basis of which access control is enforced.

An Example of Execution

In this section we show an example of the case study implemented to test Ws-AC1 applicability. The client program is executed specifying a URL address (see Figure 8). In the example, several services are proposed to users browsing the Web.

Once one of the available services is selected, the client is asked to select the parameter values to invoke the service and also to input his or her authentication attributes. The forms to fill out are reported in Figures 9 and 10.

Once the required fields are filled, Ws-AC1 processes the request and then returns a reply. If new attributes are needed to satisfy any policy for the invoked service, an attribute request is displayed, as shown in Figure 10. In case a parameter negotiation is possible, the

Algorithm 5. Algorithm for translating Ws-AC 1 policies in WS-policy compliant format.

Input:
s: the service identifier;
PolSet = {pol_1, pol_2, ., pol_n}: set of Ws-AC1 access control policies applying to the service s where each pol_i = < st; C; ParamSet; ParamConstr ; ParamSet >

Output:
OutPol: a Ws-Policy compliant policy

1) OutP ol = "<wsp:Policy wsu:Id =" ⌣ s ⌣ ">"
2) OutP ol = OutPol ⌣ "<wsp:ExactlyOne>"
3) foreach pol_i ∈ PolSet
4) OutPol = OutPol ⌣ "<wsp:All>"
5) OutPol = OutPol ⌣ "<ServiceId Reference>" ⌣ " s " ⌣ "</ServiceId Reference>"
6) OutPol = OutPol ⌣ "<AttributesConditions>"
7) foreach CA_i in pol_i.C
8) OutPol = OutPol ⌣ "<AttributeCondition>"
9) OutPol = OutPol ⌣ "<AttributeName>"
10) OutPol = OutPol ⌣ " *AttributeName(CA_i)* "
11) OutPol = OutPol ⌣ "</AttributeName>"
12) OutPol = OutPol ⌣ "<Operator>"
13) OutPol = OutPol ⌣ " *Operator(CA_i)* "
14) OutPol = OutPol ⌣ "</Operator>"
15) OutPol = OutPol ⌣ "<AttributeValue>"
16) OutPol = OutPol ⌣ " *AttributeValue(CA_i)* "
17) OutPol = OutPol ⌣ "</AttributeValue>"
18) OutPol = OutPol ⌣ "<AttributesCondition>"
19) OutPol = OutPol ⌣ "<ParameterSet>"
20) foreach P_i in pol_i.ParamSet
21) OutPol = OutPol ⌣ "<ParameterName>"
22) OutPol = OutPol ⌣ "<AttributeName>"
23) OutPol = OutPol ⌣ "</ParameterName>"
24) OutPol = OutPol ⌣ "</ParameterSet>"
25) OutPol = OutPol ⌣ "<ParameterConstraints>"
26) foreach $Constr_i$ in pol_i.ParamConstr
27) OutPol = OutPol ⌣ "<Constraint Id = " ⌣ " i " ⌣ " >"
28) OutPol = OutPol ⌣ "<Conditions>"
29) foreach $Cond_i$ in *Body($Constr_i$)*
30) OutPol = OutPol ⌣ "<Condition>"
31) OutPol = OutPol ⌣ " $Cond_i$ "
32) OutPol = OutPol ⌣ "</Condition>"
33) OutPol = OutPol ⌣ "</Conditions>"
34) OutPol = OutPol ⌣ "<Consequence>"
35) OutPol = OutPol ⌣ " *Head($Constr_i$)* "
36) OutPol = OutPol ⌣ "</Consequence>"
37) OutPol = OutPol ⌣ "</Constraint>"
38) OutPol = OutPol ⌣ "</ParameterConstraints>"
39) OutPol = OutPol ["</wsp:All>"
40) OutPol = OutPol ["</wsp:ExacltyOne>"
41) OutPol = OutPol ["</wsp:Policy>"
42) return OutPol

