BP-XACML an authorisation policy language for business processes

Khalid Alissa[1,2] <Khalid.alissa>, Jason Reid[1]<jf.reid>, Ed Dawson[1]<e.dawson>, and Farzad Salim[1]<f.salim>

[1] Institute of Future Environment, Queensland University of Technology, Brisbane, QLD, Australia <@qut.edu.au>
[2] King Abdulaziz City for Science & Technology (KACST), Riyadh, Saudi Arabia

Abstract. XACML has become the defacto standard for enterprise-wide, policy-based access control. It is a structured, extensible language that can express and enforce complex access control policies. There have been several efforts to extend XACML to support specific authorisation models, such as the OASIS RBAC profile to support Role Based Access Control. A number of proposals for authorisation models that support business processes and workflow systems have also appeared in the literature. However, there is no published work describing an extension to allow XACML to be used as a policy language with these models. This paper analyses the specific requirements of a policy language to express and enforce business process authorisation policies. It then introduces BP-XACML, a new profile that extends the RBAC profile for XACML so it can support business process authorisation policies. In particular, BP-XACML supports the notion of tasks, and constraints at the level of a task instance, which are important requirements in enforcing business process authorisation policies.

Keywords: XACML, BPM, Workflow, Authorisation management, Authorisation policy language.

1 Introduction

The domain of 'Business Process Management (BPM)' is an important and maturing domain. A survey by Gartner [6] showed that BPM is the number one concern of many senior executives. This increasing interest in BPM has prompted research in a variety of directions, including the domain of access control. Several access control models designed specifically for the business process environment have been presented in the literature including [3], [11], and [16]. Most of these proposals focus on the authorisation model itself, and do not specify an authorisation policy language, which is an important aspect of authorisation management. One of the most accepted and widely discussed authorisation policy languages is the eXtensible Access Control Markup Language (XACML) [8].

XACML is an XML defined standard language for access control policies. XACML uses rules, which are defined in policies, in conjunction with a standard component called a Policy Decision Point (PDP). The PDP evaluates access requests against the policies to decide whether to allow or deny the request [9].

The XACML RBAC Profile [2] was proposed to extend the initial version of XACML. It supports the notion of roles to be able to support role-based access control policies [2]. The RBAC profile supports both core and hierarchical RBAC, but it explicitly states that it does not support separation of duty (SoD) constraints [2], although an earlier draft of the profile [1] did mention SoD, but it was removed in the final release. To the best of the author's knowledge, currently there is no published work that aims to extend the XACML language to support authorisation policies for business processes.

This paper proposes BP-XACML, an extension to XACML to express authorisation policies for business processes. The proposed extension builds on the RBAC profile to support the notion of tasks and task instances, to support instance level restrictions, and separation of duty (SoD) constraints. The paper identifies the features that the language should support, defines the needed extension, and describes the new XACML profile, BP-XACML. The paper introduces a new function called 'performers list' along with new authorities to support the history-based instance-level restrictions. It also proposes a new policy set to support tasks, and a new attribute to recognise the task instances. It introduces new conditions and functions to support SoD. Figure 1 shows the complete framework of BP-XACML showing all 'policy-sets' and authorities. Elements shaded in white are from the XACML standard. Those with dotted background are added in the RBAC-XACML profile, while the ones with dark background are introduced in this paper (BP-XACML).

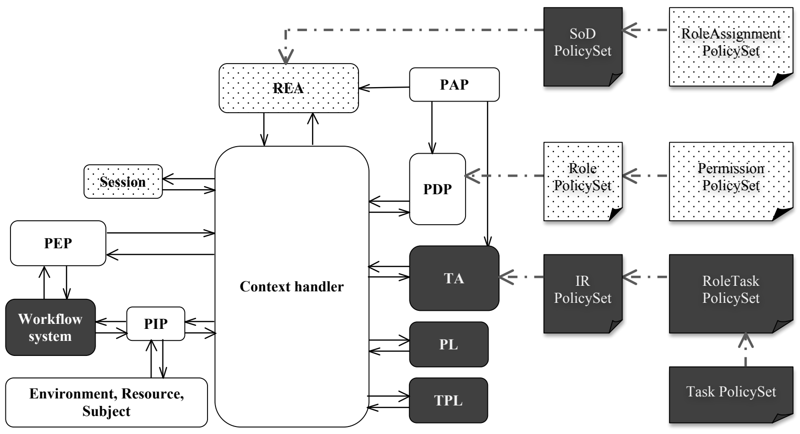


Fig. 1: BP-XACML Authorities and policy sets

The rest of the paper is organized as follows. Section 2 provides an example scenario to illustrate the need for a new policy language. Section 3 discusses the characteristics of the language. The structure of the policy language is explained in section 4. Section 5 discusses the policy model. The language semantics are explained in section 6. Section 7 provides a review of related works. Finally, section 8 concludes the paper.

2 Example Scenario

In order to illustrate the various policy language requirements for an access control system for business processes, this section will consider the access control policies for a hypothetical business process that runs across a number of systems. The example scenario is of a process of fixing a pump malfunction in an air-conditioning system in a high-security facility such as an airport.

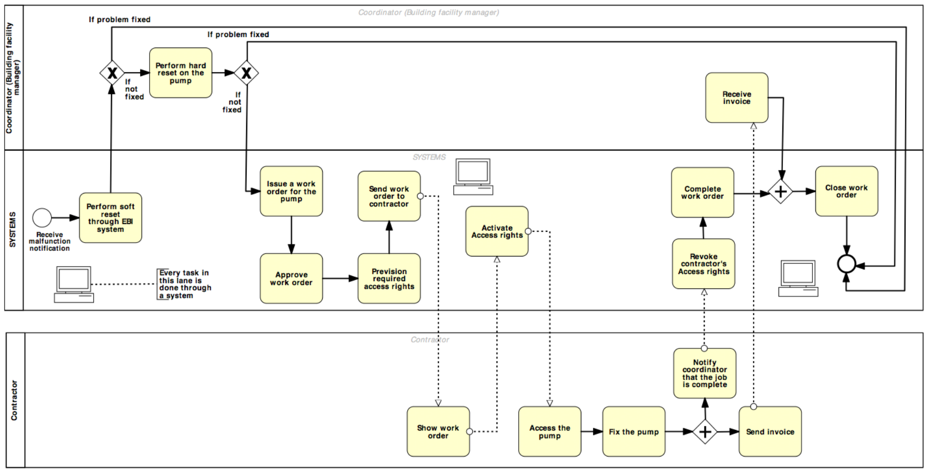

Coordinator (Building facility manager)
Coordinator (Building facility manager)
If problem fixed
If problem fixed
If not fixed
Perform hard reset on the pump
If not fixed
Receive invoice
SYSTEMS
SYSTEMS
Receive malfunction notification
Perform soft reset through EBI system
Every task in this lane is done through a system
Issue a work order for the pump
Approve work order
Send work order to contractor
Prevision required access rights
Activate Access rights
Complete work order
Revoke contractor's Access rights
Close work order
Contractor
Contractor
Show work order
Access the pump
Fix the pump
Notify coordinator that the job is complete
Send invoice

