XACML and Role-Based Access Control

Jason Crampton
Royal Holloway, University of London

DIMACS Workshop on Secure Web Services and e-Commerce

Programme

Examine the XACML standard and the XACML RBAC profile

- Examine the XACML implementation of role-based access control policies
- Identify any shortcomings
- Identify any omissions
- Propose some extensions and alternative approaches

Outline of talk

- Introduction to XACML
- Introduction to RBAC
- The XACML RBAC profile
- An alternative approach to RBAC using XACML
- Assigning subjects to roles
- Separation of duty

XACML

Introduction

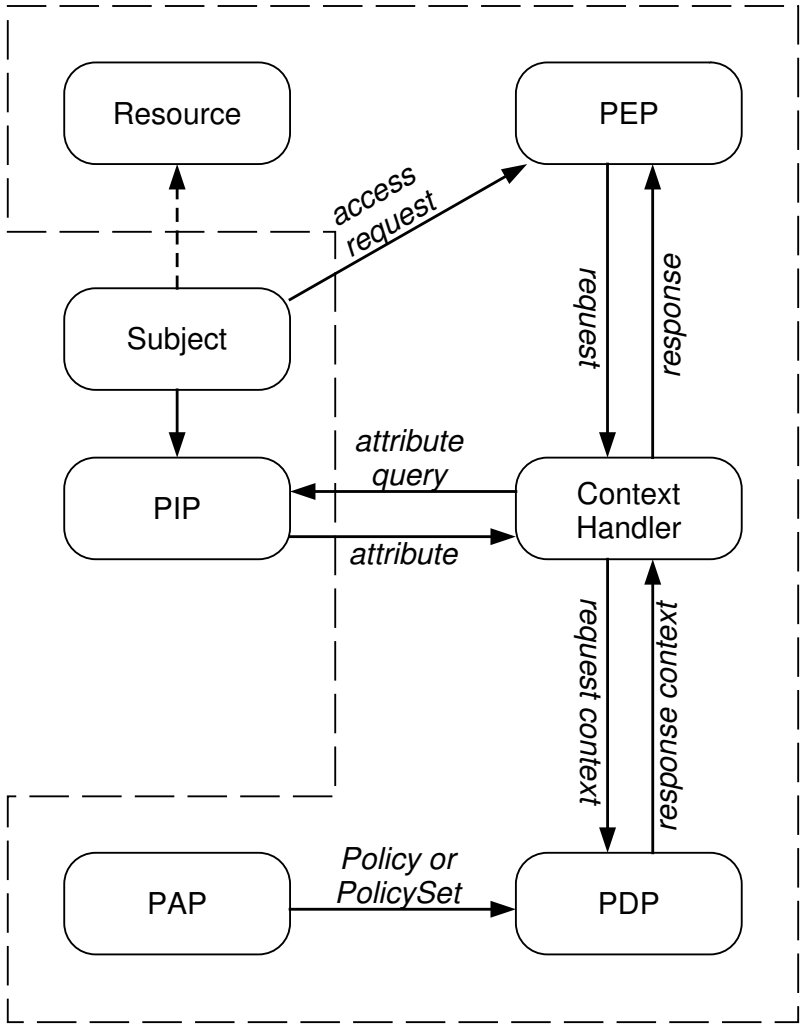
XACML is a dialect of XML used to specify and enforce authorization policies

