Semantics-based Access Control Approach for Web Service

Zhengqiu He, Lifa Wu, Huabo Li, Haiguang Lai, Zheng Hong
Institute of Command Automation, PLAUST, Nanjing, China
hzqzyl@163.com

***Abstract*—Due to the open and distributed characteristics of web service, its access control becomes a challenging problem which has not been addressed properly. In this paper, we show how semantic web technologies can be used to build a flexible access control system for web service. We follow the Role-based Access Control model and extend it with credential attributes. The access control model is represented by a semantic ontology, and specific semantic rules are constructed to implement such as dynamic roles assignment, separation of duty constraints and roles hierarchy reasoning, etc. These semantic rules can be verified and executed automatically by the reasoning engine, which can simplify the definition and enhance the interoperability of the access control policies. The basic access control architecture based on the semantic proposal for web service is presented. Finally, a prototype of the system is implemented to validate the proposal.**

***Index Terms*—web service, access control, RBAC, semantics**

I. INTRODUCTION

As a novel distributed computing model, web service is an effective mechanism for data and application integration on web. Many works have been done towards making web service technology a suitable solution for areas such as e-business, e-government and SOA (Service Oriented Architecture), etc. In web services, service requests and responses are conveyed by SOAP [1], which has the ability to pass unhinderedly through firewalls. If false claims or malicious contents are contained in SOAP messages, it is possible to result in unauthorized access even damage to the internal applications. In addition, some web services may be just open to the specific users group, only requesters who satisfy the corresponding conditions can be permitted to access those services. Reliable access control is thus a fundamental requirement for the acceptance of web service by organizations [2].

Web service environment is characterized by its openness, dynamics, and loose coupling. The interaction is between remotely located parties who may have little or no knowledge about each other. Access control of Web service is thus required to cross the border of security domains and address the movement of unknown users across borders so that access to services can be granted [3, 4]. Current access control approaches like Role-based access control (RBAC) [5] generally assume that the identity is established and execute roles assignment statically based on the identities of users, which can be restrictive in web service environment [6, 19]. In order to address cross-domain movement of users, a new approach called Attribute-based access control (ABAC) has been proposed [7, 8]. The basic idea of ABAC is that, it defines permissions based on three types of attributes, including subject attributes, resource attributes and environment attributes. However, the specification and maintenance of ABAC policies has turned out to be complex and error-prone, especially if heterogeneous attribute schemes are involved [8]. In addition, ABAC assigns permission directly to users rather than assigning roles to abstract permission, which violates the principles of scalability and manageability that motivates developers to use RBAC [9]. Consequently, in this paper, we follow the RBAC model and borrow ideas from ABAC. Namely, roles assignments are executed based on the attributes of credentials provided by the users, rather than the identities. These credentials can be X.509 certificate, Kerberos ticket, public key, etc, which are issued and signed by the trusted authorities and can be accepted across domains.

The concepts and functions embedded in access control model need to be represented by an appropriate policy language [10]. Presently, researchers have developed several kinds of policy languages that can be used for access control. These include industry standards like XACML [11], but also academic achievements like Ponder [12], Rei [13], and KAoS [14]. XACML is a general-purpose authorization policy model and XML-based specification language, which essentially enforces attribute based access control. Ponder is an object oriented language for specifying security and management policies where policies are rules defining behavioral choices. KAoS and Rei are Semantic Web based languages. KAoS uses OWL (Web Ontology Language, [15]) as the basis for representing and reasoning about policies within Web Services, Grid Computing, and multi-agent system platforms. The KAoS approach, however, relying on pure OWL capabilities, encounters some difficulties with regard to the definition of certain kinds of policies, specifically those requiring the definition of variables [18]. Rei adopts OWL-Lite to specify policies and can reason over any domain knowledge expressed in either RDF (Resource Description Framework) or OWL. Semantic-based policy languages have some specific advantages than others.

doi:10.4304/jcp.6.6.1152-1161

They allow policies to be described over heterogeneous domain knowledge and promote common understanding among participants who might not use the same information model. In addition, semantic techniques can provide the reasoning services needed to deduce new information from existing knowledge and detect conflicts between policies. This feature is essential to simplify the definition and management of policies [16-18].

The aim of this paper is to specifically study the relationship between Semantic Web technologies and the RBAC model, and build a flexible access control system for web service. We follow the NIST standard RBAC model [5] and extend it with credential attributes. Then we use OWL-DL along with SWRL (Semantic Web Rule Language [20]) as policy languages to represent and implement the access control model. The contributions of this paper can be summarized as follows:

Access control model. By combining RBAC with ideas from ABAC, we actually introduce a new access control model which enables dynamic privileges assignment at two levels. On the one hand, attributes associated with services can determine the association of access privileges with roles. On the other hand, roles are dynamically assigned based on the credential attributes of users.

Semantic description. A high level OWL-DL ontology is developed to express the main elements (such as user credentials, roles, sessions, services as well as relations among them) of the access control model. The hierarchies and relations of users, roles and services can be thus represented in a natural manner. Through basic inference supported by OWL-DL, we can determine, for example, the subsumption relationship among classes or how instances are classified into their corresponding categories.

Semantic rules. Due to the inherent limitation of logic basis of OWL-DL, some operations of the access control model can't be well expressed and implemented, such as for enforcing dynamic roles assignment, static and dynamic separation of duty constraints, etc. To address this problem, SWRL which is a rule description language and also a candidate standard for the logic layer of Semantic Web is adopted to define a series of semantic rules to implement these operations. However, SWRL may present a new problem domain since it extends the expressiveness of OWL-DL beyond the decidable subset of OWL. Through using reasoner that can handle SWRL rules and restricting how the user could write these rules, a decidable result can be ensured.

Implementation. In terms of the proposed semantic access control model, the basic access control architecture for web service is built. Then a prototype system that follows the architecture is implemented to evaluate the proposal. Our code is written in Java. The RDF ontology data are processed using the Protégé[1]. The reasoner adopted is Jess rule engine[2].