Fig. 2: Business process model for fixing pump malfunction

As can be seen in Figure 2, no one can perform a 'soft reset' on the pump unless a malfunction notification was received, and it has to be done by someone with the role 'coordinator'. If both 'soft reset' and 'hard reset' fail to solve the problem, a work order is issued. Only users with the role 'coordinator' can issue the work order. The approval of that work order should be done by a different user with the role 'Manager'. A user with the role 'contractor' needs to show the work order to gain access to the pump room. Once the issue has been resolved the user who fixed the problem will notify the user who issued the work order. This notification will result in revoking the access rights granted to the coordinator. The user who created the work order is the only one allowed to close it, and will only be able to do so after receiving the notification and the invoice.

The policy language must support the definition of policies that reflect these access control restrictions and conditions. Some of these access control policies cannot be expressed directly in XACML.

3 Business process authorisation policy requirements

With its focus on tasks and their controlled, sequenced execution, an authorisation model for business processes introduces a range of capabilities not found in the standard RBAC model [15]. Similarly, an authorisation policy language for business processes requires more than what RBAC-XACML currently provides. This section will identify the important concepts and constraints that such a policy language should be able to express by describing the significant functionalities that business process authorisation models support.

Role-Based access control (RBAC) [5] is a widely used authorisation concept. It assigns access privileges to 'roles' instead of directly assigning them to users, which reduces management overhead [13]. Users indirectly acquire permissions through their membership of roles. RBAC is an important concept to be supported in business process authorisation [7]. Support for RBAC among the published business process authorisation models including [16] and [11] is widespread. For a policy language to be able to support the RBAC concept it should support the notion of user, role, and permission. Moreover, the policy language should have the ability to represent Separation of Duties (SoD) constraints to assist in preventing fraud [5].

Tasks are a fundamental concept in business process management. They are the building blocks of business processes, so business process authorisation models such as [11] and [16] typically focus on extending RBAC with the notion of tasks. Reflecting this, authorisation policy language for business processes should support the notion of 'tasks' as a group of permissions. The example in Section 2 shows that requests are to perform a 'task', rather than to acquire a permission. So, it is important to be able to deal with tasks as well as permissions.

An important functionality that is supported by the more expressive business process authorisation models such as [16] is history-based restrictions between tasks on the instance-level, which we will refer to as 'Instance-level Restrictions (IR)'. These are restrictions that apply only within a unique execution instance of a process [16]. For example, consider the policy from section 2, which states that the person who closes a 'work order' must be the same user who issued it. So, a user will only be allowed to close the work order if he previously performed the issue function within the same process instance. This is an example of Binding of Duty (BoD) at the instance level. Separation of duty (SoD) on an instance-level requires that two tasks within an execution instance of a process, be performed by different users.

A first step to support instance-level restrictions is to be able to distinguish between instances. Some authorisation models such as [16] and [17] support the notion of 'task instance', which allows different execution instances to be distinguished. So, a policy language should be able to represent a 'task instance'. The language should also have the ability to represent the 'instance-level restrictions' themselves by specifying a separation or binding of duty relation between two tasks. Moreover, in order to enforce this condition (an instance-level restriction) the language should have the ability to retrieve history based information on an instance level [17], such as, who issued a specific 'work order'.

The key concepts that an authorisation policy language for business processes should be able to represent are: Users, Roles, Operations, Tasks, Task instance, Instance-level restrictions, and SoD. The RBAC profile of XACML supports representation of the first three characteristics. The proposed policy language extends RBAC-XACML to provide support for representation of tasks, task instances, SoD, and instance-level restrictions.

4 BP-XACML: Policy Structure

BP-XACML is based on XACML which implements rule-based access control [12]. An authorisation policy may contain multiple authorisation rules (AR), which are the basic building blocks for stating authorisation restrictions. Each AR consists of four elements: Subject, Object, Action, and Condition, the evaluation of which results in a Allow or Deny decision. $AR = \{S, O, A, C\} \rightarrow \{Allow, Deny\}$.

Action (A) is implementation specific. Condition (C) is a boolean expression that is evaluated based on the value of variables determined at run time as either true or false. Conditions can be used to represent complex constraints. The rule has its specified effect (allow or deny) if the condition evaluates as true. Rules are grouped together in 'policies', which may contain a target that limits the applicability of rules to requests which match the target's subject, object and action [9]. Policies also specify a rule-combining algorithm, which resolves potential conflicts when more than one rule is applicable [9]. Policies can be grouped together in a 'policySet' that also contains a target, and a policy-combining algorithm. It may also contain other policy sets included by reference[9].

4.1 Request and Decision

An XACML request message is sent to the PDP when a user tries to access a controlled resource. The PDP identifies matching policies and evaluates the request against them to arrive at an authorisation decision. The Request (RQ) is in the form of {S,O,A}. In BP-XACML there are three types of resource whose related policies are defined in three different policy sets (see Section 4.2). In the context of the request, the interpretation of S, O and A is different for each type. Because of this, each type of request is processed by a different authority. Firstly, in the case of a user requesting to perform a workflow task, the subject (S), will be the identifier for the specific user making the request. The object (O), is the task that the user wants to perform. Since a task explicitly defines its associated permissions (object, action pair), they are not separately identified in the request. The Action (A), is simply the request to 'perform'. In the second case a user requests access to a resource object that is not a workflow 'task', for example, to access the 'pump room'. In this case, O will be the 'pump room', and A will be 'access'. The third type of request is to activate a role, for example, 'Adam' wants to activate the role 'coordinator'. In such requests, S is 'Adam', O is the role 'coordinator', and A is 'activate'. The decision (DS) will be either {Allow}, {Deny}, or {Not applicable} if no matching policies are found.

4.2 Policy Sets

In XACML 'Policy sets' are used to group related policies, which contain, groups of related access control rules. The RBAC profile of XACML predefines some policy sets and makes use of them to determine the access control decision. For example, a `RoleAssignment<PolicySet>` will include all policies and rules related to role assignment. In this extended profile we will make use of these policy sets and introduce new policy sets.

