International Journal of Computer Applications (0975 – 8887)
Volume 42– No.6, March 2012

# An Access Control Model for Web Services with Dynamic Separation of Duty Rules

HadisehSeyyedAlipour
Qazvin Islamic Azad University
Qazvin, Iran

Mehdi Sabbari
Qazvin Islamic Azad University
Qazvin, Iran

EslamNazemi
ShahidBeheshti University
Tehran, Iran

## ABSTRACT

One of the most significant difficulties with developing Service-Oriented Architecture (SOA) involves meeting its security challenges. Access control is an important security mechanism for organizations to protect their resources in collaborative environments and processes. In these processes, shared resources are often used and there are complex relationships between activities and users, so the definition and administration of different security levels (tasks, users, resources, etc.) is necessary. Different access control models and mechanisms have been proposed in recent years. However, under the new collaborative paradigm based on Web services and workflow technologies, some specific access control requirements should be addressed to support the various processes. In this paper, an access control model is proposed that considers the necessary elements to represent authentication, authorization and access control aspects in SOA environment. One of the underlined issues in this model is Separation of Duty (SoD) policy, which is widely considered to be a fundamental security principle for prevention of fraud and errors in information security.

## General Terms

Software, Service Oriented Architecture,Distributed System, Security, Access Control Model, Web Application.

## Keywords

Web Service, Separation of Duty, Access Control,Rule Definition, Authentication, Authorization.

## 1. INTRODUCTION

Security is an important issue that must be well-defined in SOA environment so that it could be used in implementing the web services. In 1975, Saltzer and Schroeder identified eight principles of design to enhance security within computer systems [1]. One of these principles is Separation of Duty (SoD). The essential aim of SoD is to recognize potential conflicts of interest in activities which could result in errors or fraud. Where potential conflicts of interest arise, activities are structured such that different individuals perform specific steps of the overall activity. So it is used extensively to prevent conflict of interest, fraud and error control in organizations.

Access control security is one of the important aspects in SOA that is considered as a challenge. It has been found to be one of the effective ways of insuring that only authorized users have access to the information resources to perform their job function. This issue requires further attention and review because of the architecture’s distributed nature, its high re-usability, simple accessibility and the autonomy of logical solutions units.

Access management includes the access control and other related abilities such as identities management. Access management ensures that the resources are accessible only for users, programs, processes and/or permitted systems referring to access rules which are defined by policies and properties. One of the important aspects of SOA is that it has different users and sub-systems with the relationships and co-operations between them in doing the activities. In this architecture, recourses and services usually are shared by different users. The management of these entities, resources, access control levels, uncover vulnerabilities and identify potential threats and risks are posed as a challenge. This feature also can be seen in implementing this architecture, so the aspect of implementation should also be noted.

A number of industrystandards have been or are being developed to facilitatethe web service level security tasks such asidentification, authentication, and authorization.Among these standards:

A Language for Declaring Security: SAML (Security Assertion Markup Language), created by the Security Services Technical Committee of the Organization for the Advancement of Structured Information Standards (OASIS), is an XML-based framework for communicating user authentication, entitlement, and attribute information. As its name suggests, SAML allows business entities to make assertions regarding the identity, attributes, and entitlements of a subject (an entity that is often a human user) to other entities, such as a partner company or another enterprise application [2].

An Access Controlling Language:XACML (eXtensible Access Control Markup Language) is an OASIS standard that describes both a policy language implemented in XML and an access control decision request/response language implemented in XML [3]. The policy language details general access control requirements, and has standard extension points for defining new functions, data types, combining logic, etc. The request/response language lets you form a query to ask whether or not a given action should be allowed, and interpret the result. The response always includes an answer about whether the request should be allowed using one of four values: Permit, Deny, Indeterminate (an error occurred or some required value was missing, so a decision cannot be made) or Not Applicable (the request can't be answered by this service)[4].

The remainder of this paper is organized as follows: section 2 provides a briefoverview of existing access control models, Section 3 describes separation of duties, section 4 compares granting permission by roles vs. rules, section 5 explains the proposed access control model,section 6 illustrates sequence diagram of thismodel, in section 7 there is a case study from rule definition in the proposed model and finally section 8 concludes the paper.

## 2. EXISTING ACCESS CONTROL MODELS

Access control models are abstractions that incorporate the rules and parameters required to execute access control policies. In this section we briefly explain the conventional access control models which are IBAC, RBAC and ABAC.

### 2.1 Identity Based Access Control

Under this model, permissions to access a resource is directly associated with a subject's identifier (e.g., a user name). Access to the resource is only granted when such an association exists. An example of IBAC is the use of Access Control Lists (ACL), commonly found in operation systems and network security services [5]. The concept of an ACL is very simple: each resource on a system to which access should be controlled, referred to as an object, has its own associated list of mappings between the set of entities requesting access to the resource and the set of actions that each entity can take on the resource.