[1] http://protege.stanford.edu/

[2] http://www.jessrules.com/jess/

The rest of this paper is organized as follows. The background about the NIST Standard RBAC model and the semantic web technologies are reviewed in section II. Section III is the main part in which the representation and reasoning on the access control model is elaborated. The basic access control architecture for web service is proposed in section IV. We provide a prototype implementation of the access control system in section V. An overview of other related works is given in section VI. The conclusions and future work are outlined in section VII.

II. BACKGROUND

A. NIST Standard RBAC Model

The NIST Standard RBAC model [5] includes sets of three basic elements called users, roles and permissions. A user is often defined as a human being, and may allow to be extended. A role is a job function within the context of an organization with associated semantics regarding the authority and responsibility conferred on the user assigned to the role. Permission is an approval to perform an action on one or more RBAC protected objects. User assignment and permission assignment are two many-to-many relations in the model, which connect roles with users and permissions respectively. In addition, the model includes a set of sessions where each session is a mapping of one user to an activated subset of roles that are assigned to the user.

The NIST RBAC model comprises three kinds of reference models: Core RBAC, Hierarchical RBAC and Constrained RBAC which can be concluded as follows.

Core RBAC. a minimum collection of RBAC elements, elements sets and relations, many-to-many user-role assignment and permission-role assignment, users get permissions through roles, roles activation as part of user's session.

Hierarchical RBAC. Core RBAC plus role hierarchies. A hierarchy is mathematically a partial order defining a seniority relation between roles.

Constrained RBAC. Hierarchical RBAC plus static and dynamic separation of duties constraints. Static separation of duty constraint adds exclusivity relations among roles with respect to user assignments. Dynamic separation of duty constraint defines exclusivity relations with respect to roles that are activated as part of a user's session.

B. Semantic Web Technologies

The Semantic Web notion was first articulated by Tim Berners-Lee as an extension to the existing web which refers to both a vision and a set of technologies [29]. The vision of the Semantic Web is the ability to express web information in a natural and formal language that can be interpreted by intelligent agents, thus permitting them on behalf of the human user to locate, share and integrate information in an automated way. Under the works of W3C, a set of languages, protocols and specifications have been developed to partially realize this vision.

The Semantic Web provides a framework for dynamic, distributed and extensible structured knowledge (so-

called ontology) founded on formal logic. Ontology is capable of describing concepts that exist in certain domain and relationships among them. Ontologies and their associated reasoners are the building blocks of the Semantic Web initiative. The current set of W3C standards are based on Resource Description Framework (RDF) and RDF Schema (RDF-S). RDF is a language that provides a basic capability of specifying graphs with a simple interpretation as a "semantic network" and serializing them in XML. RDF-S is based on class concept and inheritance, and already features simple constructs for describing ontologies. OWL is a W3C recommendation for describing ontologies on the Semantic Web, which has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S [15]. OWL supports the specification and use of ontologies that consist of terms representing classes, properties, individuals and axioms that assert constraints over them. Particularly, new information and conclusions can be drawn from ontologies by using so-called inference engines. Simple inferences are already possible with RDFS and OWL, for instance through inheritance. More complex custom inference rules require the usage of a special rule language. A promising approach is SWRL, whose rules are Horn-clause like rules written in terms of OWL concepts, properties and individuals. A SWRL rule is composed of a body (antecedent) part and a head (consequent) part, both of which consist of positive conjunctions of atoms.

III. Representation and Reasoning on the RBAC Model

In this section, we will describe how to define OWL-DL ontology and SWRL rules that can be used to represent and extend the NIST Standard RBAC model, and show how they can be used to specify and implement access control for web service. For each main concept in RBAC model including users, roles, permissions and sessions, there is an OWL class to represent them. For the relations and restrictions such as user-to-role assignment, permission-to-role assignment, role hierarchies and separation of duties, there are corresponding properties and rules to describe and reason about them.

A. Users

As mentioned above, in traditional RBAC system, roles assignment based on identities is static which may be restrictive in web service environment. To introduce flexibility into the procedure of roles assignment, we borrow ideas from ABAC [7] where permissions are granted to users based on a set of attribute values. However, in ABAC, it is difficult to collect and manage the various attributes, especially to verify their reliability [8]. Consequently, in this paper, we use credentials that can be directly verified and accepted across domains to represent the users, and perform user-to-role assignment based on attributes of the credential. Credentials in web application may employ different well-known authentication techniques such as Name Token, Binary Token, Key or SAML assertion. Our goal is to represent and extend the RBAC model using semantic web technologies. So, in order to extract the concepts, properties and relationships among these techniques at abstract level, a *Credential* class is constructed whose main classes and their relationships as well as some common properties are presented in Fig. 1. Concept and property assertions based on this class can be used for dynamic roles assignment through related semantic rules that will be introduced in section *E*.

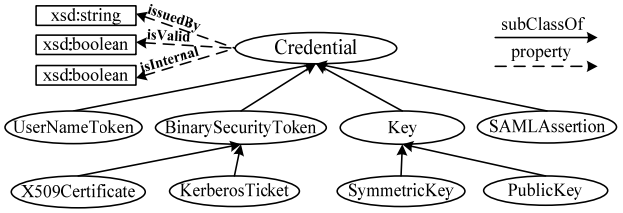

Fig. 1 Credential Class

As shown in Fig. 1, the top-level class *Credential* is subclassed to *UserNameToken*, *BinarySecurityToken*, *Key* and *SAMLAssertion*. All subclasses are pairwise disjointed.

The *UserNameToken* class denotes the users set whose identity is represented by name and password.

The *BinarySecurityToken* class represents the users set whose identity is confirmed by certain binary security token such as X.509 certificate or Kerberos ticket. So this class can have two disjoint subclasses *X509Certificate* and *KerberosTicket*.

The *Key* class represents the users set whose identity is represented by secret key such as Public key or Symmetric key. So *PublicKey* and *SymmetricKey* may be two disjoint subclasses of this class.

The *SAMLAssertion* class denotes the users set whose authentication information is described by the SAML assertion.