BP-XACML includes seven types of access control policy sets. The PDP will use two policy sets, the `Role<PolicySet>`, and the `Permission<PolicySet>` to make decisions on the requests directed to the PDP. The `task<PolicySet>`, and `RoleTask<PolicySet>` are used to state the tasks that a role is allowed to perform. `IR<PolicySet>` is used to state instance-level restrictions. The `SoD <PolicySet>`, and `RoleAssignment<PolicySet>`, are used for stating and activating roles of each user. The `Role<PolicySet>`, `Permission<PolicySet>`, and `RoleAssignment<PolicySet>` are adopted from the XACML RBAC profile [9]. The rest of the `PolicySets` are newly introduced in this paper, and will be discussed in this section. The mechanism and application of these policy sets will be discussed in more detail in section 6. Figure 3 shows the relation between IRPS, RTPS, and TPS, and gives a summary of the structure of each policy set.

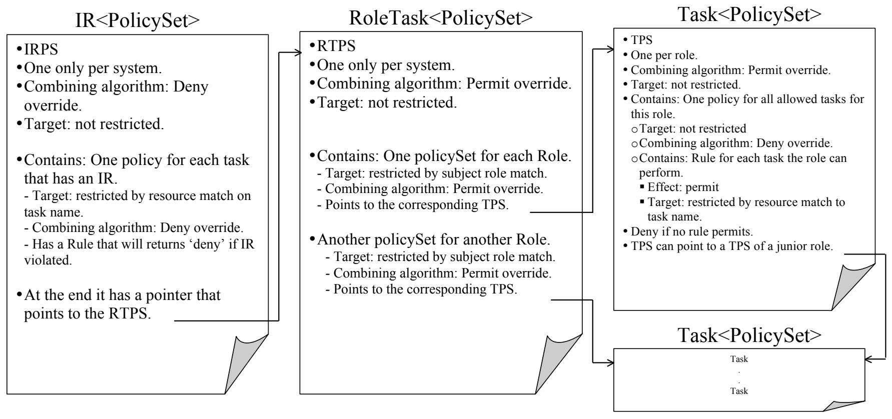

IR<PolicySet>
•IRPS
•One only per system.
•Combining algorithm: Deny override.
•Target: not restricted.
•Contains: One policy for each task that has an IR.
- Target: restricted by resource match on task name.
- Combining algorithm: Deny override.
- Has a Rule that will returns 'deny' if IR violated.
•At the end it has a pointer that points to the RTPS.
RoleTask<PolicySet>
•RTPS
•One only per system.
•Combining algorithm: Permit override.
•Target: not restricted.
•Contains: One policySet for each Role.
- Target: restricted by subject role match.
- Combining algorithm: Permit override.
- Points to the corresponding TPS.
•Another policySet for another Role.
- Target: restricted by subject role match.
- Combining algorithm: Permit override.
- Points to the corresponding TPS.
Task<PolicySet>
• TPS
• One per role.
• Combining algorithm: Permit override.
• Target: not restricted.
• Contains: One policy for all allowed tasks for this role.
○Target: not restricted
○Combining algorithm: Deny override.
○Contains: Rule for each task the role can perform.
▪ Effect: permit
▪ Target: restricted by resource match to task name.
• Deny if no rule permits.
• TPS can point to a TPS of a junior role.
Task<PolicySet>
Task
.
Task

Fig. 3: New Policy Sets

`Task<PolicySet>` (TPS) is a `<PolicySet>` that contains the actual tasks authorised for a given role. The `<Target>` element of a TPS, should not limit the applicability of the `<PolicySet>` as the `IR<PolicySet>` and the `RoleTask<Policy Set>` restrict access (see Figure 3). To achieve role hierarchy, a TPS associated with a senior role may also contain references to TPSs associated with junior roles, thereby allowing the senior role to inherit all access to tasks associated with the junior roles. In a TPS, (S) refers to user's role, and (O) refers to task.

`RoleTask<PolicySet>` (RTPS) is a `<PolicySet>` that contains the Roles, and for each role it points to the corresponding `Task<PolicySet>` (i.e. the TPS is included in the RTPS by reference). The `<Target>` element of a RTPS, should not restrict the applicability of the `<PolicySet>`, but the `<PolicySet>s` for each role (that are included within the RTPS) has a target restricting applicability for the specified role only. The RTPS is used to achieve role hierarchy. In the RTPS Subject (S) refers to the user's role, and Object (O) refers to the task.

`IR<PolicySet>` (IRPS) is a `<PolicySet>` that describes instance-level restrictions. The RTPS and TPS can only be reached through the IRPS where they are included by reference. The Task Authority will first access this policy set to check if there is no violation of an IR constraint, then it will be pointed via the `RoleTask<PolicySet>` to the related `Task<PolicySet>`. Section 6.3 explains this in more detail. In the IRPS, the subject (S) is not restricted because IR constraints deal with task instances regardless of the subject. The object (O) is the task, and the action (A) is 'perform'. For example, no user is allowed to perform both 'issue work order' and 'approve work order'. In this case the IRPS will have both tasks in one policy making sure that the user does not perform both for the same instance.

`SoD<PolicySet>` (SoDPS) is a `<PolicySet>` that describes separation of duties constraints. It restricts access to the `RoleAssignment<PolicySet>`. The Role Enablement Authority will first access this policy set to check if there is no violation of SoD constraint, then it will be pointed to the `RoleAssignment <PolicySet>`. In a SoDPS, the subject (S) is not restricted because SoD constraints deal with roles regardless of the subject. The object (O) is a role. Each policy in the `SoD<PolicySet>` includes a pair of conflicting roles.

4.3 Conditions

A condition is specified as a Boolean expression that is evaluated at runtime. There are two main types of policy conditions of interest in specifying access restrictions in this policy model. The first one is dynamic Separation of Duties (SoD) conditions on role level. The other one is instance-level restrictions (IR).

A dynamic SoD condition is an expression that can be evaluated for user-role relation by testing the current active roles for this user. It is used to prevent a user from activating two conflicting roles by the same user at the same time. They are defined as policies in the `SoD<PolicySet>`. $SoD : (Role1, Role2)$. The Role enablement authority (REA) will be able to know the SoD restriction before enabling a role. It will use the 'session' component to check the current status of role enablement for a user requesting role activation. Session maintains the list of active roles for each user.

An 'instance-level restriction' (IR) condition is an expression that can be evaluated to check the relation between two objects within the same instance. They are defined as policies within the `IR<PolicySet>`. $IR : (\{Task1, Task2\}, type)$. IR is a type of SoD (or BoD) restriction on a task level that only applies within the same instance. It makes sure that the restriction is met within the same instance. For example, the task of 'closing work order' should have a restriction that it can be only done by the same user who performed 'issue work order' for this same instance.