Drawbacks of IBAC are: the number of identifiers in the ACL will increase and become difficult to maintain as more users request access, making this approach impossible to scale. The ACL for a particular file, process, or other resource must be checked every time the resource is accessed, and this can be an inefficient means of providing access control. Access control decisions are not based on any business function or characteristics of the user but solely on the identifiers, making it unsuitable for enterprise level use.

### 2.2 Role Based Access Control

The RBAC model restricts access to a resource based on the business function or role the subject is performing. The permissions to access a resource are then assigned to the appropriate role(s), rather than directly assigned to subject identifiers [6]. When a user changes jobs, some other user is allowed to take on that role. No ACL changes are needed. Of course, sometimes only a few of the user's rights change. In that case, a new role needs to be introduced. Often the rights associated with a role depend on which user is acting in that role. In that case, too, a new role needs to be introduced [7]. The RBAC reference model is defined in terms of four model components: Core RBAC, Hierarchical RBAC, Static Separation of Duty Relations, and Dynamic Separation of Duty Relations [8].

Some RBAC limitations in SOA are: In RBAC you still have to manage every user account and bind these accounts to roles. Unknown accounts can only be linked to a default role, like guest or customer. Because the core RBAC model limits the abstraction of user function to roles only, it does not consider any other characteristics that a user may demonstrate. Further, RBAC generally doesn't take into account the characteristics of resources (other than their identifiers); nor does it capture any security relevant information of the environment.

With RBAC, as shown in [9], the SoD violation occurs when permissions contained in two different roles are in conflict.The role concept may thus expand the effects of SoD resolution into innocent parts of the user population and also expands the problem across unaffected permissions.With SoD enforced on these roles, none of the permissions they bundle can be combined even if most combinations do not represent a business risk.By expanding the SoD conflict in either or both dimensions, affecting innocent users and/or innocent permissions, RBAC generates considerable administrative costs.RBAC has forced the implementer to execute some quite intensive business analysis tasks, which may include: The need for (sometimes massive amounts of) role mining to understand what business functions currently exist and how these map to a set of logical roles.Mapping logical roles to user entitlements within target business applications, in very large deployments, the need to create role hierarchies to rationalize the overall number of roles.

### 2.3 Attribute Based Access Control

Policy-Based Access Control (PBAC), which is called Attribute-Based Access Control (ABAC) in the US Defense Department jargon, extends RBAC to a more general set of properties [10]. Unlike IBAC and RBAC, the ABAC model [5] can define permissions based on just about any security relevant characteristics, known as attributes. For access control purposes, we are concerned with three types of attributes:

- Subject Attributes (S). Associated with a subject that defines the identity and characteristics of that subject.
- Resource Attributes (R). Associated with a resource, such as a Web service, system function, or data
- Environment Attributes (E). Describes the operational, technical, or situational environment or context in which the information access occurs.

In the most general form, a Policy Rule that decides on whether a subject s can access a resource r in a particular environment e, is a Boolean function of s, r, and e's attributes:

$$\begin{aligned}&Rule\ X : can\_access(s, r, e) \leftarrow \\ &\qquad f(ATTR(s),\ ATTR(r),\ ATTR(e)).\end{aligned}$$

ABAC clearly provides an advantage over traditional RBAC when extended into SOA environments, which can be extremely dynamic in nature. ABAC policy rules can be custom-defined with consideration for semantic context and are significantly more flexible than RBAC for fine-grained alterations or adjustments to a subject's access profile. ABAC also integrates seamlessly with XACML, which relies on policy-defined attributes to make access control decisions. One additional benefit to Web service implementations of ABAC lies in the nature of the loose definition of subjects. Because ABAC provides the flexibility to associate policy rules to any actor, it can be extended to Web service software agents as well [11].

But With ABAC, as shown in [9] the failure to capture a business requirement for SoD would be the result of imprecise or erroneous rule definitions.

Separation of duties to minimize risk exposure remains important regardless of what type of authorization technique is used. Enforcing SoD is therefore equally important with ABAC as with RBAC. So in the proposed model we will define new policy rules to address this problem.

## 3. SEPARATION OF DUTY

Separation of Duties (SoD) is a traditional security control predating information systems and is particularly well-known in financial accounting. Some of its more common forms are known by popular monikers such as "four-eye-principle" or "two-man rule". The essential aim of SoD is to recognize potential conflicts of interest in activities which could result in errors or fraud. Where potential conflicts of interest arise, activities are structured such that different individuals perform specific steps of the overall activity. As a simple example, in a purchasing process, the person who requests a purchase usually is not the same person who approves purchases. Distributing responsibilities reduces the impact that a single individual can have, requiring collusion to perpetuate a fraud [12].