XACML 2.0 was approved as OASIS standard on 1 February 2005

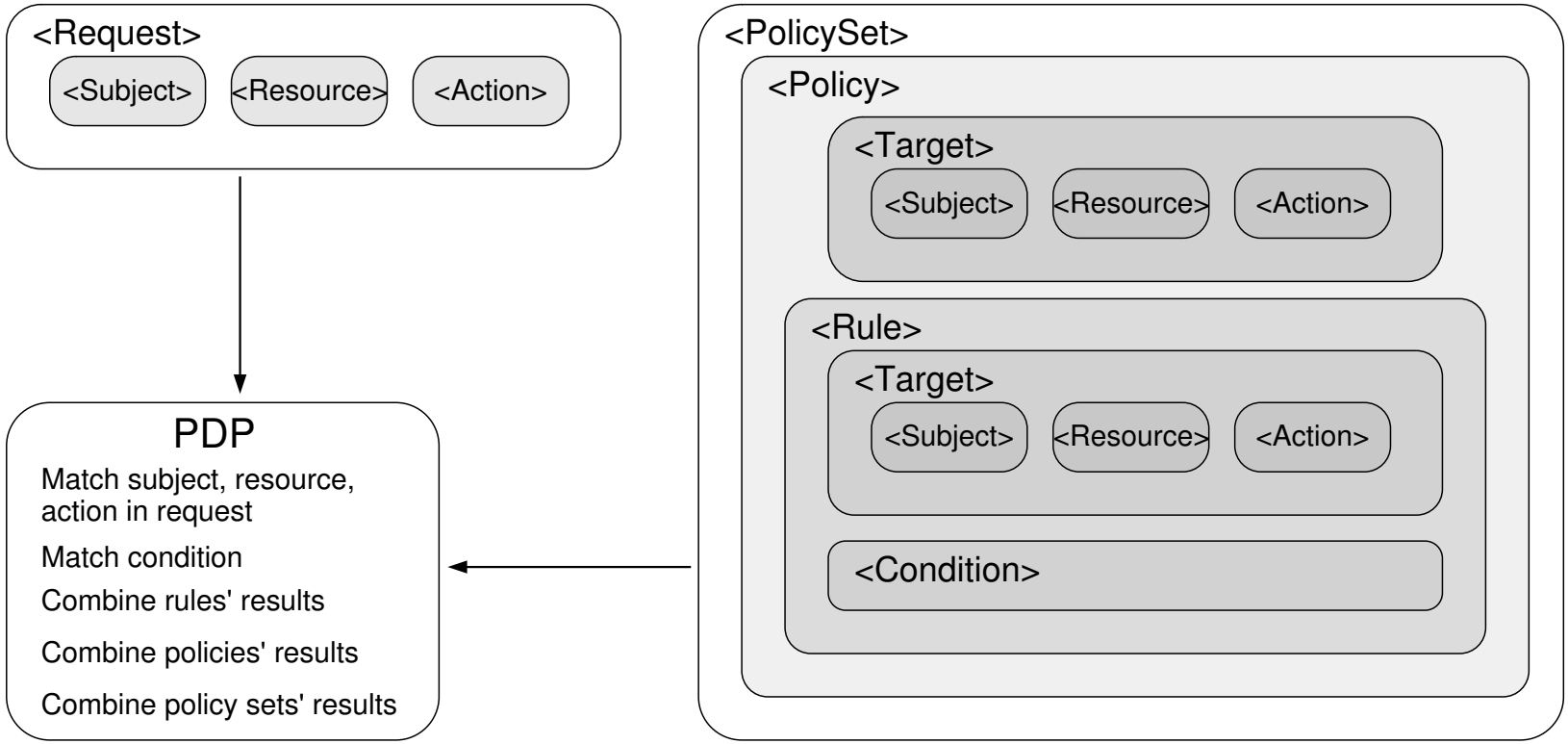
XACML is intended to provide

- Interchangeable policy format
- Support for fine- and coarse-grained authorization policies
- Conditional authorization
- Policy combination and conflict resolution
- Independency from implementation