5 BP-XACML: Policy Model

BP-XACML is designed to be backward-compatible with the RBAC-XACML policy structure. This has an important benefit: It means that role-based authorisation policies can be defined and managed independently of the workflow authorisation system. These policies will still be applied when a user requires access to a controlled resource to execute an instance of a business process. This design approach introduces some complexity, most notably in the inclusion of the Task Authority as a separate PDP to authorise task activation. But it is necessary because the role-based authorisation policies that control access to an organisation's valuable resources, (e.g. customer records, financial records etc.) are typically created and maintained independently of the business processes. They will often exist before a workflow is created that uses the controlled resources. These policies still need to be applied in the context of the workflow but we argue that this should not be done by a parallel and duplicated workflow authorisation system, since this would be inefficient and difficult to maintain. Accordingly, we have designed the BP-XACML policy structure to work with an existing RBAC-XACML policy set. This results in an integrated system which can handle both workflow and non-workflow requests from a single (and therefore consistent) set of policies.

After explaining the structure of the BP-XACML policy language, this section will describe the BP-XACML policy model. It shows how access decisions are made using the defined 'policy sets', describes the needed authorities and repositories, and explains the policy model framework.

5.1 Authorities and Repositories

In this policy model we are introducing a new authority and two repositories that are needed to fulfill the requirements. They are the Task authority (TA), Performers list (PL), and Task-permissions list (TPL). We also include the 'Role Enablement Authority (REA)', and 'session' concept from the XACML RBAC profile, and we elaborate on how to use them, as the RBAC profile does not provide these details.

It might appear unnecessarily complex to add these authorities, where some of them are essentially a specialised PDP. One might argue that one PDP should be enough. However, the RBAC-XACML profile [2], which is proposed by OASIS, adopts this approach in introducing the REA. The RBAC profile shows that role enablement should be out of the scope of the PDP, that is why REA was introduced to be responsible for role assignment and enablement [2]. The justification for having a specialised PDP for role enablement can be understood by looking at the basic request concept of XACML, where each request contains a subject and an object. The PDP is designed to deal with one interpretation of each aspect of the request (subject, object, action). For example, in RBAC-XACML if the PDP receives a request it will look at the subject as the user's role, and the object as the resource that the user wants to perform an action on. Adam, who is a manager, wants to read file2. The PDP will use the permission policy to determine if managers are allowed to read file2. A request to activate a role has the subject as the user ID, and the object as the role that needs to be activated. That is why it was necessary to have a specialised PDP called REA. This REA is designed to look at the subject as the user's ID, and the object is the role that the user wants to activate. Therefore it will be able to deal with activation requests.

BP-XACML deals with three different type of requests, where each type of request has a different interpretation of subject and object. The change in subject and object interpretation, makes it necessary to have a different authority to deal with each different type of request. The request to perform a task has the user's role as the subject and the task ID as the object. Therefore, the authorisation of task performance is out of the scope of the PDP. TA is introduced in this model to be the specialised PDP responsible for making a task performance decisions. It permits backward compatibility with the role and permission policy sets defined in RBAC-XACML. TA deals with requests on the basis that the subject is the user's role and the object as the task ID. Therefore, it is able to deal with requests to perform a task.

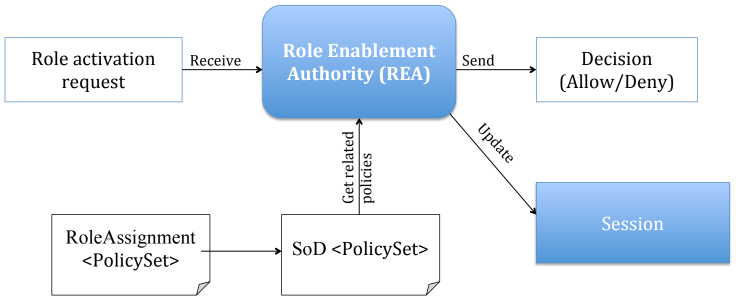


Fig. 4: Role Enablement Authority

Role Enablement Authority (REA) uses the `SoD<PolicySet>`, and `Role Assignment<PolicySet>` to either allow or deny activation of a specific role for a specific user.
Session provides a quarryable service, which maintains and continuously refreshes the state of user role enablement relations.

Figure 4 shows how REA uses the `RoleAssignment<PolicySet>` to know if the user is allowed to enable a role or not. Before reaching the `RoleAssignment <PolicySet>`, REA checks the `SoD<PolicySet>` to check if a SoD policy is available for this role. If such a policy exists, REA needs to know the status of the user's activated roles. This information can be retrieved from the user's session. The information allows REA to evaluate if the condition is met or not. Based on that REA will send the final decision on the role enablement request.

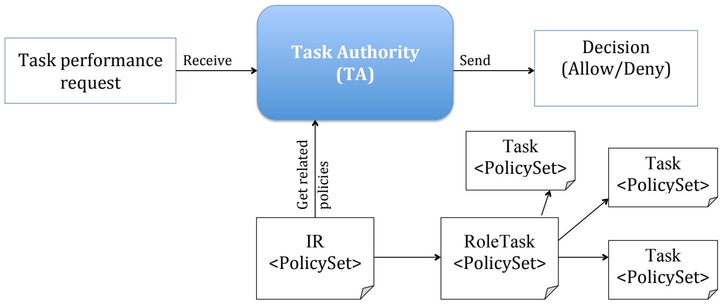


Fig. 5: Task Authority

Task Authority (TA) uses the `IR<PolicySet>`, and `Task<Policy Set>` to either allow or deny a user's request to perform a specific task.

Performers list is introduced to provide a quarryable service to report the user that performed a completed task instance. It maintains the state of user 'task instance' performance relations, and continuously refreshes the state.

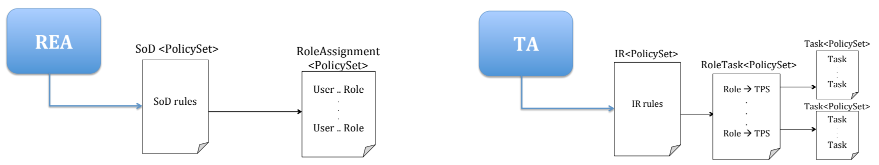
As can be seen in Figure 5, TA uses `Task<PolicySet>` to check if the role is allowed to perform the task or not. Before checking the `Task<PolicySet>`, TA first checks the `IR<PolicySet>` to determine if there are any instance-level restrictions on such task. If a restriction is found TA needs to retrieve extra information to assess the restriction. This information can be found through the 'Performers list (PL)'. It is important to be able to check IR restrictions. In order for an IR condition to be evaluated, it is necessary to know the performer of a given task instance.

