Definition of Action and Attribute Based Access Control Rules for Web Services

Hadiseh Seyyed Alipour
Department of Software Engineering
Qazvin Islamic Azad University, Qazvin 1416-34185, Iran

Mehdi Sabbari
Department of Software Engineering
Qazvin Islamic Azad University, Qazvin 1416-34185, Iran

Abstract

Access control 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. The overall objective of any access control policy rule is to define, sensitive information may be disclosed when, why, how and to whom. In one hand resources should be protected against unauthorized access and in the other hand it should be available for authorized use. Separation of duties (SoD) is a security principle that has been used extensively to prevent conflict of interest, fraud and error control in organizations. In recent years many IT organizations have struggled to identify potential SoD violations within their IT systems. Since new technologies such as SOA widely used in IT systems, in this article we describe two access control model: most recently used which is RBAC model and most suitable for SOA environment which is ABAC and compare them in SoD Issue with an example. We show that failure to capture a business requirement for SoD, and then poor rule definitions can lead to violation. Hence we propose an approach for defining rules, based on subject, resource, environment attributes and the action that would be performed.

Keywords
Separation of Duty, Rule Definition, Web Services, Action, Attribute

1. Introduction

SOA establishes an architectural model that aims to enhance the efficiency, agility, and productivity of an enterprise by positioning services as the primary means through which solution logic is represented in support of the realization of strategic goals associated with service-oriented computing (Erl, 2007). SOA supplies various services to process business by joining various specific components together in a loose coupled manner with uniform interfaces (Singhal et al. 2007). SOA and the corresponding Service-Oriented Computing (SOC) have received significant attention recently as major computer and software companies, such as IBM, Intel, Microsoft, HP, SAP, and Sun Microsystems, as well as government agencies, such as U.S. Department of Defense, have embraced SOA/SOC (Wang et al. 2009). New technologies and methodologies, such as SOA, facilitate the integration of legacy information systems with new system components and the dynamic outsourcing of business functionality. These advances enable organizations to concentrate on mission critical and value generating business activities and to outsource less central activities. Web services seem to become the preferred implementation technology for realizing the SOA promise of maximum service sharing, reuse, and interoperability (Parveen and Tilley 2008, Papazoglou and Heuvel 2007). Web Service supports the communication between applications developed on different platform by different programming languages with different technological standards. The ultimate goal of Web Service is to realize the integration and interaction between various systems in Internet/Intranet environment, just similar to the function of components (Wang et al. 2009). Web services technology platform is comprised of the following core open technologies and specifications: Web Services Description Language (WSDL), XML Schema Definition Language (XSD), SOAP (formerly the Simple Object Access Protocol), UDDI (Universal Description, Discovery, and Integration), and the WS-I Basic Profile (Erl 2007, Karp 2006). 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 (Saltzer and Schroeder 1975). One of these principles is Separation of Duty (SoD). Separation of Duties is an essential security control for managing the integrity of information technology systems and human processes. It is used extensively to prevent conflict of interest, fraud and error control in organizations. The organizations should enforce internal controls in order to fight fraud and to comply with regulatory requirements. Access control security is one of the important aspects in SOA that is considered as a challenge. 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 control models are abstractions that incorporate the rules and parameters required to execute access control policies. Since multiple mechanisms can be constructed to support a particular access control policy, access control models provide a framework for policy implementation. Application of the model promotes consistent access control mechanisms across platforms, which lowers costs, increases security, and supports interoperability (Gallaher and et al. 2002).

In this paper we explain two most well-known access control models that are used recently, and we will discuss about the advantages and disadvantages of them. We compare these models about enforcing separation of duty issue with an example. Then we proposed our approach to defining SoD policy rules in ABAC model. The remainder of this paper is organized as follows: Section 2 describes separation of duties, section 3 explains RBAC and ABAC access control model and says which one is better for using in SOA environment, section 4 compares the RBAC and ABAC models with an example about SoD issue, section 5 compares Access permission based on Role and Rules, section 6 proposes an approach to define SoD rules in ABAC policy rule model and section 7 gives a case study for this approach. Finally section 8 concludes the paper.

2. Separation of Duties
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 (Giblin and Hada 2008). 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 (Saltzer and Schroeder 1975) 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 (Glossary 2008). With specific regards to computing technology, security controls are measures taken "to protect the confidentiality, integrity, and availability of the system and its information" (NIST 2007).