The roles assignment is based on attributes of the credential, so specific properties should be defined for the *credential* class. For example, the class defines an data type property *issuedBy* that has *xsd:string* as its range, which can be used to indicate who have issued the credential presented by the requestor. *isInternal* and *isValid* are also data type properties which both have *xsd:Boolean* as range. *isInternal* indicates whether the credential is issued by the internal or external authority. *isValid* indicates whether the credential is valid or not. In addition, each subclass in *credential* class can define its own properties based on its own characteristics and practical application requirements. For example, the *UserNameToken* class may have two data type properties *hasName* and *hasPassword*, both of type *xsd:string*. The *X509Certificate* class may have a data type property *serialNumber* of type *xsd:string*. These properties and the corresponding values can be taken as the preconditions for user-to-role assignment.

It is worthy to mention that the credential class here is not meant to be complete. As new authentication mechanisms become available, we can extend the existing classes and instances to incorporate the latest developments.

Formal expression of the credential class and other classes described below of the RBAC ontology by OWL-DL abstract syntax can be referred to the appendix A.

B. Roles

Role is the key concept in RBAC model. It adds an intermediary for assigning permissions to users which greatly simplifies authorization administration.

In our proposal, a generic *Role* class is defined, and each concrete role is as instance of this class. An object property *hasRole* is defined to link a user to his possible roles.

Roles hierarchy. Roles hierarchy defines an inheritance relation among roles and is commonly included as a key aspect of RBAC model. Inheritance is described in terms of permissions, that is, *r*1 "inherits" role *r*2 if all privileges of *r*2 are also privileges of *r*1. In order to represent the hierarchical relation among roles, a transitive object property *subRoleOf* is defined, which holds between two roles to state that one inherits all privileges of the other.

Constraints. Constraints are important aspects of RBAC and often regarded as one of the principal motivations behind RBAC. There are two kinds of constraints in the NIST Standard RBAC model, including static separation of duty (SSD) and dynamic separation of duty (DSD). SSD places constraints on user-role assignments over pairs of roles where any user can only have one of the pair as a possible role. DSD constraint holds between two roles where no user can have both simultaneously active. We define two object properties *ssd* and *dsd* to represent static separation of duty and dynamic separation of duty constraint respectively. Both of these two properties are defined to be symmetric and transitive.

C. Permissions

Permission is an approval to perform an action on one or more RBAC protected objects. The types of actions and objects depend on the practical application system.

In web service environment, we argue that both services and their operations should be protected, and access control should be enforced at service level and operation level. The action here is to invoke the services and operations. Consequently, for simplicity, we replace permissions just with services and operations. Two classes *Service* and *Operation* are defined to represent the services set and operations set respectively. The object property *hasOperation* is used to link a service to its operations. Some common properties such as *publishedBy* and *securityLevel* can be defined. The former indicates who are the publishers and the latter can be used to indicate the needed protection level of the service or operation. Assertions about these properties can be taken as the conditions to enforce permission assignment. *Service* may have two subclasses *InternalServie* and *ExternalService* which represent internal services set and external services set respectively. However, a service instance may simultaneously belong to these two classes, namely, they are not disjoint. When a user is permitted to call a service, he can invoke all operations of this service, conversely, when a user is permitted to invoke certain operation of a service, he may not be allowed to call other operations in this service. Additionally, according to the application requirements, new hierarchies of services and operations, as well as new properties can be constructed in *Service* and *Operation* class.

In order to establish relations between permissions and roles as well as users, some specific properties should be defined. Properties *assignedService* and *assignedOperation* associate a role with a service and an operation respectively, which means that a user with this role can invoke the corresponding service or operation. Properties *permittedService* and *permittedOperation* associate a user (represented by credential) with a service and an operation respectively, which means that a user has privilege to invoke the corresponding service or operation through certain role assigned to him. These two properties are usually used to view all the permitted services or operations of a user through rule reasonings. The situation of properties *activatedService* and *activatedOperation* is similar to that of *permittedService* and *permittedOperation*, but the privilege exists in session through activated roles.

D. Sessions

Session is a mapping of one user to possibly many roles. A user establishes a session during which the user activates a subset of roles that he is assigned. Each session is associated with a single user and each user is associated with one or more sessions.

We define a *Session* class and several special properties to capture the session concept mentioned above. Class *Session* represents the instance set of sessions. The object property *establish* is an inverse functional property to associate a user with a session established by this user. The object property *activatedRole* associates a session with a role activated in this session. When a user establishes a session, a new session instance assertion, along with other facts related to this session instance should be produced at run time. The processing of session instance is very different from the foregoing ones, because it can not be predefined and must be processed dynamically.

E. Rules

In above sections, a series of classes and properties are defined to fully describe the concepts and relations of RBAC. Combined with certain reasoner, some basic reasoning such as subsumption and satisfiability can be performed. However, due to the inherent limitation of logic basis of OWL-DL, some operations of the access control model can't be well implemented just based on these reasonings, such as for enforcing dynamic user-to-role assignment, static and dynamic separation of duty constraints, and for role activation and deactivation, etc. Consequently, technologies of logic layer above ontology layer in semantic web architecture should be adopted. That is, some specific rules should be defined and added into the RBAC ontology to implement the required operations. As a matter of fact, the description and

reasoning of logic rules is an important research aspect to satisfy the requirements of practical world in semantic web. We adopt SWRL, which is a rule description language based on a combination of the OWL-DL and Horn logic clause and also a candidate standard for logic layer, to define the required rules.

Five kinds of rules are defined at present and the rules instances are showed in table 1.