Task-permissions list (TPL) is a new proposal. It maintains the state of task-permission relations. TPL is used by the context handler (CH) to determine the permissions associated with each task. TPL provides a list of permissions for each task, where a permission is an action on a resource.

5.2 Access Control

BP-XACML model controls three types of access control requests: activating a role (controlled by the REA), performing a task (controlled by the TA), and performing action on a resource (controlled by the PDP). The context handler is responsible for forwarding the request to the corresponding authority depending on its type.

A single `SoD<PolicySet>` is defined in the system, which contains all SoD restrictions. The policy set itself is not limited (i.e. the target is empty and therefore does not restrict the applicability of the included policies), but each policy is limited to a specific role. It contains a single `<PolicySetIdReference>` element, which refers to the `RoleAssignment<PolicySet>` (RAPS). There is a single RAPS in a system, which contains the information on whether to allow or deny the role activation for a specific user.



(a) REA can only access SoD PolicySet (b) TA can only access IR PolicySet

Fig. 6: New PDPs access

As shown in Figure 6-a, the RAPS must be stored in a policy repository in such a way that it can never be reached directly by the REA; RAPS must be reachable only through the SoDPS. This is because, in order to support separation of duties, it is important that the SoD policies are satisfied before reaching the RAPS. For REA to achieve a decision on role activation request it accesses the SoDPS and only check the RAPS if the SoD rules were satisfied.

A single `IR<PolicySet>` is defined in the system, which contains all IR restrictions. The policy set itself is not limited, as the policy set target is empty, but each policy is limited to a specific task. The policy set contains a single `<PolicySetIdReference>` element, which refers to the `RoleTask<PolicySet>`. For the system there is a single `RoleTask<Policy Set>`, which contains a `<PolicySet>` for each role, which points to the corresponding `Task<PolicySet>`. A user will be authorised to perform a task if there is a permit rule for the task in the TPS for a role that the user has active.

As shown in Figure 6-b, TPS instances, and the RTPS must be stored in a policy repository in such a way that they can never be reached directly by the TA. RTPS must be reachable only through the IRPS. This is because, in order to support 'role hierarchy', the TPS depends on the RTPS to ensure that only subjects holding the corresponding role attribute (or senior role) will gain access to perform tasks in the given TPS. For TA to achieve a decision on a request to 'perform a task', it first must access the IRPS, and check if there are any related IR policies. IRPS will then point to RTPS. Using the user's role, RTPS points to the corresponding TPS, which contains rules stating whether this role is allowed to perform a task or not. These `<PolicySet>` relationships and constrains are summarised in Figure 3.

5.3 Policy Framework

Figure 7 shows the complete BP-XACML framework without the 'policy sets'. It includes all the authorities, components, and repositories. As explained earlier, the policy model should be unified and deal with all authorisation requests, regardless of whether or not they arise in the context of a workflow. For this reason, the BP-XACML policy model is designed to deal with several types of requests. It could be either a request to activate a role for a user, a request to perform a task, or a standard RBAC request to perform an action on a resource. In this section we will discuss each type of request by it self, and show how it is handled within the framework.

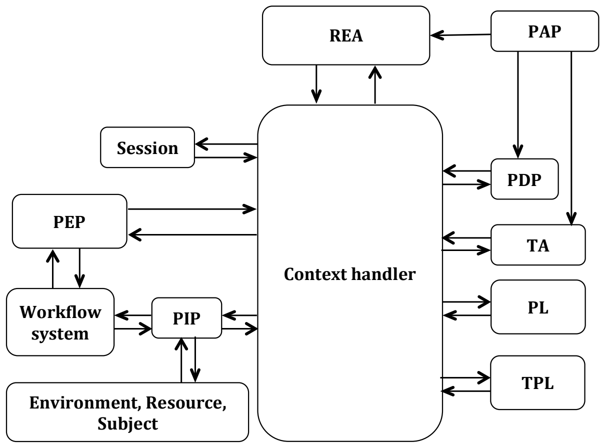


Fig. 7: BP-XACML framework

The role activation requests are directed to the 'role enablement authority (REA)' by the context handler. The REA will use the`SoD<PolicySet>` to check for any SoD restrictions on this role. Then it will point to the RAPS to decide if this user is allowed to activate this role or not. If there was an SoD constraint on the role, REA will require extra information. For example, if there is a SoD condition on activating this role, REA needs to make sure that activating this role will not breach the SoD condition. Information about the activated roles of this user will be obtained from the 'session' through the CH. Figure 8 shows the steps related to this type of request.

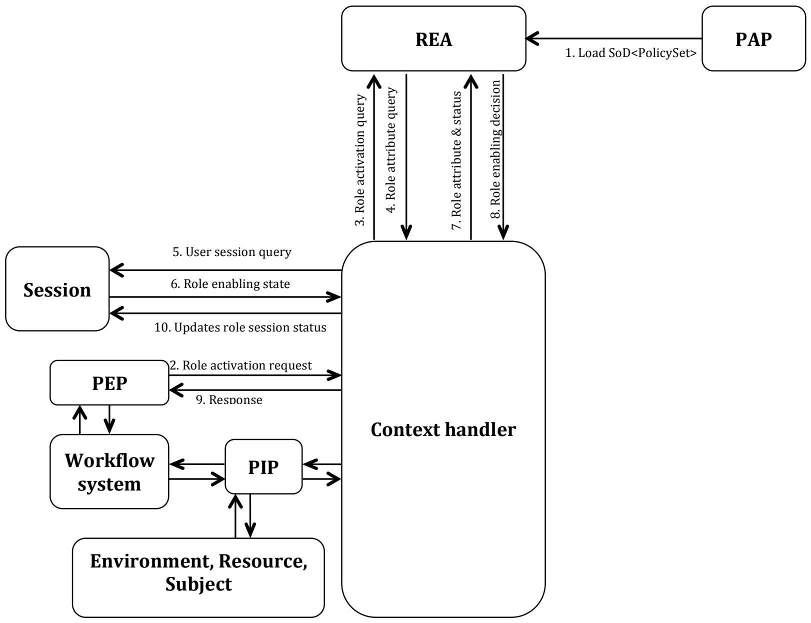


Fig. 8: Activating a role request

The standard RBAC request is a request to perform an action on a resource, where the resource is not a role or a task (e.g. read a file). This type of request is directed to the PDP, and will be handled exactly how access requests are handled in the RBAC-XACML profile [9] using the `Role<PolicySet>` and `permission<PolicySet>`. For more information please refer to [9]. In BP-XACML, a request to perform a task, will produce a set of one or more requests of this type (standard request).