Figure 8. Service selection

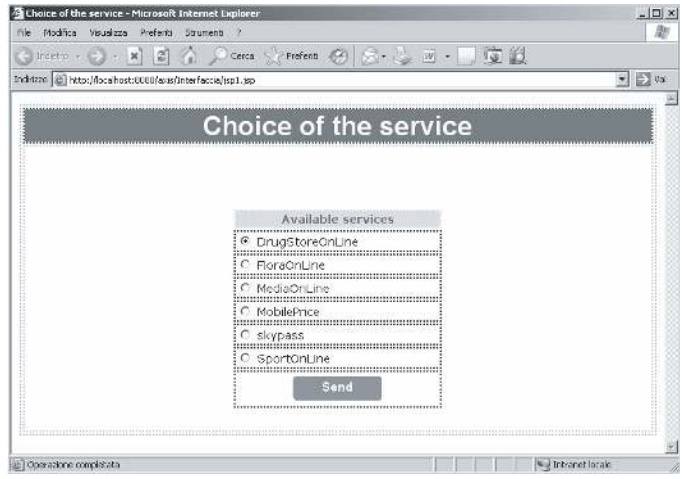
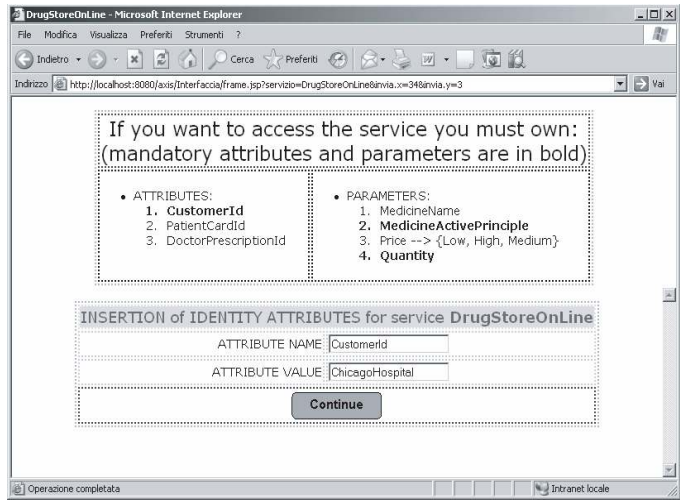


Figure 9. Identity attribute insertion



message in Figure 11 is displayed. Here, the user is allowed either to select the counterproposal proposed by the system, or he or she can submit a new request.

Based on the attributes and the parameters sent in the new counterproposal, access is either granted or denied.

RELATED WORK

Security support is one of the major challenges for the wide-scale adoption of service-oriented computing. Important security issues are related to secure message transmission, access control and identity management. Our work is related to the development of policy-driven access control models for Web services.

Figure 10. Service parameter insertion

Figure 11. Request for additional identity attributes

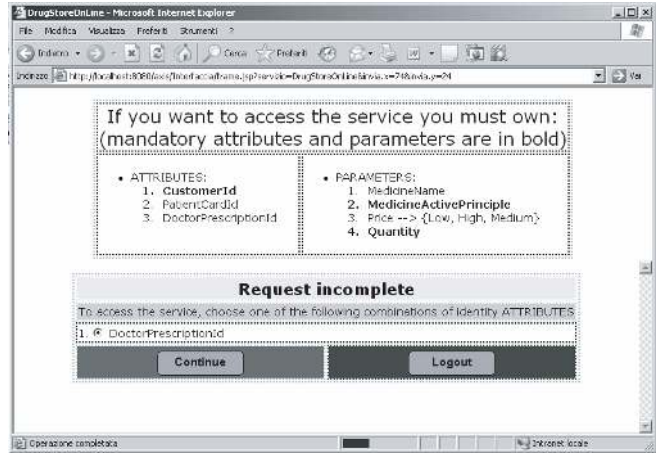
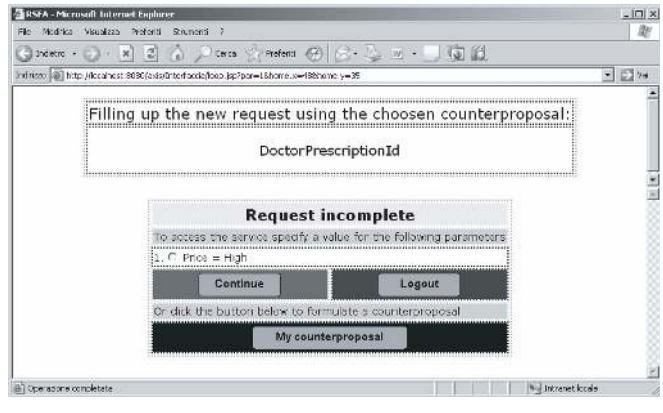


Figure 12. Parameter negotiation



Policy-driven access control has been extensively investigated in the last years, but only limited work has been carried out for access control models specifically tailored to Web services. The most significant proposals are by OASIS XACML profile for Web services (2003); Wonohoesodo and Tari (2004); Gun and Wang (2002); Ardagna, Damiani, De Capitani di Vimercati and Samarati (2004); and Kagal Paolucci, Srinivasan, Denker, Finin and Sycara (2004).