The XACML view of access control

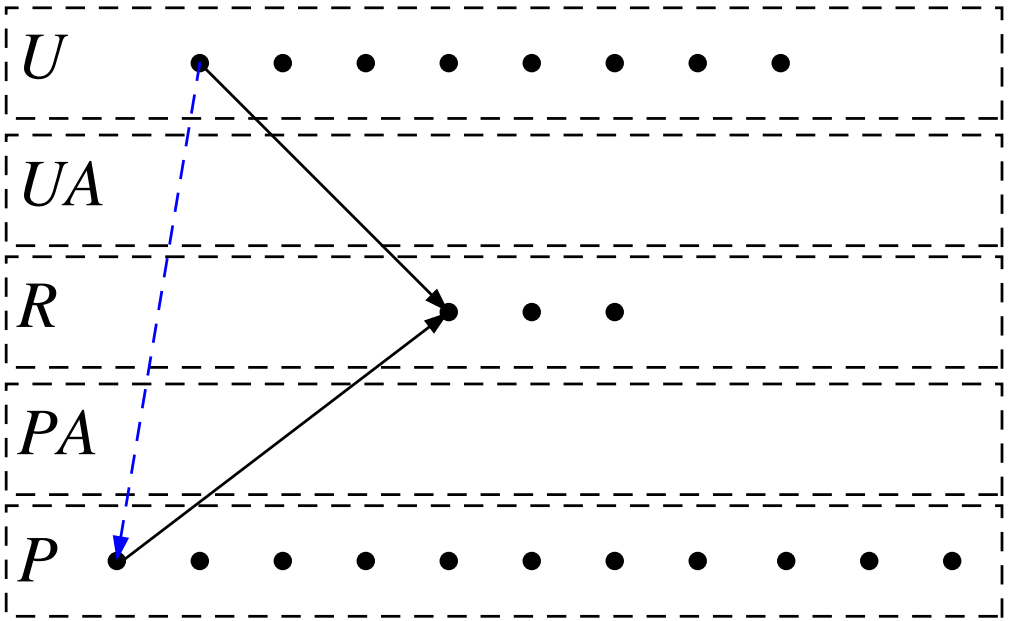


XACML building blocks

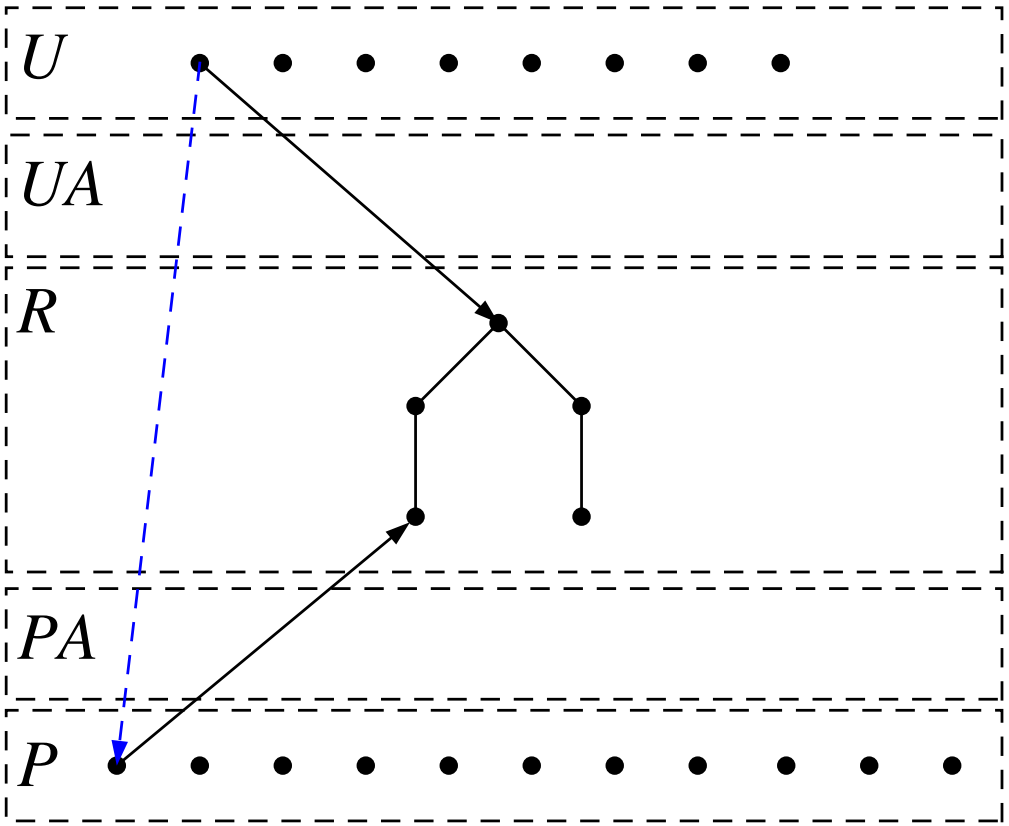


ANSI RBAC

Core RBAC



Hierarchical RBAC



The XACML RBAC profile

Introduction

RB-XACML 2.0 approved as OASIS committee draft 30 September 2004

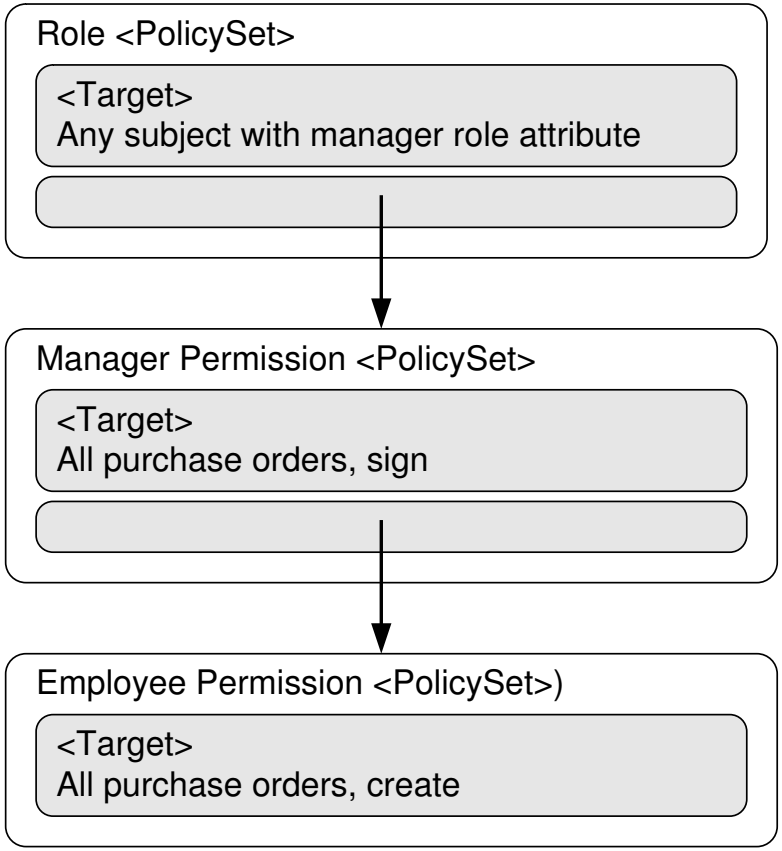
Implements *core* and *hierarchical* components of ANSI standard

- Roles and role hierarchies
- Permission-role assignment relation
- User-role assignment relation

Does not support separation of duty

- RB-XACML 1.0 did support separation of duty

RB-XACML policies



- Role assignment is strongly bound to role definition
- Permissions are strongly bound to roles
- Role hierarchy is defined implicitly using permission aggregation
- Extensive use is made of `<PolicyIdReference>` and `<PolicySetIdReference>` elements