Table 1. Reasoning rules represented by SWRL

No.	Reasoning rules
R1.1	hasRole(?u,R1)←X509Certificate(?u) ∧ isInternal(?u,false) ∧ issuedBy(?u, "ca") ∧ isValid(?u, true)
R1.2	hasRole(?u,R2)←PublicKey(?u) ∧ isInternal(?u,true) ∧issuedBy(?u, "ka") ∧ isValid(?u, true)
R2.1	assignedService(R1, ?so)←Service(?so) ∧ publishedBy(?so, “sp”) ∧ securityLevel(?so, ?i) ∧ swrlb:lessThan(?i, 1)
R2.2	assignedService(R2, ?so)←Service(?so) ∧ securityLevel(?so, ?i) ∧ swrlb:greaterThan(?i, 3)
R3.1	¬HasRole(?u, ?r2)←hasRole(?u, ?r1) ∧ ssd(?r1, ?r2)
R3.2	¬ActivatedRole(?s, ?r2)←activatedRole(?s, ?r1) ∧ dsd(?r1, ?r2)
R4.1	hasRole(?u, ?r2)←hasRole(?u, ?r1) ∧ subRoleOf(?r1, ?r2)
R4.2	assignedService(?r2, ?so)←assignedService(?r1, ?so) ∧ subRoleOf(?r2, ?r1)
R4.3	assignedOperation(?r2, ?oo)←assignedOperation(?r1, ?oo) ∧ subRoleOf(?r2, ?r1)
R4.4	activatedRole(?s, ?r2)←activatedRole(?s, ?r1) ∧ subRoleOf(?r1, ?r2)
R5.1	permittedService(?u, ?so)←hasRole(?u, ?r) ∧ assignedService(?r, ?so)
R5.2	activatedService(?u, ?so)←establish(?u, ?s) ∧ activatedRole(?s, ?r) ∧ assignedService(?r, ?so)

User-to-role assignment rules. For user-to-role assignment, we distinguish between registered and unregistered users. Because that, in web service environment, although most of users may come from external domain and can’t be known as a prior, there are also some known users for a service provider such as business partners. Consequently, for registered users, roles assignment can be predefined just by adding related assertions of *hasRole* into the RBAC ontology base. For example, suppose that there are assertions *UserNameToken*(*u*), *Role*(*r*) and *hasRole*(*u, r*) exist in the initial ontology base, these assertions indicate that *u* is a known user of type *UserNameToken* and *r* is one role assigned to him. On the other hand, for unregistered users, corresponding semantic rules must be constructed to dynamically perform roles assignment. For example, rule 1.1 in table 1 specifies that a user with X.509 certificate issued by the external certificate authority ‘*ca*’ can be assigned with a role *R1*. Another one specifies that a user with public key issued by the internal authority ‘*ka*’ can own a role *R2*, which can be described by rule 1.2. Other roles assignment rules can be defined similarly in terms of specific application.

Permission-to-role assignment rules. Just as mentioned in section *C*, the related property assertions of *Service* and *Operation* class can be used as the conditions to enforce permission assignment. For instance, rule 2.1 in table 1 shows that the services published by service provider ‘*sp*’ and with security level less than ‘1’ can be assigned to the role *R1*. Rule 2.2 indicates that the services with security level greater than ‘3’ can be only assigned to the role *R2*. Other permission assign rules can be defined similarly.

Separation of duty constraints rules. We have defined two properties *ssd* and *dsd* to represent static and dynamic separation of duty constraint respectively. For static separation of duty, it means that two roles can’t be assigned to a user simultaneously if the *ssd* relation assertion exists between them. Rule 3.1 in table 1 is used to enforce SSD constraint. For dynamic separation of duty, it means that two roles can’t be activated simultaneously in a session if the *dsd* relation assertion exists between them. Rule 3.2 is used to enforce DSD constraint. For convenience, we use notation ‘ ¬ ’ to express negation, which doesn’t exist in SWRL. Consequently, extra properties such as *cannotHasRole* should be defined to enable these rules.

Role hierarchies reasoning rules. Rules 4.1-4.4 express the role hierarchies reasoning. Rule 4.1 indicates that, for any user *u* and any role *r1* and *r2*, if *u* has been assigned with role *r1* and *r1* has *subRoleOf* relation with *r2*, then *r2* is an implied role of this user. Rule 4.2 indicates that, for any service object *so* and any role *r1* and *r2*, if *r1* has privilege to access the service *so* and *r2* has *subRoleOf* relation with *r1*, then *r2* also has privilege to access this service. The meaning of rules 4.3 and 4.4 can be deduced similarly.

Association rules. In fact, through enforcing rules defined above, basic facts can be gained such as what roles a user can have or activate and what permissions are assigned to these roles. However, in order to directly view all the permissions of a user, specific rules to associate users with permissions should be defined such as rules 5.1 and 5.2 in table 1. Rule 5.1 indicates that a user can acquire all the services he is permitted to invoke, through the roles as well as the implied roles he owns. Rule 5.2 indicates that a user can acquire all the services he is currently permitted to invoke in his sessions. When a user wants to view all the operations he has privileges to invoke, similar rules to rules 5.1 and 5.2 can be defined.

However, using SWRL may also present a new problem domain since it extends the expressiveness of OWL-DL beyond the decidable subset of OWL. There are two ways in which this can be overcome. Either restrict the expressiveness of SWRL and use an existing

reasoning engine such as Pellet or Racer, or use a reasoner such as Jess which can handle SWRL rules [21]. Through using the Jess rule engine and restricting how the user can write the rules, we can ensure a decidable result. In addition, following the open world assumption (OWA), the knowledge base may be incomplete. If a fact is not contained in or does not follow from the knowledge base, we cannot conclude that the complement is true [17]. In our case, if we check for an authorization that does not exist, the reasoner answers that it is unknown. Because the binary authorization decision is required, we will interpret "unknown" as a negative authorization, thus the user does not get the authorization.

IV. Proposed Architecture

In terms of the representation and reasoning on the extended RBAC model presented in previous section, the basic access control architecture for web service is proposed. The main blocks of this architecture and the working process are depicted in Fig. 2.