3. RBAC and ABAC Access Control Models
A number of models have been developed to address various aspects of access control problem. But in this section we describe two authorization models most relevant to access management in a SOA, namely, Role Based Access Control (RBAC) and Attribute Based Access Control (ABAC).

3.1 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 (Sandhu and et al. 1996). 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 (Ferraiolo and Kuhn 1992). In 2004, the American National Standard and International Committee for Information Technology Standards (ANSI/INCITS) approved an RBAC standard that combined features of the models introduced in Ferraiolo and Kuhn and Sandhu et al. The standard defined four levels of RBAC with their respective administrative, system, and review functions: 1) core RBAC; 2) hierarchical RBAC; 3) static separation of duties (SSD) RBAC, 4) dynamic separation of duties (DSD) RBAC. Thus RBAC can be used to enforce SoD within digital systems by identifying toxic combinations of permissions and rendering them mutually exclusive. RBAC’s use of roles as an abstraction of the connection between users and privileges simplifies management, contributing to economy of mechanism and making correct privilege assignment easier because roles reflect user jobs. Although RBAC may take slightly different forms, a common representation as defined in (WU and XI 2009) is depicted in Figure 1.

![Figure 1: Role-based access control model](image-url)

In most cases, Web service platforms will support designation and assignment of privileges to roles as part of their standard definition of user accounts and access control privileges. In worst cases, the administrator will have to create the necessary user groups, enroll the appropriate users, and assign them role-appropriate privileges. RBAC on a Web service platform should be implemented at a minimum for the administrator, developers, and any other privileged accounts that will be required for the Web service to operate. The Web service platform must be configured to enforce separation of roles (i.e., not allowing a user assigned to one role to perform functions exclusively assigned to another role). The privileges associated with each role should be assigned in a way that implements least privilege each role should be assigned only the minimum privileges needed to perform the functions required by the role (Rolls 2008). RBAC is recognized for simplifying access control administration and improving visibility of both the access control policy and the organizational structure (Gallaher and et al. 2002) because:

- Deprovisioning is automated by removing the roles that no longer apply to the user’s new position. If the organization upgrades a system, an IT administrator needs to only update the new permissions to the appropriate roles, and the permissions will be propagated to all relevant users via roles.

- Review of the access control policy is made easier because roles already may contain the user and permission information in a central location. Understanding the access control policy that is currently enforced within the organization is crucial to identify security threats and may aid in the attestation and auditing required by law.

Thus RBAC has several advantages and it is fine within a single security domain, like an application. But it has some limitation in SOA environment:

- In RBAC you 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. But users coming in through the Internet channel are not only guests or customers, but they can also be employees, partners, external service providers, you name it. There may be plenty accounts you don't want to manage, because they don’t belong to your security domain. Unless you manage different portals for different roles and have some form of identity management within the portal environment in place, RBAC is of no use.

- RBAC has a problem with the concept of context. If someone has a role, then he will get the permissions associated with that role automatically. There is no restriction in place based on the concept of context. A context could be the channel used (internet, LAN), time of day, legal entity etc. An account using a wireless home PC might not deserve the same permissions as the same account in an office using a centrally managed workplace. But such a concept does not exist in RBAC terms (unless you define lots of extra roles). RBAC must always be complemented with a rule-based access control function.

- Another big objection is that there is no possibility for user profiling. Meaning authorization based on the skills of an employee instead of the single fact that the user is connected to a role. RBAC doesn’t know the
difference between junior or senior employees within the same role. Unless you define separate roles. And creating duplicate roles is not why you implement RBAC.

3.2 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 (Karp and Li 2010). Unlike IBAC and RBAC, the ABAC model (Yuan and Tong 2005) 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:

\[
\text{Rule: } \text{can}_{ac}(s, r, e) \leftarrow f(ATTR(s), ATTR(r), ATTR(e))
\]

Given all the attribute assignments of s, r, and e, if the function’s evaluation is true, then the access to the resource is granted; otherwise the access is denied. A Policy rule base or Policy Store may consist of a number of policy rules, covering many subjects and resources within a security domain. The access control decision process in essence amounts to the evaluation of applicable policy rules in the policy store. Typical attribute-based authorization architecture is shown in Figure 2.

The diagram reflects the following logical actors involved in an ABAC model:

- The Attribute Authorities (AA) are responsible for creating and managing the attributes for subjects, resources, and the environment, respectively.
- The Policy Enforcement Point (PEP) is responsible for requesting authorization decisions and enforcing them. In essence, it is the point of presence for access control and must be able to intercept service requests between information consumers and providers.
- The Policy Decision Point (PDP) is responsible for evaluating the applicable policies and making the authorization decision (permit or deny). The PDP is in essence a policy execution engine. When a policy references a subject, resource, or an environment attribute that is not present in the request, it contacts the appropriate AA to retrieve the attribute value(s).
- The Policy Authority (PA) creates and manages access control policies. The policies may consist of decision rules, conditions, and other constraints for accessing the resources.