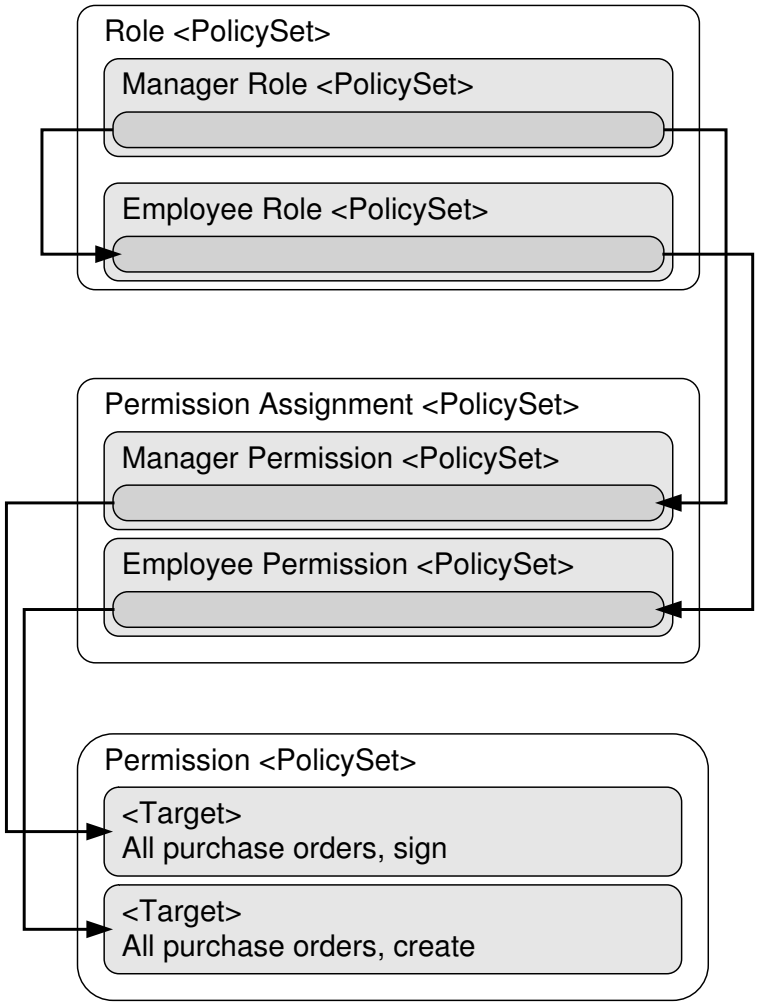
A different formulation of RBAC using XACML

Introduction

Aims are to

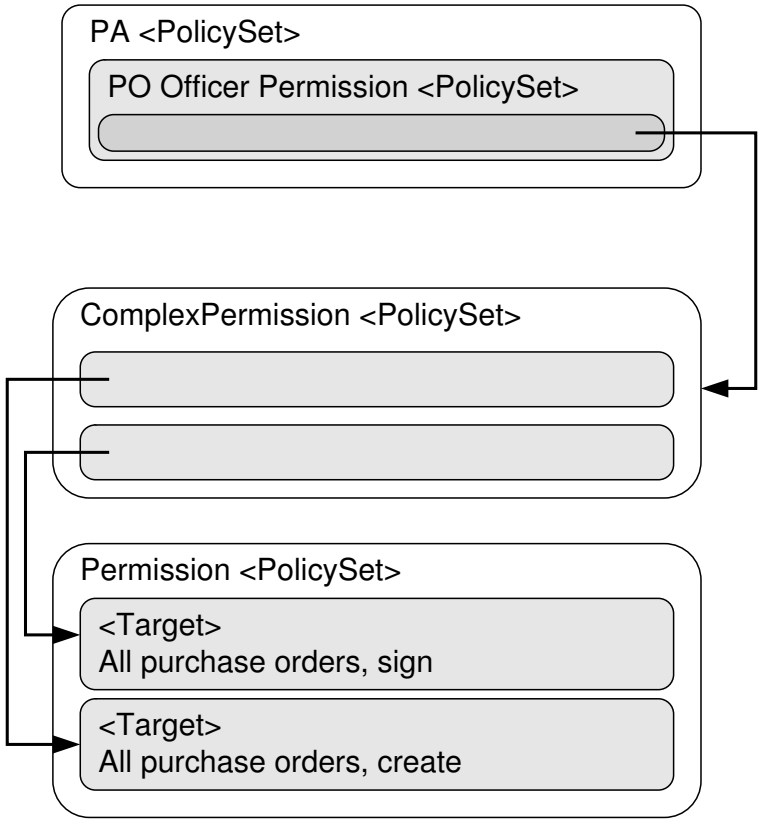
- Obtain a closer correspondence between XACML policies and RBAC model
- Provide a more natural way of defining
  - Role hierarchies
  - Permissions
  - Permission-role assignment
- Support the idea of *complex permissions*