The most recent version of OASIS XACML profile for Web-services (referred to as XACML2) extends the former version of the standard to address access control requirements of Web services. While XACML provides an extensible, XML-encoded language to express both policies and access control decision requests/responses, the extensions proposed in XACML2 allow one to express policies associated with Web services end-points. XACML2 supports the specification of policies for a WSDL port (the whole service), WSDL operation or WSDL message, or a combination of them. The policies associated with a port are represented by a <PolicySet> element, that in turn can include other <PolicySet> elements representing the policies for an operation or a message. Each <PolicySet> element contains <Policy> elements, which are associated with a single aspect of an end-point policy where an aspect is an independent set of technical features and parameters associated with the use of the Web service (for example, data rate allocation). The <Target> subelement of a <Policy> element identifies the set of conditions governing the aspect (referred to as objective) of the end-point policy. Further, a <Policy> element must contain a set of <Rule> elements that define acceptable alternative solutions for achieving the objective. A < Rule> element includes a set of <Apply > elements containing predicates expressing conditions on attributes. Attributes can be of three different types: unconstrained, constrained and authorized. An unconstrained attribute is such that its value can be set by the policy-user; for instance, the minimum time between re-transmission of an unacknowledged message. The value of a constrained attribute, on the other hand, is out of the control of the policy-user. Examples of constrained attributes are environmental attributes like time, or subject's attributes, the values of which are set by some an entity or user different from the policy-user (for instance, the status of the subject in a customer loyalty program). The value of an authorized attribute is asserted by an authority, like the policy-user's role.

Another interesting feature of the new XACML2 is that it adopts the XACML mechanisms for combining either multiple policies or multiple rules in a single policy, to blend the policies of the service consumer, expressing the preference/requirements of the consumer about the service provision and the policies of the service provider. The <PolicySet> element, resulting from the combination process, represents a solution to both the consumer and provider policy statements. A service invocation using this solution conforms with the policy of both the consumer and the provider.

The XACML profile for Web services and Ws-AC1 have similar features. Both XACML2 and Ws-AC1 allow the definition of policies at a level of the entire service or at a level of the single service operation; in addition, XACML2 supports the specification of policies at a message level. They both support the definition of multiple policies for the same Web service: In XACML2, this is achieved by defining a <PolicySet> element having a <Target/Resource> subelement referring the WSDL port. In Ws-AC1, the policies related to the same service have the same service identifier or belong to the same service, while XACML2 does not support this capability. In fact, Ws-AC1 allows the specification of policies for a specific instance of a Web service or for a group of services. Further, the formulation of the policies is based on the specification of constraints on attributes. A Ws-AC1 policy expresses conditions against the identity attributes of the service consumer and constraints specifying the acceptable values of service parameters on the base of the context and of the values of the other parameters, while a <Rule> element in XACML2 contains

predicates that are constraints on attributes. The identity attributes of Ws-AC1 are equivalent to the authorized attributes of XACML2, while the context variables and the service parameters correspond, respectively, to the environmental attributes and the constrained attributes in XACML profile. Furthermore, the negotiation capabilities of Ws-AC1 can be matched with the process of combining the policies of the service consumer and of the service provider. Both can be seen as an approach to drive the consumer toward a specification of a service invocation compliant with the policies, thus reaching a trade-off between the requirements of the consumer and the provider.

In Wonohoesodo and Tari (2004), the authors propose two RBAC-access control models: SWS-RBAC for single Web services, and CWS-RBAC for composite Web services. Both enforce access control at two levels, the service level and attribute level. In both models, a service has minimum access modes applied to one or many attributes (an attribute can be either a service's parameter or a returned value) and a role is associated with a list of services; clients, who are assigned the role, have permission to execute. In addition, a role is related with a list of attributes the client has access to and the access types. The permission to invoke a service is granted to a client if he is assigned to a role that has the requested service granting to it and which satisfies all the minimum access requirements on attributes used by the service. In the CWS-RBAC model, the role to which a client is assigned to access a composite service must be a global role, who is mapped on local roles of the service providers of the component Web services. As in Ws-AC1, access control is enforced at service and service parameter levels: Instead of defining the set of service parameters values acceptable to access, Wonohoesodo and Tari specify the access modes (read, write, modify) on service parameters the client must have to invoke the service.