Adequate separation of duties reduces the likelihood that errors (intentional or unintentional) will remain undetected by providing for separate processing by different individuals at various stages of a transaction and for independent reviews of the work performed. The separation of duties provides four primary benefits: 1) the risk of a deliberate fraud is mitigated as the collusion of two or more persons would be required in order to circumvent controls; 2) the risk of legitimate errors is mitigated as the likelihood of detection is increased; 3) the cost of corrective actions is mitigated as errors are generally detected relatively earlier in their lifecycle; and 4) the organization’s reputation for integrity and quality is enhanced through a system of checks and balances.

Separation of duties is a basic, key internal control and one of the most difficult to accomplish. In essence, there is greater assurance that internal control responsibilities will be fully deployed when there is increased dispersion of such responsibilities among multiple individuals and work groups.

Most authors cite Saltzer and Schroeder [1] as the first authors who mentioned the concept of SoD, at that time under the name separation of privileges. A control is a planned measure or countermeasure designed to mitigate a risk or assure the integrity of activities in pursuit of an organization’s goals [13]. With specific regards to computing technology, security controls are measures taken "to protect the confidentiality, integrity, and availability of the system and its information” [14].

## 4. ROLES VS. RULES

In the RBAC model, permissions are not directly assigned to users but are instead collected in roles (see figure 1). Every role represents a special scope of permissions. A user is assigned to one or more roles, thereby acquiring permissions defined for the roles.

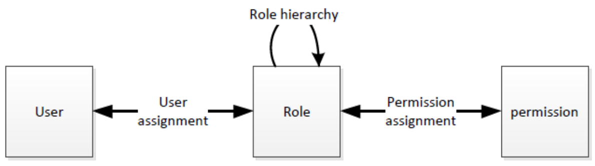


**Fig 1: Access permission based on Role**

In the proposed approach, permission is granted to the user using a set of policy, which these policies is consist of a set of rules (see figure 2).

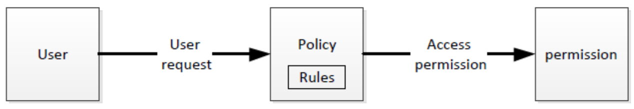


**Fig2: Access permission based on Rule**

In the proposed approach that is based on ABAC, Policy representation is semantically richer and more expressive, and can be more fine-grained because it can be based on any combination of subject, resource, and environment attributes. So, next section shows the new definition of policy rules for access controlling that is based on ABAC model.

## 5. PROPOSED ACCESS CONTROL MODEL

The proposed model is defined in terms of its authentication, authorization architecture and policy formulation.

### 5.1 Authentication

Authentication in service oriented architecture model, introduces some challenges. When a service is called from different ways and by users whose are in different organizations, how should the authentication process be done? WS-Security, WS-Trust, and WS-SecurityPolicy provide a basic model for federation between Identity Providers and Relying Parties. These specifications define mechanisms for codifying claims (assertions) about a requestor as security tokens which can be used to protect and authorize web services requests in accordance withpolicy. WS-Federation extends this foundation by describing how the claim transformation model inherent in security token exchanges can enable richer trust relationships and advanced federation of services.

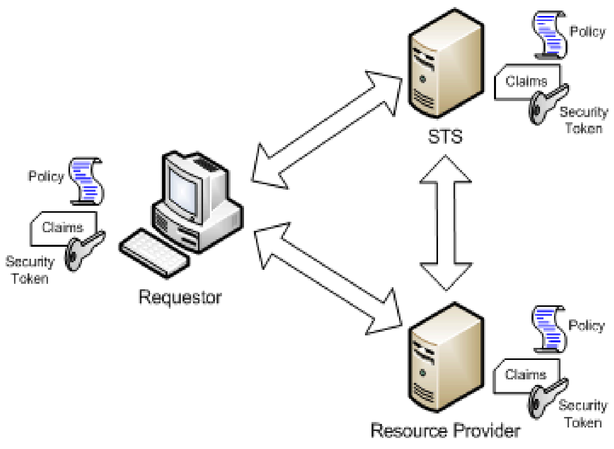


**Fig 3: Security Token Service (STS) Model**

Figure 3, illustrates a view of the Security Token Service (STS) model defined by WS-Trust [15]. Each arrow represents a possible communication path between the participants. Each participant has its own policies which combine to determine the security tokens and associated claims required to communicate along a particular path.

```
<s:Envelope>
  <s:Header>
    <wsa:Action>
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue
    </wsa:Action>
    <!-- Other headers not shown for brevity -->
  </s:Header>
  <s:Body>
    <wst:RequestSecurityToken>
      <wst:TokenType>
        http://example.org/mySpecialToken
      </wst:TokenType>
      <wst:RequestType>
        http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
      </wst:RequestType>
    </wst:RequestSecurityToken>
  </s:Body>
</s:Envelope>
```

**Fig 4:Example Request Security Token (RST) message[15]**

WS-Trust introduces protocol mechanisms independent of any particular application for requesting, issuing, renewing, cancelling and validating security tokens which can be exchanged to authenticate principals and protect resources. The core of this protocol is the request-response message pair, Request Security Token (RST) and Request Security Token Response (RSTR).