Crampton's role-based XACML policies



- Role set explicitly defines role hierarchy
- No mechanism for associating subjects with roles
- Permissions are first-class entities
- Permission can (easily) be assigned to multiple roles

Complex permissions

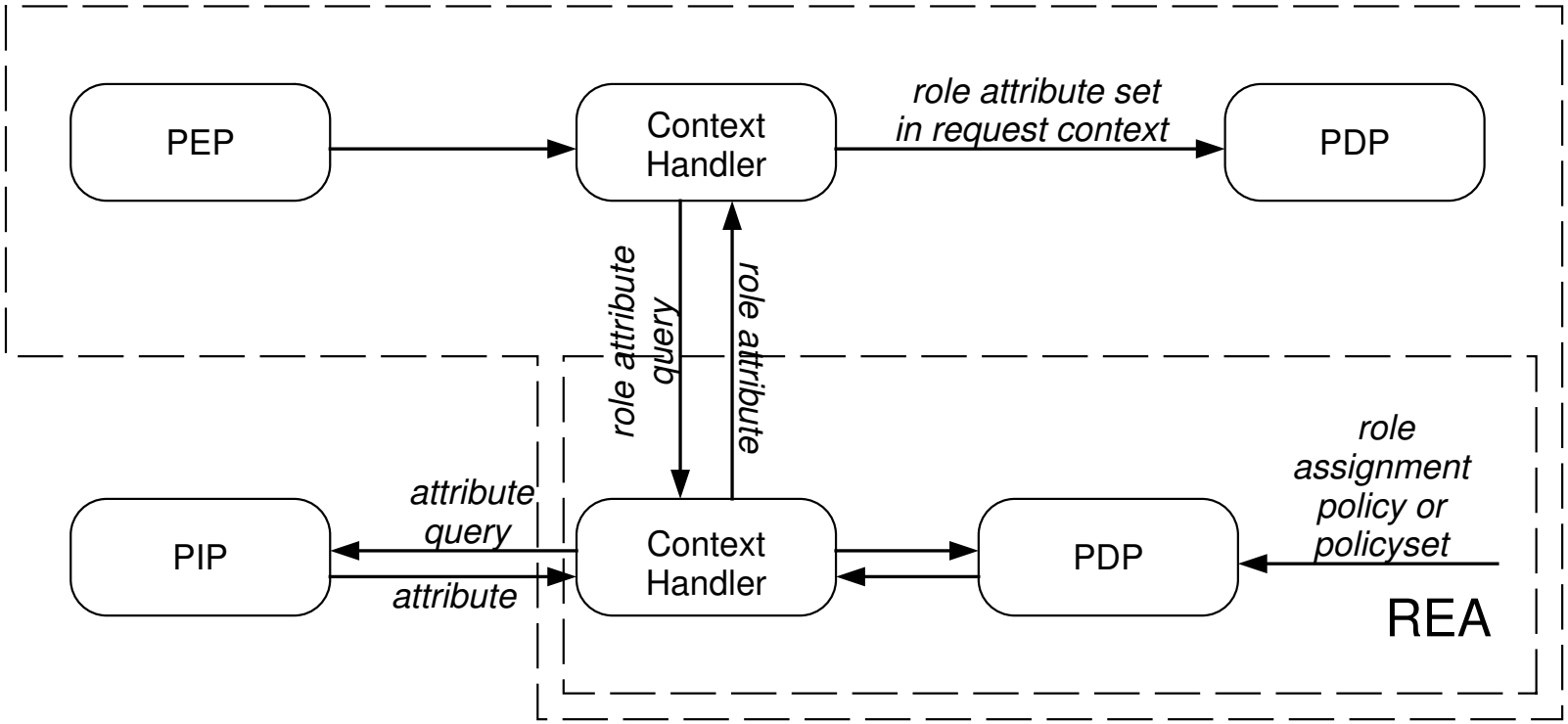


Useful for hierarchically structured resources

- XML data (Crampton, SWS 2004)
- File systems
- Object-oriented applications

Assigning subjects to roles

RB-XACML view of user-role assignment



Observations

The design of the REA and role assignment policies is rather unambitious

- The REA matches subject IDs to role attributes using a Role Assignment `<PolicySet>`