If the request was to perform a specific task, the context handler will forward the requests to the TA. The TA will use the IRPS to check if there are any IR restrictions on this task. Violation of an IR results in a deny decision. Then it will use the user's role to identify the proper TPS through the RTPS. TPS identifies the tasks that this user is allowed to perform based on the role activated at the request time. If an IR is restricting the assigning of a task, the TA will obtain extra information from the 'performers list (PL)' through the CH to evaluate the IR condition. If the TA allows the user to perform the task, CH will use

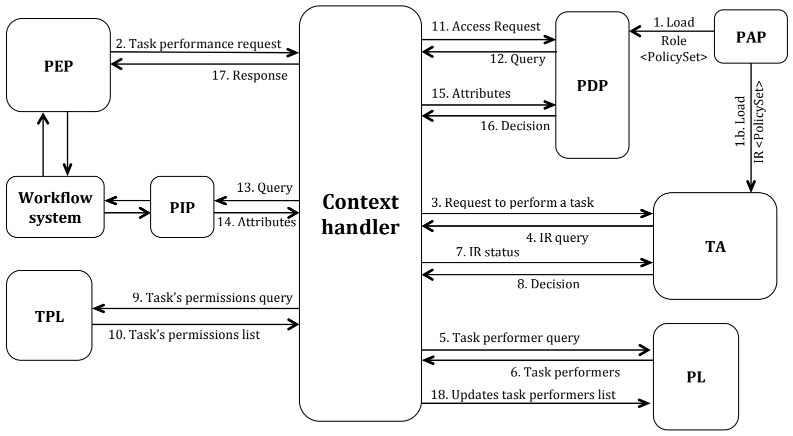


Fig. 9: A request to perform a task

the Task Permissions List (TPL) to retrieve all permissions associated with the task. Each permission is a pair of an action and a resource. CH will create a request for each permission, with requests containing the user's role, action, and resource. These requests will be sent to the PDP and dealt with as standard requests to perform an action on a resource. PDP will send back each decision individually. CH will combine the decisions, where deny over rides. So, if one request was denied, the whole request to perform the task will be denied. If all requests were allowed, the CH will then send back to the PEP that this user is allowed to perform the task. Figure 9 shows the steps for this type of request.

6 BP-XACML: Policy Semantics

In this section, we will refine the previously described policy structure with specific data and language representations. Users, roles, operations and permission are all part of the RBAC profile of XACML. In this paper we adopt these entities and the way they are expressed from the RBAC-XACML [2]. Refer to [2] for information on the representation of users, roles, operation, and permissions.

6.1 Task and Task instances

Task and task instances are new features that are not supported in RBAC-XACML. In BP-XACML tasks are expressed as an XACML Resource. Listing 1 shows an example `Task<PolicySet>` showing the task as a resource.

```
<PolicySet ... PolicySetId="TPS:coordinator:role" PolicyCombiningAlgId="&policy-combine;permit-overrides">
 <Target>  <Subjects><AnySubject/></Subjects>  <Resources><AnyResource/></Resources>  <Actions><AnyAction
       /></Actions>  </Target>
<Policy PolicyId="Allowed tasks" RuleCombiningAlgId="&policy-combine;permit-overrides">
<Target>
 <Target>  <Subjects><AnySubject/></Subjects>  <Resources><AnyResource/></Resources>  <Actions><AnyAction
       /></Actions>  </Target>
 <Rule Effect="Permit" RuleId="issue:work:order:task">
 <Subjects><AnySubject/></Subjects>
 <Resources>
  <ResourceMatch MatchId="&function;string-match">
   <AttributeValue DataType="&xml;string"> issue work order</AttributeValue>
   <ResourceAttributeDesignator AttributeId="&resource;resource-id" DataType="&xml;string"/>
  </ResourceMatch> </Resources>
 <Actions>
  <ActionMatch MatchId="&function;string-match">
  <AttributeValue DataType="&xml;string">perform</AttributeValue>
   <ActionAttributeDesignator AttributeId="&action;action-id" DataType="&xml;string"/>
  </ActionMatch>  </Actions>  </Target> </Rule>
 <Rule Effect="Deny" RuleId="DenyRule"/>
 </Policy>  </PolicySet>
```

Listing 1: Task Policy set

In listing 1 task was represented as an object in the policy, because TPSs are linking the user's role to the task, so the task is the object. To be able to represent 'Task instance' a new object attribute is introduced, it is called 'instance'. It is similar to the 'role' attribute from the RBAC-XACML profile. In section 6.3 an example listing showing instance-level restriction will show how to make use of the new attribute 'instance".

As it can be seen the `Task<PolicySet>` will include a policy for each task the role is allowed to perform. The example includes a policy for the task 'issue work order' as a part of the policy set. The policy says if someone wants to perform the action 'perform' on the object 'task: issue work order' they will be allowed. As can be seen in the listing, the policy set target is not limiting the applicability of the policy set. `RoleTask<PolicySet>` will limit the applicability to users with the role 'coordinator' and then point to this policySet. Listing 2 is an example `RoleTask<PolicySet>` for the role 'coordinator'.

```
<PolicySet ... PolicySetId="RTPS" PolicyCombiningAlgId="&policy-combine;permit-overrides">
 <Target> <Subjects><AnySubject/></Subjects>  <Resources><AnyResource/></Resources> <Actions><AnyAction/></
       Actions> </Target>
 <Policy PolicyId="Coordinator:Role" RuleCombiningAlgId="&rule-combine;permit-overrides">
 <Target>
  <Subjects>
   <SubjectMatch MatchId="&function:any-of">
   <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function;string-equal"
    <AttributeValue DataType="&xml;string">coordinator</AttributeValue>
    <SubjectAttributeDesignator AttributeId="urn:someapp:attributes:role" DataType="&xml;string"/>
   </Apply>  </SubjectMatch>  </Subjects>
  <Resources><AnyResource/></Resources>
  <Actions><AnyAction/></Actions>
 </Target>
 <!-- Use tasks associated with the "coordinator" role -->
 <PolicySetIdReference> TPS:coordinator:role </PolicySetIdReference>
 </policy> </PolicySet>
```

Listing 2: The Role Task Policy Set

6.2 SoD on a role level

In BP-XACML SoD is expressed as policies in the `SoD<PolicySet>`. SoD refers to the dynamic role level separation of duties, which is used to make sure that no one user activates two conflicting roles at the same time. Listing 3 shows an example SoD policy set. The policy set includes policies stating conflicting roles. For example, the policy set in listing 3 includes a policy stating that in order to activate the role 'coordinator, the role 'manager' should not be in the activated roles of the same user.