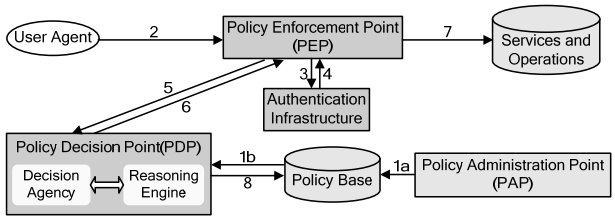

Fig. 2 Access control architecture for web service

Policy Base. It is actually a knowledge base that can be formalized as $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{R})$. Here, we call $\mathcal{K}$ the access control knowledge base (ACKB), and $\mathcal{T}$, $\mathcal{A}$, $\mathcal{R}$ correspond to the TBox, ABox and RBox of ACKB respectively. $\mathcal{T}$ comprises the axioms set about all the common classes and properties defined in subsection *A-D* of section III. $\mathcal{A}$ is the assertions set that is stated based on the TBox, and it has relationship with the practical application. Table 2 in section V is an example of the ABox. $\mathcal{R}$ denotes the rules set that is defined in subsection *E* of section III. The Policy Administration Point (PAP) is responsible for creating, maintaining and updating the ACKB.

Policy Decision Point (PDP). The PDP is the kernel component of the access control architecture. We logically divided the PDP into two parts: Decision Agency and Reasoning Engine. The main responsibilities of the Decision Agency include: loading ACKB into the Reasoning Engine when the system is initiated; receiving the request from PEP and extracting the request parameters; reproducing the request parameters in RDF and hands them over to the Reasoning Engine to trigger the decision reasoning; generating session assertions and related facts dynamically during the reasoning process; making access decision when the reasoning is completed and returning the decision result to the PEP; updates the ACKB if necessary. For the Reasoning Engine, it is responsible for classification and consistency verification when the ACKB is loaded, and executing reasoning based on the query condition from Decision Agency.

Policy Enforcement Point (PEP). The PEP is used to intercept access requests and enforce access control. When an access request is received, PEP extracts the credential provided by the user and forwards it to the Authentication Infrastructure. If the credential is accepted and verified to be valid, PEP will construct a formatted request that includes related attribute values of the credential as well as the requested object, and send it to the PDP. If not, PEP will deny the access request and notify the user for the required credential.

Authentication Infrastructure. In our architecture, Authentication Infrastructure is mainly used to verify the credential and extract the needed attribute values for PEP. Service providers may have different Authentication Infrastructure which however can be classified into two main types: central ones or federated ones. The concrete implementation of Authentication Infrastructure exceeds the range of this paper, and more details about it can be referred to [8, 23]. Service provider is assumed to have established a required Authentication Infrastructure in our prototype implementation.

The access control decision and enforcement process is thus performed according to the following steps:

1. The ACKB is predefined by the PAP (1a). When the system is initiated, Decision Agency loads the ACKB into the Reasoning Engine for classification and consistency verification (1b).
2. The user agent sends a service request to the PEP. Besides stating the access objects (what services or operations he wants to invoke), at least one type of credential should be contained in the request.
3. The PEP extracts the credential from this request and forwards it to the Authentication Infrastructure.
4. The Authentication Infrastructure returns the verification result, which may contain the needed attribute values, to the PEP.
5. In case the credential is proved to be valid, PEP will encapsulate the related attribute values of the credential and the requesting object in a formatted request and send it to the PDP.
6. According to the request parameters, the PDP makes access decision through proper reasoning, and returns the decision result to the PEP.
7. If access is granted, the PEP allows the user to access the service. Otherwise, access is refused.
8. After the decision reasoning is completed, the Decision Agency can choose to update the ACKB or not, which depends on the update policy of the system. Generally, assertions related to the strange user need not be updated into the knowledge base and should be removed immediately.

V. Implementation

In order to evaluate the proposal, we have implemented a prototype system that follows the architecture presented above with minor simplifications. We assume that the organization has established a secure

infrastructure to verify, request and propagate the user credentials. Also, the actual execution of service access (namely the PEP) is omitted in the test scenario. Our code is written in Java. Protégé and Jess rule engine are adopted as the ontology processing tool and reasoning system respectively. The interaction between the knowledge base containing SWRL rules and the rule engine is implemented by the bridge mechanism provided in Protégé-OWL. We will give an application scenario as follows to demonstrate the reasoning process and verify the enforcement effect of the system.

Scenario description: Suppose, for example, that for an organization there are four kinds of roles: *R1, R2, R3, R4*, and it provides five services: *query, purchase, exchange, refund, approve*. The hierarchies and relations of the roles are as follows. *R4* is senior to *R2* and *R3* which are both senior to *R1*. Role *R2* has *ssd* and *dsd* relationship with *R3* and *R1* respectively. We assume that services *query* and *purchase* are assigned to role *R1*, services *exchange* and *refund* are associated with the roles *R2* and *R3* respectively, and role *R4* has privilege to access the service *approve*. So initial assertions showed in table 2 should be added into the ABox of ACKB. Now, we assume that a user *u1* with a valid public key issued by the internal authority '*ka*' wants to access the service *purchase*. Intuitionally, according to the reasoning rules in table 1 and permission assignment described above, we can infer that user *u1* can be assigned to role *R2*, and *R2* inherits all privileges of *R1* which is permitted to access service *purchase*. Consequently, the system should make the authorization decision that *u1* can access the requested service *purchase*.

Experiment environment: PC (Core2 CPU 2.33GHz, 2GB Memory); Operating system: Windows XP Pro. SP2; Ontology processing tool: Protégé 3.4; Reasoning engine: Jess 7.0p2; Integrated developing environment: Eclipse 3.6, etc.

Table 2. Initial assertions of the Policy Base

Role(R1)	subRoleOf(R2, R1)	Service(query)	assignedService(R1, query)
Role(R2)	subRoleOf(R3, R1)	Service(purchase)	assignedService(R1, purchase)
Role(R3)	subRoleOf(R4, R2)	Service(exchange)	assignedService(R2, exchange)
Role(R4)	subRoleOf(R4, R3)	Service(refund)	assignedService(R3, refund)
ssd(R2,R3)	dsd(R2, R1)	Service(approve)	assignedService(R4, approve)

Experiment process analysis: Firstly, the PDP will load and map the policy knowledge base ACKB into the reasoning engine, and initial inferences like consistency check will be performed by the engine. Because the verification for credential is omitted in the test scenario, the corresponding request parameters of *u1* are directly included in a formatted request and sent to the API of PDP. Then, related assertions such as *PublicKey(u1), isInternal(u1,true), issuedBy(u1,"ka"), isValid(u1,true)* are constructed by the Decision Agency and mapped to the Jess engine to trigger the decision reasoning. After that, a SQWRL query as follows can be constructed to view what roles the user *u1* can have and what services he can access with theses roles.