- Designed for centralized systems with a known user population
  - Hardly suitable for web services!

In XACML the `<Subject>` of an access request can be defined in terms of the requester's attributes rather than its identity

- The context handler is responsible for constructing the request and verifying the authenticity of the attributes (using PIPs)
- The PDP matches `<Target>` elements in policies and rules to attributes in the request context

Attribute-based role assignment (1)

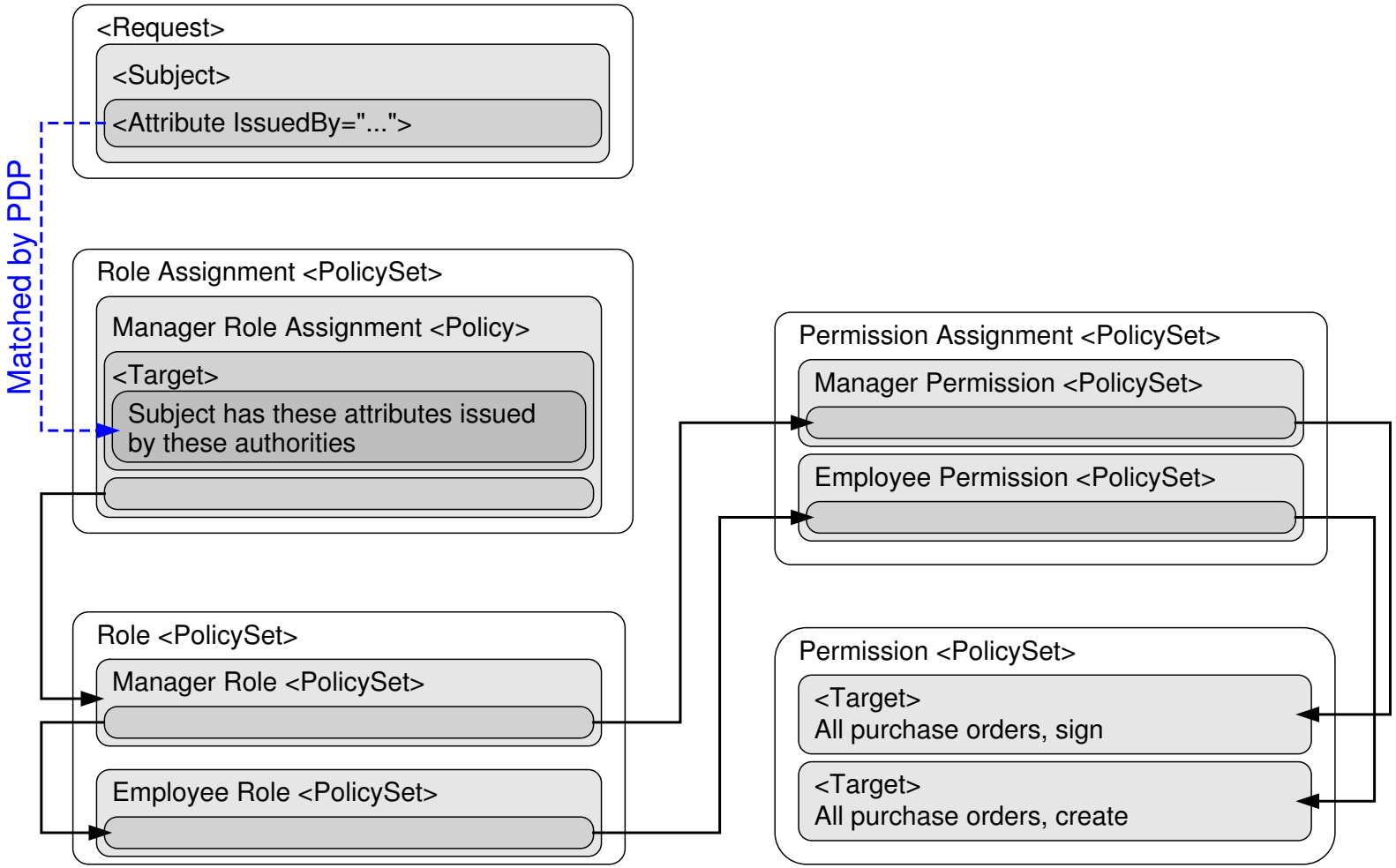
Use policy that assigns subjects to roles based on requester attributes (RBTM, Author-$\mathcal{X}$, TPL)