The function 'Session' is a new function that helps to check that the given role is not available in the activated roles of the given user. This function takes one argument of data-type "..#string", which is the user's ID. It returns a list of all roles currently activated for this user. Then the predefined function "any-of" will compare the given string with the list retrieved by the session function. If the role was found the function will return the result 'true', and if it was not found, it will return 'false'. If the condition was true the rule will return 'deny' and the request will be denied. If the condition returns false, the rule will not do anything and continue to the `RoleAssignment<PolicySet>`.

```
<PolicySet ... PolicySetId="SoD" PolicyCombiningAlgId="&policy-combine;deny-overrides">
 <Target>  <Subjects><AnySubject/></Subjects>  <Resources><AnyResource/></Resources>  <Actions><AnyAction
      /></Actions> </Target>
<Policy PolicyId="Coordinator:Role" RuleCombiningAlgId="&rule-combine;deny-overrides">
 <Target>  <Subjects><AnySubject/></Subjects>
  <Resources>
      <ResourceMatch MatchId="&function;string-match">
    <AttributeValue DataType="&xml;string"> Coordinator </AttributeValue>
    <ResourceAttributeDesignator AttributeId="&resource;resource-id" DataType="&xml;string"/>
   </ResourceMatch> </Resources>
  <Actions> activate </Actions>
 </Target>
<Rule RuleId="role:manager:not:active" Effect="Deny">
 <Target>  <Subjects><AnySubject/></Subjects>  <Resources><AnyResource/></Resources>  <Actions><AnyAction
      /></Actions>  </Target>
  <Condition FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-of">
    <AttributeValue DataType=&xml;string"> manager </AttributeValue>
   <Apply FunctionId="http://localhost/BPXACML/function#function;Session">
   <SubjectAttributeDesignator AttributeId="urn:someapp:attributes:role" DataType="&xml;string"/>
   </Apply> </Condition>   </Rule>  </Policy>
 <PolicySetIdReference> Role:Assignment </PolicySetIdReference>
</PolicySet>
```

Listing 3: SoD policy set example

Listing 4 shows an example `RoleAssignment<PolicySet>`. The `<PolicySet>` contains a policy for each user, which contains rules for each role the user can activate. The policy set in listing 4 includes an example policy for the user Adam, which includes an example rule for activating the role 'coordinator'.

```
<PolicySet ... PolicySetId="Role:Assignment" PolicyCombiningAlgId="&policy-combine;deny-overrides">
 <Target>  <Subjects><AnySubject/></Subjects>  <Resources><AnyResource/></Resources>  <Actions><AnyAction
      /></Actions>  </Target>
<Policy PolicyId="Roles:For:user:Adam" RuleCombiningAlgId="&rule-combine;deny-overrides">
 <Target>
  <Subjects>
   <SubjectMatch MatchId="&function;string-match">
    <AttributeValue DataType="&xml;string"> Adam </AttributeValue>
    <SubjectAttributeDesignator AttributeId="&subject;subject-id" DataType="&xml;string"/>
   </SubjectMatch> </Subjects>
  <Resources><AnyResource/></Resources>
  <Actions><AnyAction/></Actions>
 </Target>
<Rule RuleId="Permission:to:activate:coordinator:role" Effect="Permit">
 <Target>
  <Subjects><AnySubject/></Subjects>
  <Resources>
   <ResourceMatch MatchId="&function;string-equal">
    <AttributeValue DataType="&xml;string"> Coordinator </AttributeValue>
    <ResourcesAttributeDesignator AttributeId="&resource;resource-id" DataType="&xml;string"/>
   </ResourcetMatch>  </Resources>
  <Actions> Activate </Actions>  </Target>  </Rule>  </Policy> </PolicySet>
```

Listing 4: Role Assignment policy set example

6.3 Instance-level Restrictions (IR)

Instance-level restrictions (IR) are used to fulfill the need to apply history based restrictions within the same instance. For example, the scenario states that the user who close the 'work order' should be the same user who issued it. So, for the same 'work order' (same instance), the user to perform 'close work order' must be the same user who performed 'issue work order'. IR restrictions are written as policies in the IR policy set.

Listing 5 is an example `IR<PolicySet>` that includes instance-level restriction using the BP-XACML language. The IR policy set has a policy for the task ‘close work order’. The policy has a rule stating that the user must be the same user who issued the work order for the same instance.

The function ‘PL’ is a new function that retrieves the performers list of a specific task for a specific instance. This function takes two arguments of datatype ”..#string”, which are a task name and an instance number. It returns a list of all users who performed the task for this instance. Then the predefined function “any-of” will compare the given string, which is the username of the user who requests to perform the task, with the list retrieved by the PL function. The function “any-of” will return ‘true’ if the user was in the performers list, and it will return ‘false’ if it was not found in the performers list. If it was a SoD-IR then this condition will be satisfied and the user will be denied if he was part of the list. Because it is a binding of duties constraint in this case, we want the rule to deny only if the user was not found in the list (i.e. the function ‘any-of’ returned false), and permit it if he was in the list (i.e. the function ‘any-of’ returned true). For this reason the function “not” has been added to reverse the output of the function.

```
<PolicySet ... PolicySetId="IR" PolicyCombiningAlgId="&policy-combine;deny-overrides">
 <Target>  <Subjects><AnySubject/></Subjects>  <Resources><AnyResource/></Resources>  <Actions><AnyAction
       /></Actions>  </Target>
<Policy PolicyId="close:work:order:task RuleCombiningAlgId="&rule-combine;deny-overrides""
<Target>  <Subjects><AnySubject/></Subjects>
        <Resources>
          <ResourceMatch MatchId="&function;string-match">
             <AttributeValue DataType="&xml;string"> close work order</AttributeValue>
             <ResourceAttributeDesignator AttributeId="&resource;resource-id" DataType="&xml;string"/>
          </ResourceMatch>      </Resources>
    <Actions>
     <ActionMatch MatchId="&function;string-match">
      <AttributeValue DataType="&xml;string"> perform </AttributeValue>
      <ActionAttributeDesignator AttributeId="&action;action-id" DataType="&xml;string"/>
     </ActionMatch> </Actions> </Target>
 <Rule RuleId="Must:be:who:issued:work:order" Effect="Deny">
 <Target> <Subjects><AnySubject/></Subjects>  <Resources><AnyResource/></Resources> <Actions><AnyAction/></
       Actions> </Target>
    <Condition FunctionId=urn:oasis:names:tc:xacml:1.0:function:not>
       <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-of">
        <SubjectAttributeDesignator AttributeId="&subject;subject-id" DataType="&xml;string"/>
       <Apply FunctionId="http://localhost/BPXACML/function#function;PL">
        <AttributeValue DataType=&xml;string"> issue work order </AttributeValue>
        <ResourceAttributeDesignator AttributeId="urn:someapp:attributes:instance" DataType="&xml;string"/>
     </Apply>  </Apply> </Condition>  </Policy>
 <!-- Point to the RoleTask policy set -->
 <PolicySetIdReference> RoleTask:PolicSet </PolicySetIdReference>
</PolicySet>
```