Query 1: hasRole(u1,?r) ∧ assignedService(?r,?so) → sqwrl:select(?r, ?so)

The query results returned by the reasoning engine are as follows: (*R1, query*), (*R1, purchase*), (*R2, exchange*), (*R2, query*), (*R2, purchase*). From the results, we can know that *u1* can be assigned to roles *R1* and *R2* which both have privilege to access the service *purchase*. Thus Decision Agency can make the decision that *u1* is permitted to access the requested service. According to the discussion in scenario description, the decision result is correct. Then, Decision Agency will establish a session for *u1* and choose *R1* to be activated in terms of the least of privilege rule. So session assertions such as *session(s1), establish(u1,s1)* and *activatedRole(s1, R1)* will be produced and mapped to carry on the reasoning.

It is worthy to mention that similar queries like *Query 1* can be constructed to examine other reasoning results. For example, we can construct a query as follows to view all the privileges of *u1* in his current session.

Query 2: activatedService(u1,?so)→ sqwrl:select(u1,?so)

The query results are *(u1,query)* and *(u1,purchase)*, which means that *u1* also has privilege to access the service *query* except *purchase* in current session.

In fact, many inferred axioms are generated by the reasoning engine that have not been defined initially. These inferred axioms are very important to make the access decision and examine the reasoning process. Table 3 shows all the inferred axioms generated in the test scenario.

Table 3. Inferred axioms in the test scenario

No.	Inferred Axioms	No.	Inferred Axioms	No.	Inferred Axioms
IA1	hasRole(u1, R1)	IA7	assignedService(R3, query)	IA13	permittedService(u1,purchase)
IA2	hasRole(u1, R2)	IA8	assignedService(R3,purchase)	IA14	permittedService(u1,query)
IA3	notHasRole(u1, R3)	IA9	assignedService(R4, query)	IA15	permittedService(u1,exchange)
IA4	notActivatedRole(s1,R2)	IA10	assignedService(R4,purchase)	IA16	activatedService(u1,query)
IA5	assignedService(R2, query)	IA11	assignedService(R4, refund)	IA17	activatedService(u1,purchase)
IA6	assignedService(R2,purchase)	IA12	assignedService(R4,exchange)		

In table 3, the inferred axioms IA1-IA3 indicate that *u1* can be assigned to roles *R1* and *R2* but not *R3* due to

the SSD constraint. IA4 shows that *R2* can't be activated in current session *s1* because of the DSD constraint. We can get all the implicit privilege relations between roles and services through the inferred axioms IA5-IA12, and all the possible privileges of *u1* through the inferred axioms IA13-IA15. IA16 and IA17 correspond to the results of *Query 2* above.

Performance analysis: The consumption of resource and time for the experiment is shown in Table 4. Because the program code and Jess engine both run in JRE (Java Runtime Environment), the columns "CPU" and "Memory" respectively represent the CPU and memory consumption of the JRE when the corresponding operations is executed. The column "Time" represents the time consumption to execute initialization and query reasoning on the knowledge base. The statistical results in table 4 are average values by 10 experiments. From table 4, we can know that the resource consumed to run the prototype system is little. The primary time consumption consists in the initialization of knowledge base (loading the knowledge base, executing classification and consistency check, etc). However, initialization needs only once and can facilitate the subsequent queries. Consequently, the time delay of query (from sending the request to receiving the result) can be at the level of 10 ms, which shows the presented method is feasible in practice.

Table 4 Resource and time consumed for prototype system

	CPU	Memory	Time
Idle	0%	≈2.8M	——
Initialization	15.6%	≈27.5M	0.094 s
Query	6.9%	≈21.2M	0.013 s

From the experiment described above, we can draw some conclusions as follows. i) Correct decision result has been gotten which can verify the validity of the proposed approach. ii) The anticipated functions such as dynamic roles assignment, SSD constraint and DSD constraint have been implemented, which are not considered or implemented in some other similar works. iii) Many inferred axioms can be gotten that needn't be defined as a prior, and this feature can simplify the definition and administration of the policies far and away. iv) Useful information can be directly gotten through simple SQWRL query, and it will facilitate the interaction with the system. v) Performance analysis for prototype system shows the feasibility of the proposal.

VI. Related Work

We mainly talk about the related works that employ Semantic Web technologies in the access control policy representation and enforcement.

Technically, the works of [6], [16] and [26] are most approximate to our proposal. Lorenzo et al. [6] has developed an OWL-DL ontology to express the elements of a RBAC system. They state that roles are dynamically associated with users based on the contextual attributes. However, on the one hand, it has been approved that collecting and managing various context attributes is difficult, especially verifying their reliability. On the other hand, they also do not explain how to model these context attributes and combine them with the OWL-DL ontology. Compared with their work, we use credentials that can be directly verified and accepted across domains to represent the users, and perform roles assignment dynamically based on attributes of the credential. More importantly, we have constructed a *Credential* class to model various credentials and represent the user concept that can be integrated with the RBAC ontology naturally. In addition, they formulate the SSD constraint by the construct *owl: disjointWith*, but define the object property $notTogetherWith \subset Role \times Role$ to express the DSD constraint. While they don't use any rules and provide a running example, it is unclear how to enforce the constraints in practice. In our approach, SSD and DSD constraints are both expressed by the object properties, and specific rules are defined to enforce them. Finin et al. [16] propose two approaches to model RBAC: one where roles are represented as classes and another one where roles are as instances. Our approach shares some similarities with the latter. However, their approach is confined within the expressiveness of OWL, thus some operations like enforcing role activation and deactivation can't be well implemented. Contrasted with this, we are not limited to the expressiveness of OWL by adding another semantic web language SWRL, and a series of SWRL rules are constructed to implement the required operations effectively. Furthermore, role assignments are not modeled in their proposal and there is also no associated system, whereas these issues are considered and realized elaborately in this paper. Wu et al. [26] focus on specifying the static description of RBAC constraints by OWL. They do not provide a systematic modeling approach that can be used to create applications with these constraints and validate them. Additionally, it is not appropriate in their description that using the same property *conflictRole* to express SSD and DSD constraints, because the implications of these two constraints are different.