Gün Sirer and Wang (2002) proposed an approach for formally specifying and enforcing security policies that is independent from the Web service implementation. Security policies are specified using a language called WebGuard based on temporal logic and are processed by an enforcement engine to yield site- and platform-specific access control. This code is integrated with a Web server and platform-specific libraries to enforce the specified policies on a given Web service. The emphasis is posed on automating and componentizing security and access control services for Web services. In our work, we focus on specification of flexible access control policies, and provide mechanisms for enforcing access control in an adaptive manner. As we do not deal with automating our security policies, we believe it might be interesting to integrate our approach with that of Gün Sirer and Wang (2002).

Another significant work is the one of Ardagna et al. (2004). They present a Web service architecture for enforcing access control policies, which are expressed in WS-Policy. The architecture is similar to the one proposed in the XACML standard and is characterized by three main components: PDP (Policy Decision Point), PEP (Policy Enforcement Point) and PAP (Policy Administration Point). The PDP realizes the interface between a service and the access control architecture. When a client requests to invoke a service, the service forwards the request to the PDP, which, in turn, sends it to the PEP. The PEP asks the PAP for the policies applicable to the request and evaluates it against the applicable policies. Then, it returns the final decision to the PDP, which issues the service access. Compared with Ws-AC1, the proposed model enforces access control only on the base of the attributes in credentials submitted by the clients. No negotiation capability for the attributes is offered: If the credentials of the client do not match the policies, the system raises an exception and denies access to the service. It would be interesting to investigate if the architecture proposed by Ardagna et al. can be applied to implement WS-AC1.

The work from Kagal et al. (2004) addresses security of semantic Web services by using policy annotations for OWL-S service descriptions. An OWL-S description, similar to the service description of Definition 1, com-

prises a profile, process model and grounding of the Web service. The authors add to these basic annotations other annotations about security, trust and privacy policies for the semantic Web. These annotations are used by the client to select the service to invoke. Indeed, the authors propose an algorithm to combine the security requirements of the client with the security policies of the service provider in the OWL-S service description. The result of the combination process is used to select the service. It will be interesting to investigate how and if it is possible to integrate Kagal et al.'s approach with ours to obtain a comprehensive system protecting privacy and enforcing authorizations adaptively and flexibly.

In 2004, a first preliminary model for Web services access control was proposed by Bertino et al. The system, called WS-Aba, supports attribute-based access control and a first simple notion of access negotiation of Web service parameters. However, no actual protocol for supporting access negotiation was provided. In a subsequent work, the authors designed WS-AC (Bertino, Squicciarini, Paloscia, & Martino, 2004), providing a more sophisticated approach for parameter negotiations. A formalization of the protocol was developed, along with algorithms showing how to encode access control policies with WS-Policy standard. The system presented in this paper extends and enhances WS-AC under several aspects. First, WS-AC relied on a relational representation of Web services parameters and context variables. Ws-AC1 is not tightly coupled with any specific representation of the data to be used, offering more flexibility on data representation and encoding. In WS-AC, policies were specified only at a fine-grain level, and no possibility of encoding coarse-grained policies was provided. Further, in WS-AC, user authentication was not adaptive: a subject could only submit the requested attributes once and was not allowed to adjust requests. Also, the notion of context was vague and exploited only for negotiations. Other relevant extensions of Ws-AC1 deal with the effort we made in encoding all the messages using the WS stack. This led us to notice an important shortcoming in one of the adopted standards; that is, WSDL. As such, we also proposed some extensions to obtain a standard compliant access control system.

CONCLUSION

In this paper, we presented an adaptive access control model for Web services. The model is characterized by varying protection granularities, in that an access control policy can be associated with both a single service with specific parameter values or with a set of services. Such a range of granularity allows one to specify general policies and to refine them as needed for specific services. The other novel characteristic of our model is related to negotiation capabilities. The model allows two parties to negotiate both the identity attributes that a requesting agent has to submit and the values to be used for the service parameters. In this paper, we provided formal definitions of the basic concepts of our model as well as all relevant algorithms implementing the main functions of our model. As part of our work, we also developed a specification of our model in terms of the WSDL standard. An important result we have obtained here is the development of an extension of WSDL; such an extension is required to model the fact that an operation may have multiple input and output messages and not a single input and output message, as in the current version of the standard.