Listing 5: Example IR Policy set

7 Related Literature

To the best of the authors’ knowledge, currently there is no published work that aims to extend XACML to support business process authorisation policies. There are several published works that extend XACML to support different models but none of them focus on ‘business processes’. For example, Wolter et. al. in [19] developed a XACML customised profile that supports RBAC concept, mandatory access control, and permission-based separation of duty policies. The work does not take into consideration the special requirements that ‘business process’ authorisation policies need such as ‘tasks’, and it does not extend XACML to support business process authorisation policies.

The work in [4], [10], [18], and [19] all focus on proposing a model transformation framework that focuses on deriving security policies from the process model, using a form of extended BPMN as a process modeling language and XACML as a policy language. These papers start by proposing a new extension to the process modeling language BPMN, and then propose a model-driven extraction of the policies based on a mapping between the new modeling language and the policy language. All four proposals are limited to the BPMN extension proposed in the corresponding paper ("seBPMN"[19], "ConstrainedBPMN"[18] "SecureBPMN"[4], and "BPMS"[10]) and they do not extend XACML. Sinha et. al. [14] also propose a method of translating security requirements into XACML, by making use of the obligation feature in XACML. It does not provide an extension to XACML. A draft version of the RBAC-XACML profile [1] proposed a policy structure to handle dynamic SoD. We use the same policy structure, but implement the SoD restriction in a different way. We also provide more details on the way it should be used.

8 Conclusion

This paper introduced BP-XACML, a new profile that extends the RBAC-XACML and enables the specification of business process authorisation policies. In addition to supporting the XACML RBAC profile, the extended language also supports the representation of tasks and tasks instance. It proposes a new policy set called `Task<PolicySet>` for the incorporation of business process tasks. BP-XACML also supports separation of duties and binding of duties constraints at the level of process instances. It supports the representation of the instance-level restrictions in a way that can be linked to the related tasks and can be evaluated. The paper proposes a new policy set `IR<PolicySet>`. The new repository 'performers list' helps in evaluating the instance-level restrictions. The new repository TPL links tasks to permissions. Finally, it supports the 'separation of duties' on the role level, making use of the 'REA' and 'sessions' to find, and evaluate the SoD restrictions.

As a future work, it is intended to do an experimental implementation to test the efficiency, feasibility and usability of this design.

References

1. A. Anderson. Xacml profile for role based access control (rbac), committee draft 01. Standard, OASIS, February 2004.
2. A. Anderson. Core and hierarchical role based access control (rbac) profile of xacml version 2.0, oasis standard. Standard, OASIS Open, February 2005.
3. V. Atluri and W. kuang Huang. An authorization model for workflows. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *European Symposium on Research in Computer Security*, volume 1146 of *Lecture Notes in Computer Science*, pages 44–64. Springer, 1996.
4. A. D. Brucker, I. Hang, G. Lückemeyer, and R. Ruparel. Securebpmn: Modeling and enforcing access control requirements in business processes. In *Proceedings of*

the 17th ACM Symposium on Access Control Models and Technologies, SACMAT '12, pages 123–126, New York, NY, USA, 2012. ACM.

5. D. Ferraiolo and D. Kuhn. Role-Based Access Control. In *15th National Computer Security Conference*, pages 554–563, October 1992.
6. Gartner. Leading in times of transition: The 2010 CIO agenda. In *Gartner EXP CIO report*, 2010.
7. M. Leitner, S. Rinderle-Ma, and J. Mangler. Aw-rbac: access control in adaptive workflow systems. In *Sixth International Conference on Availability, Reliability and Security (ARES)*, pages 27–34. IEEE, 2011.
8. A. X. Liu, F. Chen, J. Hwang, and T. Xie. Xengine: A fast and scalable xacml policy evaluation engine. In *Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '08, pages 265–276, New York, NY, USA, 2008. ACM.
9. T. Moses. Extensible access control markup language (xacml) version 2.0. oasis standard. Technical report, OASIS Open, 2005.
10. J. Mülle, S. v. Stackelberg, and K. Böhm. A security language for bpmn process models. In *Karlsruhe Reports in Informatics.* Karlsruhe, 2011.
11. S. Oh and S. Park. Task-role based access control (t-rbac): An improved access control model for enterprise environment. In M. T. Ibrahim, J. Küng, and N. Revell, editors, *DEXA*, volume 1873 of *Lecture Notes in Computer Science*, pages 264–273. Springer, 2000.
12. P. Samarati and S. C. Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 137–196, Berlin Heidelberg, 2001. Springer.
13. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role - based access control models. In *IEEE Computer*, volume 29, pages 38–47, 1996.
14. S. Sinha, S. Sinha, and B. Purkayastha. Synchronization of authorization flow with work object flow in a document production workflow using xacml and bpel. In V. Das and R. Vijaykumar, editors, *Information and Communication Technologies*, volume 101 of *Communications in Computer and Information Science*, pages 365–370. Springer Berlin Heidelberg, 2010.
15. M. Strembeck and J. Mendling. Modeling process-related rbac models with extended uml activity models. *Information & Software Technology*, 53:456–483, 2011.
16. J. Wainer, A. Kumar, and P. Barthelmess. WRBAC a work-flow security model incorporating controlled overriding of constraints. In *International Journal of Cooperative Information Systems (IJCIS)*, volume 4, pages 455–486, 2003.
17. J. Wainer, A. Kumar, and P. Barthelmess. DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Inf. Syst.*, 32(3):365–384, 2007.
18. C. Wolter, A. Schaad, and C. Meinel. Deriving xacml policies from business process models. In M. Weske, M.-S. Hacid, and C. Godart, editors, *Web Information Systems Engineering – WISE 2007 Workshops*, volume 4832 of *Lecture Notes in Computer Science*, pages 142–153. Springer Berlin Heidelberg, 2007.
19. C. Wolter, C. Weiss, and C. Meinel. An xacml extension for business process-centric access control policies. In *Policies for Distributed Systems and Networks, 2009. POLICY 2009. IEEE International Symposium on*, pages 166–169, July 2009.