The fragmentwhich is shown in figure 4 is a simple example of a Request Security Token (RST) message, illustrating the token type being requested (mySpecialToken) and the request type in the body of the message (Issue). Other request types are possible, for example: Cancel Renew and Validate. WS-Trust is not limited to any particular token type. This example illustrates a custom token type being requested.

```
<s:Envelope>
  <s:Header>
    <wsa:Action>
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Issue
    </wsa:Action>
    <!-- Other headers not shown for brevity -->
  </s:Header>
  <s:Body>
    <wst:RequestSecurityTokenResponseCollection>
      <wst:RequestSecurityTokenResponse>
        <wst:RequestedSecurityToken>
          <xyz:CustomToken xmlns:xyz="...">
            ...
          </xyz:CustomToken>
        </wst:RequestedSecurityToken>
      </wst:RequestSecurityTokenResponse>
    </wst:RequestSecurityTokenResponseCollection>
  </s:Body>
</s:Envelope>
```

**Fig 5: Example Request Security Token Response (RSTR) message [15]**

The fragment in figure 5 is an example Request Security Token Response (RSTR) message, illustrating a successful (and simple) issuance of the custom token from the previous RST example. Other responses are possible, including challenges that must be met before the requested token would be issued.

## 5.2 Authorization Architecture

The proposed architecture for controlling access to protected resources is shown in Figure 6.The main parts of this architecture are:

- Policy Administration Point (PAP): PAP is the repository for the policies and provides the policies to the Policy Decision Point (PDP).
- Policy Enforcement Point (PEP): PEP is actually the interface of the whole environment to the outside world. It receives the access requests and evaluates them with the help of the other actors and permits or denies the access to the resource.
- Policy Decision Point (PDP): PDP is the main decision point for the access requests. It collects all the necessary information from other actors and concludes a decision.
- Policy Information Point (PIP): PIP is the point where the necessary attributes for the policy evaluation are retrieved from several external or internal actors. The attributes can be retrieved from the resource to be accessed, environment (e.g., time), subjects, and so forth.
- Audit & Report Service:The system must provide the completed log mechanism to record all the events (requests, actions and decisions that happened) and the related identity as the audit clue. Regular security audit is useful to find out the security flaws, violating security behavior, cheating and other behaviors that attempt to bypassed safety measures. Besides, the system should apply for the prevention measures such as load balancing, virus testing, packet filtering and fault switching or backup to prevent service denial of attack or other potential security attack.
- Conflict monitor: This point checks that SoD constraints won't be violated and captures the required policies and information from policy repository. In fact this point is part of PAP, PDP and uses from Audit point to check history of actions.