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. 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 (Singhal 2007). 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) (Moses and et al. 2005). 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 (Tong 2005).

4. Comparing Separation of Duty in RBAC and ABAC

In this section we will compare the two models that explained above with an example about enforcing SoD. 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. But this example reveals some important differences as illustrated below (Axiomatics white paper 2009):

With RBAC as shown in figure 3 the violation occurs when permissions contained in two different roles are in conflict. Users having both Role 1 and 2 in the above figure are affected whereas those who have either Role 1 or Role 2 in combination with other roles are not. Altering Role 1 and/or 2 therefore often leads to cascading role redefinitions and/or role explosions. Unaffected users in group 1 and 3 may be impacted by measures taken to handle a conflict within group 2. The role concept may thus expand the effects of SoD resolution into innocent parts of the user population. The role concept also expands the problem across unaffected permissions. Although only individual permissions within Role 1 and Role 2 are in conflict they contaminate their respective roles. 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.

With ABAC, as shown in Figure 4 the failure to capture a business requirement for SoD would instead be the result of imprecise or erroneous rule definitions. For example the rule was defined as: "if action= approve, resource= payment, subject has a Driver’s License then permit. The rule definition saying "Yes, you may approve payments, provided you
have a driver’s license”, as illustrated above, would not adequately capture the business risk we want to avoid. Furthermore, this type of rule error may also affect an innocent user. A legitimate transaction could be blocked because the user does not have a driver’s license, which from a business risk perspective is rather irrelevant.

This example illustrates that:

- SoD requirements on authorization require the ability to provide conditional decisions based on contextual data. The RBAC model as opposed to the ABAC model is not context-aware and thus not well suited to handle SoD requirements.

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.

Many leading vendor IdM solutions implement SoD policy using roles which allow the IdM administrator to define which role combinations are invalid in their organization. The prerequisite of this task is clearly that an organization must have already distilled the existing user entitlements found across the business into a set of roles. To attain the level of role granularity required to implement even a simple set of SoD rules for a small set of business applications (COTS or infrastructure such as Active Directory), requires the definition and management of a massive number of subtly different roles. Such an implementation within even a small to medium sized business can lead to so called 'Role Explosion'. So, a better approach would be to give an organization the ability to define SoD policies on the actual account entitlements, instead of a set of logical interpretations that do not actually exist anywhere. As showed above, ABAC is much better approach to define SoD policy rules on the actual account entitlements but the failure to capture a business requirement for SoD would instead be the result of imprecise or erroneous rule definitions. Therefore an approach should be structured to define what rules should consist SoD constraints and what attribute should be considered for subjects, resources and actions to address these problems.

5. Roles & Rules

In the RBAC model, permissions are not directly assigned to users but are instead collected in roles (see figure 5). Every role represents a special scope of permissions. A user is assigned to one or more roles, thereby acquiring permissions defined for the roles.

![Figure 5. Access permission based on Role](image)

In our previous work we proposed a policy based access control model (Seyyed alipour et al. 2011). 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 6).

![Figure 6. Access permission based on Rule](image)

In this 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.
6. Defining SoD Rules to ABAC Policy Model

Today's solutions for defining SoD rules are based on role definition and can't be used for ABAC. As said before ABAC uses a set of attributes and a number of policy rules which use from these attributes to make a decision for permit or deny a request. Thus an approach should be defined to allows IdM and policy administrators to express SoD rules in terms of business application attributes, thereby moving away from an RBAC centric approach and more towards ABAC. In this section we propose definitions for SoD constraints that can be used for rule definition and apply appropriate attributes for this reason. Since, ABAC model is based on attributes and policies for making decision, first we should determine what actions or resources are in conflict with each other and should be controlled for SoD violations. Because each policy rule shows that the permission to a subject for a particular action, can be granted or should be denied. 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 follow:

- S= subject,
  \[ \{s_1, s_2, s_3, \ldots, s_n\} = S \]
- R=protected resource,
  \[ \{r_1, r_2, r_3, \ldots, r_m\} = R \]
- A= the requested action for protected resource,
  \[ \{a_1, a_2, a_3, \ldots, a_n\} = A \]
- E= environment,
  \[ \{e_1, e_2, e_3, \ldots, e_n\} = E \]
- O= obligations,
  \[ \{o_1, o_2, o_3, \ldots, o_n\} = O \]
- P= permission,
  \[ \{\text{Permit, Deny}\} = P \]