In current work, we rely on heuristics to determine how to modify the original access request when a NAP is to be proposed. We would, however, like to automate the process of parameter selection based on some more formal reasoning. Further, we are currently exploring the possibility of adopting Ws-AC1 for composite services. We are thus evaluating extensions of the current system in order to fully support Ws-AC1 authentication methods in composite services. Other issues we plan to explore are related to attacks the Ws-AC1 system might be subject to. For instance, it is not clear what can be learned by an attacker on a test-and-fail basis. As additional future work, we plan to investigate the integration

of our model with existing standards, such as XACML. Finally, since this aspect is still missing in the WS-Security stack, we would like to integrate our model with mechanisms supporting user privacy, to allow clients to confidently send private credentials to unknown services.

ACKNOWLEDGMENTS

The developments presented in this paper were partly funded by the European Commission through the IST program under Framework 6 grant 001945 to the Trustcom Integrated Project, by the National Science Foundation under Grant No. 0430274 and the sponsor of CERIAS.

REFERENCES

Ardagna, C., Damiani, E., De Capitani di Vimercati, S., & Samarati, P. (2004). A Web service architecture for enforcing access control policies. In *Proceedings of the 1st International Workshop on Views on Designing Complex Architectures*, Bertinoro, Italy (pp. 25-35).

Bertino, E., Ferrari, E., & Squicciarini, A.C. (2003). X-TNL: An XML-based language for trust negotiations. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, Lake Como, Italy (pp. 81-84).

Bertino, E., Squicciarini, A.C., & Mevi, D. (2004). A fine-grained access control model for Web services. In *Proceedings of the IEEE International Conference on Service Computing (SCC 2004)*, Shanghai, China (pp. 33-40).

Bertino, E., Squicciarini, A.C, Paloscia, I., & Martino, L. (2005). Ws-Ac: A fine grained access control system for Web services (unpublished). Accepted for publication in *World Wide Web Journal*.

Damianou, N., Dulay, N., Lupu, E., & Sloman, M. (2001). The ponder policy specification language. In *Proceedings of the IEEE International Workshop on Policies for Distributed Systems and Networks*, Bristol, UK (pp. 18-38).

Gavriloaie, R., Nejdl, W., Olmedilla, D., Seamons, K.E., & Winslett, M. (2004). No registration needed: How to use declarative policies and negotiation to access sensitive resources on the Semantic Web. In *Proceedings of the 1st European Semantic Web Symposium*, Heraklion, Greece (pp. 342-356).

Gün Sirer, E., & Wang, K. (2002). An access control language for Web services. In *Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, Monterey, California, USA (pp. 23-30).

Herzberg, A., Mihaeli, J., Mass, Y., Naor, D., & Ravid, Y. (2000). Access control systems meets public infrastructure, or: Assigning roles to strangers. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, USA (pp. 2-14).

IBM, BEA Systems, Microsoft, SAP AG, Sonic Software, VeriSign. (2004). *WS-policy-Web services policy framework*. Retrieved from http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-s/dnglobspec/html/ws-policy.asp

Kagal, L., Paolucci, M., Srinivasan, N., Denker, G. Finin, T., & Sycara, K. (2004). Authorization and privacy for semantic Web services. In *Proceedings of the AAAI 2004 Spring Symposium on Semantic Web Services*, Palo Alto, California, USA (pp. 50-56).

OASIS eXtensible Access Control Markup Language (XACML) Version 2.0. (n.d.). Committee draft 02. Document identifier: access_control-xacml-2.0-core-spec-cd-02. Retrieved September 30, 2004, from http://docs.oasis-open.org/xacml/access_control-xacml-2.0-core-spec-cd-02.pdf

OASIS XACML. (n.d.). *Profile for Web-services* (Working Draft 04). Document identifier: draft-xacml-wspl-04. Retrieved September 29, 2004, from http://docs.oasis. open.org/committees/ documents.php?wg_abbrev=xacml

Paurobally, S., & Jennings, N. R. (2005).

Protocol engineering for Web services conversations. *International Journal of Engineering Applications of Artificial Intelligence, 18*(2), 237-254.