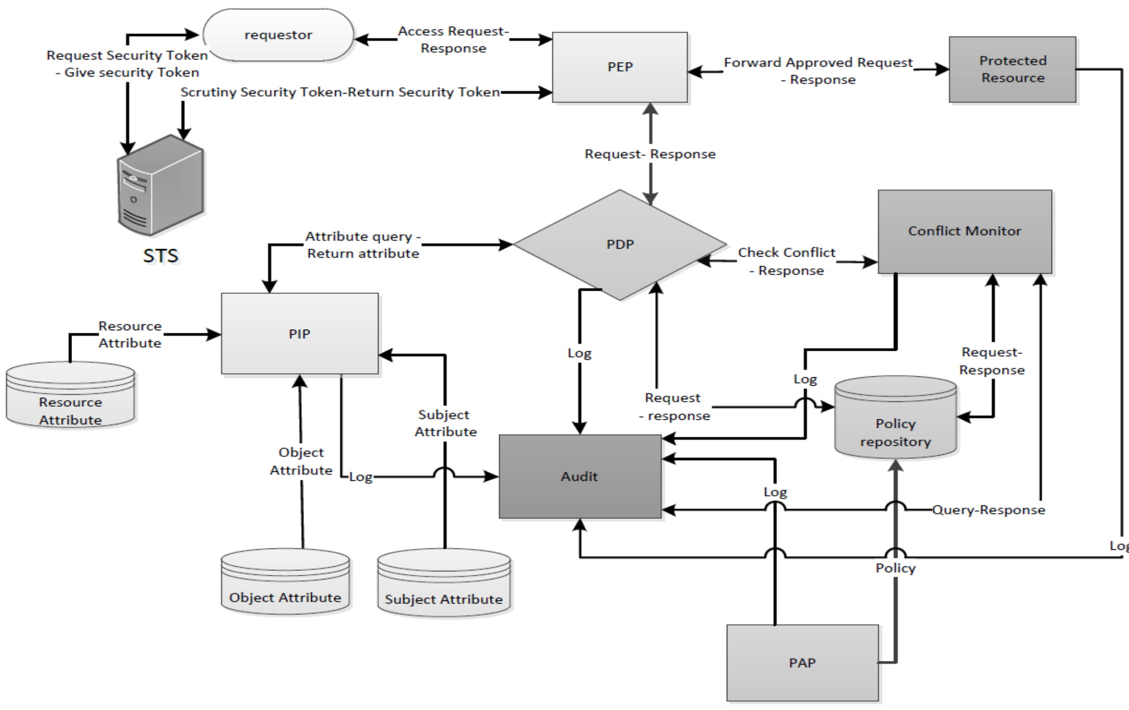


**Fig 6: The proposed architecture for access control**

## 5.3 Policy Formulation

The policy rule definition of this model is based on ABAC model [5] but uses new proposed definition of action and attribute based rules to monitor the SoD conflicting actions. For defining SoD rules, first we should determine what actions or resources are in conflict together and should be controlled for SoD violations. No two SoD conflicts are equal; each conflict poses a different risk to the business and ideally, each conflict should be rated according to the likelihood and impact of a user executing the conflicting actions. Companies have adopted many schemes and notations for risk-ranking their conflicts. Companies have defined tiers of high, medium and low based on the output of their risk calculation. High risk shows that performing the conflicting actions should not be allowed, the medium risk says that the actions are allowed but should be mitigated, and low risks show that the conflicting actions may be allowed and business should be aware of possible risk, recommended to be mitigated.
We introduce the main entities in our definition as follows:

- S= subject,$\{s1, s2, s3, \ldots, sn\} = S$
- R=protected resource,$\{r1, r2, r3, \ldots, rn\} = R$
- A= the requested action,$\{a1, a2, a3, \ldots, an\} = A$
- E= environment,$\{e1, e2, e3, \ldots, en\} = E$
- O= obligations,$\{o1, o2, o3, \ldots, on\} = O$
- P= permission,$\{Permit, Deny\} = P$

And *Policy Rules* that decide on whether a subject s can perform action a, for resource r, in a particular environment e, is Boolean functions as below:

$$Rule\ Y: can't\_perform(s, r, e, a) \leftarrow$$

$$f(s, r, e, a)$$

and $Rule\ Z: can\_perform(s, r, e, a, o) \leftarrow$

$$f(s, r, e, a, o)$$

That the functions show subset of subjects, resources, environments and actions respectively. Rule Y is used for the actions with high risk, and Rule Z is used for the actions with medium and low risks. The element "o" contains the Obligations that must be fulfilled so that the decision can be enforced. These obligations are used for mitigating control.
For example consider this SoD constraint: A subject who can do the conflicting actions cannot do both of them for the same resource in one business task. In this case we suppose that there are two conflicting actions a1 and a2 and there is high risk if one subject performs these two actions in one transaction. Subject s1 is requests both actions on same resource. In figure 7 the subject s1 has performed action a1 for protected resource r1.

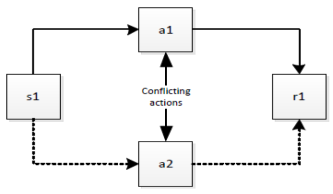


**Fig 7: One subject requests two Conflicting action for one resource**

If the subject s1 requests action a2 for the same resource r1 then he can execute two conflicting actions on r1 in one business task which is against the concept of separation of duty. The following condition should be paid attention for this case:

**Table 1. One subject requests two conflicting action in one business task**

| Subject | Action | Resource | Permission |
|---|---|---|---|
| s1 | a1 XOR a2 | r1 | permit |
| s1 | a1 AND a2 | r1 | Deny |

If we follow the proposed approach for definition of dynamic separation of duty as shown in Table 1, then the subject s1 cannot perform both actions a1 and a2 at same time or at some other time on resource r1. According to the proposed definition of dynamic separation of duty, the concept of separation of duty will be implemented properly. So the rule definition should be as follow:

$$Rule\ Y: can't\_perform(s, r, e, a) \leftarrow$$

$$(s1 \wedge r1 \wedge (a1 \wedge a2))$$

So it is understood that the resources which the subject s1 has already performed action a1 or a2 on them and also these actions are important so the history of resources and actions should be maintained. All events that happen are logged in audit point in this model.
Consider the following example of history based constraint [16]: "an employee cannot service a request from company B if he has already serviced a request by company A".
So in this case we use this definition: A subject can't perform an action for a resource if he had performed that action for another resource before (see figure 8).
Suppose that subject s1 has performed action a1 for resource r1.

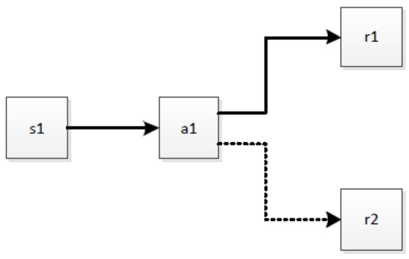


**Fig 8: One subject requests an action for two conflicting resources**

If subject s1 requests same action for another resource s2 then it isn't a valid request and subject s1 shouldn't be authorized for performing this action.
Table 2 shows the conditions for defining appropriate rules.