As the logic foundation of OWL, Description Logic (DL) is also adopted by some works to formalize the access control model. Zhao et al. [27] choose the DL language $\mathcal{ALCQ}$ to represent and reason on RBAC as well as the constraints like separation of duty and role cardinality. Chae et al. [28] extend the hierarchical RBAC by a class hierarchy of the accessed objects and implement this model in DL. However, Knechtel et al. [17] state that the proposal in [28] has several flaws, such as the object class hierarchy is inverted which leads to unintended inferences, the DL semantics is not respected and its running example provides wrong results, etc. In order to fix these flaws, they adopt the DL $\mathcal{SROIQ}(\mathcal{D})$ that can simulate the *concept product* to redescribe the model. They follow the same example scenario and give the correct results. However, while the $\mathcal{SROIQ}(\mathcal{D})$ is very expressive, its computational complexity increases. More importantly, all above works do not consider their application in open environment where users may be not

known in advance, so the statement like in [17] that the reasoning needs to be run only once is impractical.

Some other works concern the semantic extension for XACML [24, 25, 8]. Damiani et al. [24] extend the XACML by adding the capability to designate subjects and objects via generic RDF statements. Kolovski et al. [25] present a DL approach to analyze XACML polices. They focus not on runtime but design time to audit a policy to find mistakes and inconsistencies, since the services used are computationally expensive. In order to address the problem of heterogeneous attributes and simplify the specification of ABAC policies, Priebe et al. [8] extend the XACML architecture with an ontology-based inference facility for attributes management and mapping. Because this approach further complicates the access control process of XACML, its performance needs to be evaluated.

VII. Conclusions

In this paper, a semantic-based access control approach is proposed, to create a feasible and flexible access control solution for Web service.

We follow the Role-based Access control model and extend it with credential attributes. The available semantic web technologies OWL-DL and SWRL are used to represent the access control model. In our proposal, the users are modeled as a *Credential* class that can be combined with the access control ontology in nature. We construct *Service* class and *Operation* class respectively, and make access control enforcing at service level and operation level. A series of SWRL rules are defined to implement the required functions effectively, such as dynamic roles assignment and permissions assignment, roles hierarchy reasoning, SSD and DSD constraints, etc. The combination of OWL-DL and SWRL will utilize the semantic web technologies to the most extent. The basic access control architecture for web service based on the semantic proposal is presented. We have implemented the system in Java with minor simplifications.

In future work, we will investigate the extension of the proposal in this paper to the web services composition situation. Namely, how to express and integrate the access control polices in different security domains using semantic web techniques.

Acknowledgment

This work is supported by the Natural Science Foundation of Jiangsu Province of China under Grant No. BK2010132.

References

[1] W3C, “Simple Object Access Protocol (SOAP) 1.2”, http://www.w3.org/TR/soap12/, April 2007.
[2] A. Singhal, T. Winograd, K. Scarfone, “Guide to Secure Web Service”, NIST Special Publication 800-95, 2007.
[3] M. Coetzee, J. Eloff, “Towards Web Service Access Control”, Computers & Security, 2004(23), pp.559-570.
[4] M. Bartoletti, P. Degano, G. Ferrari, R. Zunino, “Semantics-Based Design for Secure Web Services”, IEEE Transactions on Software Engineering, 2008, 34(1), pp.33-49.
[5] F. David, S. Ravi, G. Serban, “Proposed NIST Standard for Role-Based Access Control”, ACM Transactions on Information and System Security, 2001, 4(3), pp. 224-274.
[6] C. Lorenzo, F. C. Isabel, T. Roberto, “A Role and Attribute Based Access Control System Using Semantic Web Technologies”, Lecture Notes in Computer Science, vol.4806, 2007, pp.1256-1266.
[7] Y. Eric, T. Jin, “Attributed based access control for Web services”, In Proceedings of the IEEE International Conference on Web Services, 2005, pp.561-569.
[8] T. Priebe, W. Dobmeier, N. Kamprath, “Supporting Attribute-based Access Control in Authorization and Authentication Infrastructures with Ontologies”, Journal of Software, 2007, 2(1), pp.27-38.
[9] R. Bhatti, E. Bertino, A. Ghafoor, J. Joshi, “XML-based Specification for Web Services Document Security”, IEEE Computer, 2004, 37(4), pp.41-49.
[10] G. Tonti, J. M. Bradshaw, R. Jeffers, et al, “Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder”, Lecture Notes in Computer Science, Vol.2870, 2003, pp.419-437.
[11] OASIS, “eXtensible Access Control Markup Language (XACML) V2.0”, http://docs.oasis-open.org/xacml/2.0, 2005.
[12] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, “The ponder policy specification language”, Lecture Notes in Computer Science, vol.1995, 2001, pp.18-38.
[13] L. Kagal, T. Finin, A. Joshi, “A Policy Language for a Pervasive Computing Environment”, In Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks, 2003, pp.63-74.
[14] A. Uszok, J. M. Bradshaw, M. Johnson, et al, “KAoS policy management for semantic web services”, IEEE Intelligent Systems, 2004, 19(4), pp.32-41.
[15] W3C, “OWL Web Ontology Language Reference”, http://www.w3.org/TR/2004/REC-owl-ref-20040210/, 2004.
[16] T. Finin, A. Joshi, L. Kagal, et al., “ROWLBAC: Representing Role Based Access Control in OWL”, In Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, Colorado, USA, 2008, pp.73-82.
[17] M. Knechtel, J. Hladik, “RBAC Authorization Decision with DL Reasoning”, In Proceedings of the IADIS International Conference WWW/Internet, 2008, pp.169-176.
[18] A. Toninelli, R. Montanari, L. Kagal, O. Lassila, “A semantic context-aware access control framework for secure collaborations in pervasive computing environments”, In Proceedings of the 5th International Semantic Web Conference, 2006, pp.473-486.
[19] M. Wu, J. X. Chen, Y. S. Ding, “Role-Based Access Control for Web Services”, WSEAS Transactions on Information Science and Applications, 2006, 3(8), pp.1553-1558.
[20] W3C, “SWRL: A Semantic Web Rule Language Combining OWL and RuleML”, http://www.w3.org/Submission/SWRL/, 2004.
[21] B. Shields, O. Molloy, G. Lyons, J. Duggan, “Using semantic rules to determine access control for web services”, In Proceedings of the 15th International Conference on World Wide Web, Edinburgh, Scotland, 2006, pp.913-914.