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

\[
\begin{align*}
\text{Rule Y: can't perform } & (s, r, e, a) \leftarrow \\
& f(s, r, e, a)
\end{align*}
\]

and

\[
\begin{align*}
\text{Rule Z: can perform } & (s, r, e, a, o) \leftarrow \\
& f(s, r, e, a, o)
\end{align*}
\]

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 defining separation of duty rules the instance cases below should be paid attention:

6.1 Case1

Definition: 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. Action a1 and a2 comprise one complete business task. Subject s1 is requests both actions on one resource. In figure 7 the subject s1 has performed action a1 for protected resource r1.
If the subject s1 requests action a2 for the same resource r1 then he can execute the whole 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

<table>
<thead>
<tr>
<th>Subject</th>
<th>Action</th>
<th>Resource</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>a1 XOR a2</td>
<td>r1</td>
<td>permit</td>
</tr>
<tr>
<td>s1</td>
<td>a1 AND a2</td>
<td>r1</td>
<td>Deny</td>
</tr>
</tbody>
</table>

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. We can use from Table1 to define appropriate policy rule. In Table1 it is shown that, if performing action a1 by subject s1 for resource r1 is granted then performing action a2 by subject s1 for resource r1 is denied, and if performing action a2 by subject s1 for resource r1 is granted then performing action a1 by subject s1 for resource r1 is denied. So, from (2) the rule definition should be as follow:

\[
\text{rule } x1: \text{can't perform}(s,r,e,a) \leftarrow \\
\quad \text{s1} \land r1 \land (a1 \lor a2)
\]

So we understood from this definition that the resources which the subject s1 has already performed action a1 or a2 on them are important so the history of resources that subject s1 is perform conflicting actions on them should be maintained.

6.2 Case2
Consider the following example of history based constraint (Clark and Wilson 1987): "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. Suppose that subject s1 has performed action a1 for resource r1.

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.
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, from (2) the rule definition should be as follow:

\[
\text{rule x1: can\_t\_perform(s,r,e,a) \leftarrow } \\
\text{s1} \land (r1 \land r2) \land a1
\]

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. In the next section we will use this definition to defining appropriate rules for the example that was discussed in section 4.

### 7. Case study

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

\[
R1: \text{can\_access(s,r,e)} \\
(Role(s) = ('SaleClerk') \lor Role(s) = ('manager') \land \\
Name(r) = ('Billingform') \land \\
Branch(s) = Branch(r) \land \\
time(e) > 8:00 \land time(e) < 16:00 \land \\
Location(s) = office).
\]

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. Table 3 shows that if a subject performs one of these actions for one resource and then he shouldn’t be able to another action on that resource in one workflow.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Action</th>
<th>Resource</th>
<th>Environment</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Create&amp;Approve</td>
<td>Billingform</td>
<td>Transaction code('create') = Transaction code('Approve')</td>
<td>permit</td>
</tr>
<tr>
<td>s1</td>
<td>Create &amp; Approve</td>
<td>Billingform</td>
<td>Transaction code('create') = Transaction code('Approve')</td>
<td>Deny</td>
</tr>
</tbody>
</table>

Then:

\[
R2: \text{can\_t\_perform(s,r,e,a)} \\
(Subject = s1 \land \\
resource = Billingform \land \\
Actions = (create \land approve) \land \\
Transaction code('create') = Transaction code('Approve')).
\]

So the final rule is:

\[
R3 = R1 \land R2
\]
8. Conclusion

In this paper we proposed an approach to defining rules in policy based access control model based on subject, resource and environment attributes and the action that would be performed. We compared ABAC model which is called PBAC with traditional access control model RBAC and showed that policy based model such as ABAC model is more suitable for dynamic Service Oriented environments and web services. But we explained that the imprecise or erroneous rule definitions can lead to fraud and unauthorized persons to access to important resources by an example about defining SoD rule. So for better defining the rules we extended the ABAC policy rule definition by considering the requested action. The benefit of defining rules in this way was showed with a case study.

References
Axiomatics white paper, Enforcing Segregation of Duties (SoD), 2009.
Yuan, E., and Tong, J., Attributed Based Access Control (ABAC) for Web Services, IEEE International Conference on Web Services (ICWS’05), 2005.