**Table 2. One subject requests for performing an action on two resources**

| Subject | Action | Resource | Permission |
|---|---|---|---|
| s1 | a1 | r1 XOR r2 | permit |
| s1 | a1 | r1 AND r2 | Deny |

If we follow the table 2 to defining rule for history based separation of duty, then the subject s1 cannot perform action a1 for resource r2 if he has already done action a1 for resource r1, and the subject s1 cannot perform action a1 for resource r1 if he has already done action a1 for resource r2. So, the rule definition should be as follow:

$$Rule\ Y: can't\_perform(s, r, e, a) \leftarrow$$

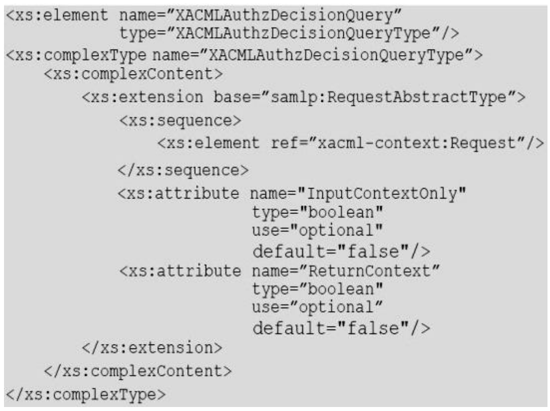
$$(s1 \wedge (r1 \wedge r2) \wedge a1)$$

We introduced some definition that should be used for designing policy rules. For simplifying we didn't introduce environment situation in these instance cases. It can extend to define more complicated situations and constraints. For example for concurrency control the current performing action for a subject on a resource should be available.

## 6. SEQUENCE DIAGRAM FOR ACCESS TO A PROTECTED RESOURCE

In order to have a better understanding of the work done when there is a request for a supported resource, the sequence diagram for the proposed architecture is shown in Figure 9.To complete the descriptions, below is a set of stages for a relationship between a web service program and a specific resource.