[22] W3C, "SPARQL Query Language for RDF", http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/, 2005.
[23] J. Lopez, R. Oppliger, G. Pernul, "Authentication and Authorization Infrastructures (AAIs): A Comparative Survey", Computers & Security, 2004, vol.23, pp.578-590.
[24] E. Damiani, S. De Capitani di Vimercati, C. Fugazza, et al., "Extending Policy Languages to the Semantic Web", In Proceedings of the 4th International Conference on Web Engineering, Munich, Germany, 2004, pp.330-343.
[25] V. Kolovski, J. Hendler, B. Parsia, "Analyzing Web Access Control Policies", In Proceedings of the 16th International Conference on World Wide Web, Banff, Alberta, Canada, 2007, pp.677-686.
[26] D. Wu, J. Lin, "Using Semantic Web Technologies to Specify Constraints of RBAC", In Proceedings of the 6th International Conference on Parallel and Distributed Computing, Applications and Technologies, 2005, pp.543-545.
[27] C. Zhao, N. Heilili, S. Liu, "Representation and Reasoning on RBAC: A Description Logic Approach", Lecture Notes in Computer Science, vol.3722, 2005, pp. 381-393.
[28] J. H. Chae and N. Shiri, "Formalization of RBAC policy with object class hierarchy", Lecture Notes in Computer Science, vol.4464, 2007, pp.162-176.
[29] T. Berners-Lee, J. Hendler, O. Lassila, "The Semantic Web", Scientific American, 2001, 284(5), pp.34-43.

Zhengqiu He is a PhD candidate at the Institute of Command Automation in the University of Science and Technology of PLA. His research interests include network security, web service and the Semantic Web. He received MS degrees in computer science from the Uni. of Sci. & Tech in 2007. Contact him at Institute of Command Automation, Uni. of Sci. & Tech., Haifu Road 1, Nanjing China, 210007; Email: hzqzyl@163.com.

Lifa Wu is a professor at the Institute of Command Automation. He received PhD degrees in computer science from the Nanjing University. His research interests include software technology, network security and protocol validation, etc. His email address is wulifa@vip.163.com.

APPENDIX A FORMAL DESCRIPTION OF THE TBOX OF ACKB BY OWL-DL ABSTRACT SYNTAX

```
Class(rbac:Credential partial owl:Thing)
Class(rbac:UserNameToken partial rbac:Credential)
Class(rbac:BinarySecurityToken partial rbac:Credential)
Class(rbac:Key partial rbac:Credential)
Class(rbac:SAMLAssertion partial rbac:Credential)
Class(rbac:X509Certificate partial rbac:BinarySecurityToken)
Class(rbac:Kerboros partial rbac:BinarySecurityToken)
Class(rbac:SymmetricKey partial rbac:Key)
Class(rbac:PublicKey partial rbac:Key)
DatatypeProperty(rbac:issuedBy domain(rbac:Credential) range(xsd:string))
DatatypeProperty(rbac:isInternal domain(rbac:Credential) range(xsd:boolean))
DatatypeProperty(rbac:isValid domain(rbac:Credential) range(xsd: boolean))

Class(rbac:Role partial owl:Thing)
ObjectProperty(rbac:hasRole domain(rbac:Credential) range(rbac:Role))
ObjectProperty(rbac:subRoleOf domain(rbac:Role) range(rbac:Role) TransitiveProperty)
ObjectProperty(rbac:ssd domain(rbac:Role) range(rbac:Role) SymmetricProperty TransitiveProperty)
ObjectProperty(rbac:dsd domain(rbac:Role) range(rbac:Role) SymmetricProperty TransitiveProperty)

Class(rbac:Servcie partial owl:Thing)
Class(rbac:Operation partial rbac:Thing)
Class(rbac:ExternalService partial rbac:Service)
Class(rbac:InternalService partial rbac:Service)
DatatypeProperty(rbac: publishedBy range(xsd:string))
DatatypeProperty(rbac: securityLevel range(xsd:integer))
ObjectProperty(rbac:hasOperation domain(rbac:Service) range(rbac:Operation))
ObjectProperty(rbac:assignedService domain(rbac:Role) range(rbac:Service))
ObjectProperty(rbac:assignedOperation domain(rbac:Role) range(rbac:Operation))
ObjectProperty(rbac:permittedService domain(rbac:Credential) range(rbac:Service))
ObjectProperty(rbac:permittedOperation domain(rbac:Credential) range(rbac:Operation))
ObjectProperty(rbac:activatedService domain(rbac:Credential) range(rbac:Service))
ObjectProperty(rbac:activatedOperation domain(rbac:Credential) range(rbac:Operation))

Class(rbac:Session partial owl:Thing)
ObjectProperty(rbac:activatedRole domain(rbac:Session) range(rbac:Role)
ObjectProperty(rbac:establish domain(rbac:Credential) range(rbac:Session) InverseFunctional)
```