Ryutov, T., Zhou, L., Neuman, C., Leithead, T., & Seamons, K.E. (2005). Adaptive trust negotiation and access control. In *Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT 2005)*, Stockholm, Sweden (pp. 139-146).

Ryutov, T., & Neuman, C. (2002). The specification and enforcement of advanced security policies. In *Proceedings of the IEEE International Workshop on Policies for Distributed Systems and Networks*, Monterey, California, USA (pp. 128-138).

Ryutov, T., Neuman, C., Kim, D., & Zhou, L. (2003). Integrated access control and intrusion detection for Web servers. *IEEE Transactions on Parallel and Distributed Systems, 14*(9), 841-850.

Wonohoesodo, R., & Tari, Z. (2004). A role based access control for Web services. In *Proceedings of the IEEE International Conference on Service Computing (SCC 2004)*, Shanghai, China (pp. 49-56).

World Wide Web Consortium. (2005). *WSDL-Web services description language 2.0* (W3C Working Draft). Retrieved from http://www.w3.org/TR/2005/WD-wsdl20-primer-20050510/

Yu, T., Winslett, M., & Seamons, K. (2003). Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security, 6*(1), 1-42.

ENDNOTE

1 The comparison operators we refer to are: $\neq, <, >, =, \leq, \geq$.

Elisa Bertino is professor of computer science and ECE and research director of CERIAS at Purdue University. Previously, she was a full professor in the Department of Computer Science at the University of Milan (Italy). From 1990-1993, she was a full professor in the Department of Computer and Information Science at the University of Genova (Italy). Her main research interests include object-oriented databases, distributed databases, deductive databases, multimedia databases, interoperability of heterogeneous systems, integration of artificial intelligence and database techniques, database security. In those areas, she has published more than 250 papers in refereed journals, such as ACM Transactions on Database Systems*,* ACM Transactions on Office Information Systems*,* IEEE Transactions on Knowledge and Data Engineering*,* Acta Informatica*,* Information Systems*, and in proceedings of international conferences and symposia. She has participated in several research projects sponsored by the Italian National Research Council and the European Economic Communities. She is or has been on the editorial board of the following scientific journals:* ACM Transactions on Information and System Security*,* IEEE Transactions on Knowledge and Data Engineering*,* Data & Knowledge Engineering Journal*,* Journal of Computer Security*,* International Journal of Theory and Practice of Object Systems*,* Journal of Distributed and Parallel Databases*,* Very Large Databases (VLDB) Journal*,* International Journal of Information Technology.

Anna Cinzia Squicciarini is a post doc at Purdue University, US. She received her Phd at University of Milan, Italy, in October 2005. During fall 2003 she was a visiting researcher at Swedish Institute of Computer Science, Stockholm. She also was a research scholar at Colorado State University, Fort Collins (CO), U.S. during the spring of 2004; and at Purdue University in the

spring of 2005. Her main research interests include trust negotiations, privacy, models and mechanisms for privilege and contract management in virtual organizations. Currently, she is a visiting scholar at Purdue University, West Lafayette, where she is exploring research issues related with identity management and Web sevice access control models. She has given talks at several research institutions in Italy and abroad, including IBM T.J. Watson Labs and Colorado State University. She has served as a PC member of Semantic Web and Policy WORKSHOP (SWPW), ECIW 2006, STD3S, and is reviewer of IEEE magazines and journals like IEEE Security & Privacy*,* IEEE Computing*,* ACM TISSEC *and others.*

Lorenzo Martino is a visiting assistant professor at the Computer and Information Technology Department and at the Cyber Center of the Purdue University. Previously he was a senior researcher at the Department of Computer Science and Communication at the University of Milan, Italy, where he leads research efforts in the area of security for virtual organizations. Before joining the University, he held positions as senior engineer and project manager at various companies; in this capacity, he lead several projects related to information management and application interoperability in the financial and trading on-line areas. His major research interests include security for Web services and trust negotiation.

Federica Maria Francesca Paci is a PhD Student at the University of Milan, Italy. She received a degree in computer science from the University of Milan in February 2004 with full marks. During the spring of 2005 Federica was a research scholar at Computer Science Department and CERIAS of Purdue University, West Lafayette, U.S.A. Her main research interests include the development of access control models for constraint workflow systems, Web services access control models and secure distribution of XML documents.