1) First, a user is connected to a web service program and requests the security token from STS for use of a protected (resource) service. STS gives the security token to user.
2) A SOAP message is generated and embeds a SAML authentication assertion in the SOAP header. These provisions include the user's security token that is ultimately sent for the PEP unit.



```
<xs:element name="XACMLAuthzDecisionQuery"
          type="XACMLAuthzDecisionQueryType"/>
<xs:complexType name="XACMLAuthzDecisionQueryType">
    <xs:complexContent>
        <xs:extension base="samlp:RequestAbstractType">
            <xs:sequence>
                <xs:element ref="xacml-context:Request"/>
            </xs:sequence>
            <xs:attribute name="InputContextOnly"
                          type="boolean"
                          use="optional"
                          default="false"/>
            <xs:attribute name="ReturnContext"
                          type="boolean"
                          use="optional"
                          default="false"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

**Fig 10:Needed labels for the request XACMLAuthzDecisionQuery [2]**

3) The handler on the Web service side acts as the SAML PEP. It intercepts the received SOAP

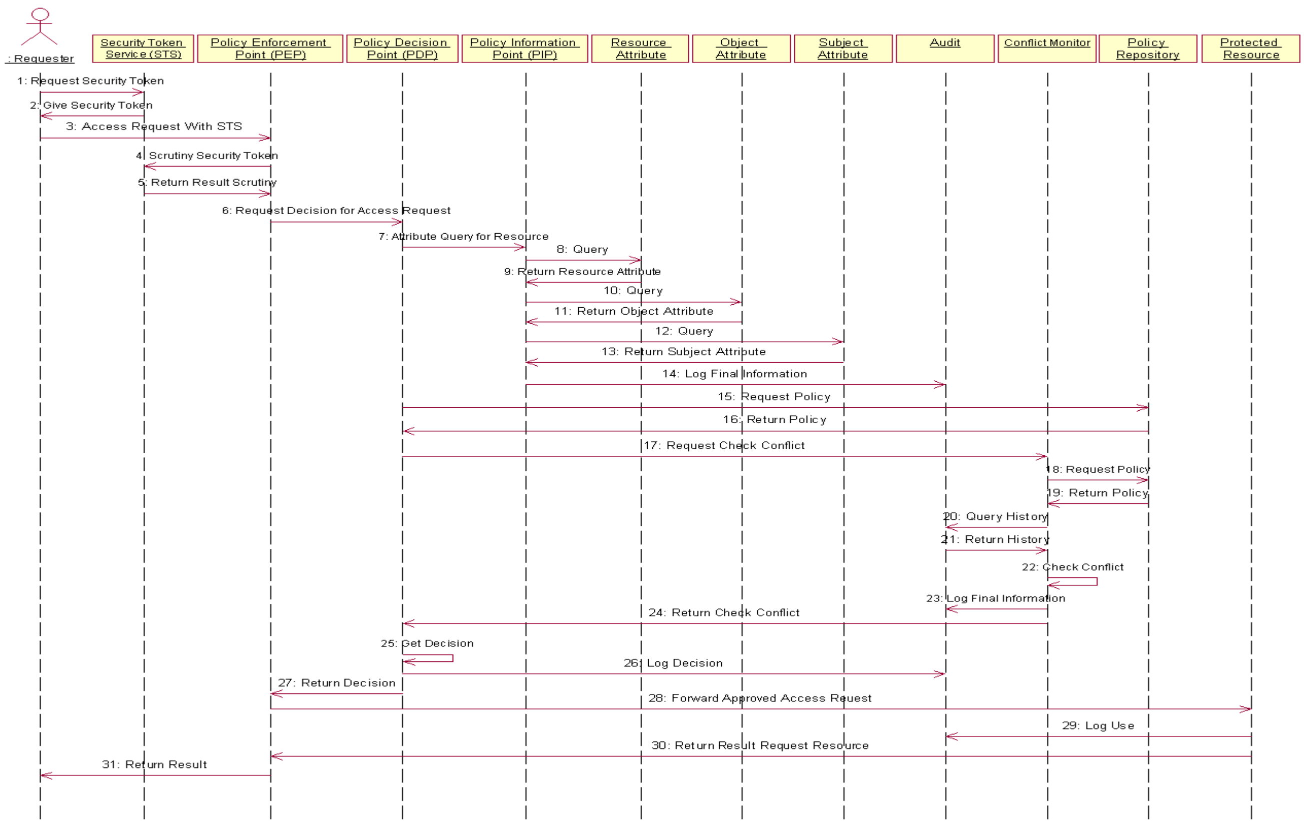
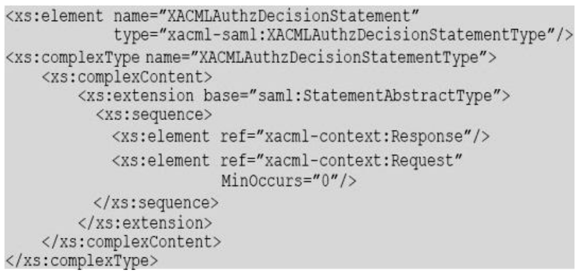

: Requester
Security Token Service (STS)
Policy Enforcement Point (PEP)
Policy Decision Point (PDP)
Policy Information Point (PIP)
Resource Attribute
Object Attribute
Subject Attribute
Audit
Conflict Monitor
Policy Repository
Protected Resource
1: Request Security Token
2: Give Security Token
3: Access Request With STS
4: Scrutiny Security Token
5: Return Result Scrutiny
6: Request Decision for Access Request
7: Attribute Query for Resource
8: Query
9: Return Resource Attribute
10: Query
11: Return Object Attribute
12: Query
13: Return Subject Attribute
14: Log Final Information
15: Request Policy
16: Return Policy
17: Request Check Conflict
18: Request Policy
19: Return Policy
20: Query History
21: Return History
22: Check Conflict
23: Log Final Information
24: Return Check Conflict
25: Get Decision
26: Log Decision
27: Return Decision
28: Forward Approved Access Request
29: Log Use
30: Return Result Request Resource
31: Return Result

**Fig 9: The sequence diagram for access to a protected resource**

message and requests the validation of the security token. Then it determines the assigned role of the requester as subject through identity management module, the operation name extracted from the SOAP body as action, and the name of Web service as resource. With these required factors, SAML PEP generates a XACMLAuthzDecisionQuery (request for decisions based on XACML) and sends it to SAML PDP. Structure of this request in the form of labels is shown in Figure 10.

4) SAML PDP is the point to actually make access decision. It retrieves the attributes required from attribute repository, such as LDAP or database, likedbXML, a native XML database to store XML.
5) The PDP point uses from conflict monitor unit to check SoD constraints not violated.
6) If the history of actions and attributes needs for some policies they will be retrieved from audit unit and then evaluates whether the request should be granted or not. The results of every stage are saved in Review and the Log unit. Finally it constructs a XACMLAuthzDecisionStatement as a SAML response containing a XACML response context and sends it back to SAML PEP in response to the XACMLAuthzDecisionQuery. This step achieves the authorization of the user on the Web service side. The label for the answer is also shown in Figure 11. Also the decisions adopted are saved in the log unit.



```
<xs:element name="XACMLAuthzDecisionStatement"
            type="xacml-saml:XACMLAuthzDecisionStatementType"/>
<xs:complexType name="XACMLAuthzDecisionStatementType">
    <xs:complexContent>
        <xs:extension base="saml:StatementAbstractType">
          <xs:sequence>
            <xs:element ref="xacml-context:Response"/>
            <xs:element ref="xacml-context:Request"
                    MinOccurs="0"/>
          </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

**Fig 11: Needed labels for the response XACMLAuthzDecisionStatement [2]**

7) The PEP interprets the received answer and extracts the adopted decisions. If the result is PERMIT, the requester has the privilege to invoke Web service (resource); in other case the right to use the service is removed from user's program and it is not obtained.
8) Finally, after calling the service and logging it, the result is sent to the applicant's program.

## 7. RULE DEFINITION IN THE PROPOSED MODEL: A CASE STUDY

Now we use the new approach to defining rules. For example, a rule that dictates "Users with role 'Manager 'or 'sale clerk' may access the 'billingform' between '8:00' to '16:00' from their office and if their branches are same" can be written as:

$$\begin{aligned}&R1: can\_access(s,r,e)\\&\quad (Role(s) = ('SaleClerk') \vee Role(s) = ('manager') \wedge\\&\quad Name(r) = ('Billingform') \wedge\\&\quad Branch(s) = Branch(r) \wedge\\&\quad time(e) > 8{:}00 \wedge time(e) < 16{:}00 \wedge\\&\quad Location(s) = office).\end{aligned}$$

For this example we suppose that the actions “create” and “approve” have high conflict with each other. So from SoD constraint these two actions can’t be performed by one person in one business task. Table 3 shows that if a subject performs one of these actions for one resource then he shouldn't be able to perform another action on that resource in one workflow.

**Table 3.Performing two conflicting actions with high risk by one person**

| Subject | Action | Resource | Environment | Permission |
|---|---|---|---|---|
| s1 | Create XOR Approve | Billingform | Transaction code (‘Create’)= Transaction code (‘Approve’) | permit |
| s1 | Create AND Approve | Billingform | Transaction code (‘Create’)= Transaction code (‘Approve’) | Deny |

Then:

$$\begin{aligned}&R2: can't\_perform(s,r,e,a)\\&\quad (Subject = s1 \wedge\\&\quad resource = Billingform \wedge\\&\quad Actions = (create \wedge approve) \wedge\\&\quad Transaction\ code('Create') = Transaction\ code('Approve')\end{aligned}$$

So the final rule would be**:**

$$R3 = R1 \wedge R2$$

## 8. CONCLUSION

In this paper we proposed an access control model for web services. In this model Abstract and generally applicable policiesreplace static mappings between users andinformation assets. Evaluation of access requestsis dynamic and executes in real-time whichmakes it easier to open up systems for new usersand new use case scenarios.Not only this model has thebenefits of ABAC model but also represents a way todefining SoD rules to prevent related violations. The benefits of defining rules in this way were showed with a case study. Theproposed architecture is based on XACML standard. Ifnew rules in accessing the resources are necessary, theycan be defined dynamically, and then added to theapproach.

## 9. REFERENCES

[1] Saltzer, J.H., . Schroeder, M.D., 1975. The Protection of Information in Computer Systems. Proceedings of the IEEE, 63(9):1278-130.

[2] Anderson, A. and Lochhart, H., 2005. SAML 2.0 profile of XACML, OASIS Standard.

[3] Moses, T. and et al, 1 Feb 2005. eXtensible Access Control Markup Language(XACML) Version 2.0, OASIS Standard

[4] Singhal, A., Winograd, T. and Scarfone, K., 2007. Guide to Secure Web Services, National Institute of Standards and Technology Special Publication.

[5] Yuan, E. and Tong, J. 2005. Attributed Based Access Control (ABAC) for Web Services, IEEE International Conference on Web Services (ICWS’05).

[6] Sandhu, R. S. and et al,1996. Role-Based Access Control Models, IEEE Computer, pp. 38-47.

[7] Ferraiolo, D.F. and Kuhn, D.R., 1992. Role Based Access Control, 15th National Computer Security Conf., pp: 554-563.

[8] National Institute of Standards and Technology, December2010. 2010 Economic Analysis of Role-Based Access Control, Final Report.

[9] Axiomatics white paper, 2009. Enforcing Segregation of Duties (SoD).

[10] Karp, A. H. and Li, J. 2010. Solving the Transitive Access Problem for the Services Oriented Architecture, IEEE International Conference on Availability, Reliability and Security, DOI 10.1109/ARES,

[11] Tong, J. Apr. 2005. Attribute based access control: a new access control approach for service oriented architectures, Workshop on New Challenges for Access Control, Ottawa, Canada.

[12] Giblin, C., Hada, S., 2008. Towards Separation of Duties for Services, IBM Research Laboratory.

[13] The Institute of Internal Auditors: Glossary. Viewed 2008.

[14] National Institute of Standards and Technology (NIST), U.S. Department of Commerce, December, 2007. Recommended Security Controls for Federal Information Systems, NIST Special Publication 800-53.

[15]Goodner, M. Hondo, M. Nadalin, A. Mcintosh, M. Schmidt, D. 2007. Understanding WS-Federation. International Business Machines (IBM) Corporation and Microsoft Corporation publication . Location:http://msdn.microsoft.com/en-us/library/bb498017.aspx.

[16] Clark, D., and Wilson, D., April 1987. A comparison of commercial and military computer security policies. In IEEE Symposium on Security and Privacy, pages 184–194, Oakland.