- Attributes define `<Subject>` element in request
- Context handler is responsible for obtaining and verifying the authenticity of the attributes

PDP matches attributes in request to role(s) using Role Assignment `<PolicySet>`

- Role is now explicitly defined by the attributes that are required to enter into the role

Attribute-based role assignment (2)



Separation of duty using XACML

Introduction

Policy requirement: *No purchase order can be created and signed by the same user*

One common solution (ANSI RBAC) is to ensure that no user has the permission to both create and sign a purchase order

- This solution imposes a constraint on users
  - There does not exist a user that can create and sign a purchase order
- The requirement is a constraint on purchase orders
  - There does not exist a purchase order that has been created and signed by the same user

Separation of duty in RBAC

This solution is particularly unattractive in a role-based context

- The permissions to create and sign a purchase order must be assigned to different roles r_{create} and r_{sign}
- No user can be assigned to both r_{create} and r_{sign}
- No role can be more senior than both r_{create} and r_{sign}

These disadvantages can be mitigated using dynamic rather than static separation of duty

Blacklists

An alternative solution is to implement user-based separation of duty at the object instance level (Crampton, SACMAT 2003)

- Inspired by history matrix concept from Chinese wall model and capability lists derived from access control matrix model
- Maintain dynamic "anti-capability-lists" or *blacklists* for each user
- If `Jason` creates purchase order p then the permission $(p, sign)$ is appended to `Jason`'s blacklist
- Concept can be generalized to implement other forms of separation of duty

Blacklists using XACML

```
<PolicySet PolicySetId="blacklists" ...>
  <Target> ... anyone ... any resource ... any action ... </Target>
  <PolicyIdReference>blacklist:jason</PolicyIdReference>
  ⋮
</PolicySet>

<Policy PolicyId="blacklist:jason" ...
        RuleCombiningAlgorithm="deny-overrides" ... >
  <Target> ... jason ... any resource ... any action ... </Target>
  <Rule Effect="Deny" ... >
    <Target> ... purchase-order-id="123" action="sign" ... </Target>
  </Rule>
    ⋮
</Policy>
```

XACML `<Obligations>`

Policy requirement: *A physician may write to a medical record provided an email is sent to the patient*

The (optional) `<Obligations>` element of an XACML `<Policy>` is a directive to the PEP to perform additional processing following the enforcement of an access control decision

- Contains one or more `<Obligation>` elements
- Typically references elements in the request context
- Processing of `<Obligations>` elements is application-specific

Putting it all together (1)

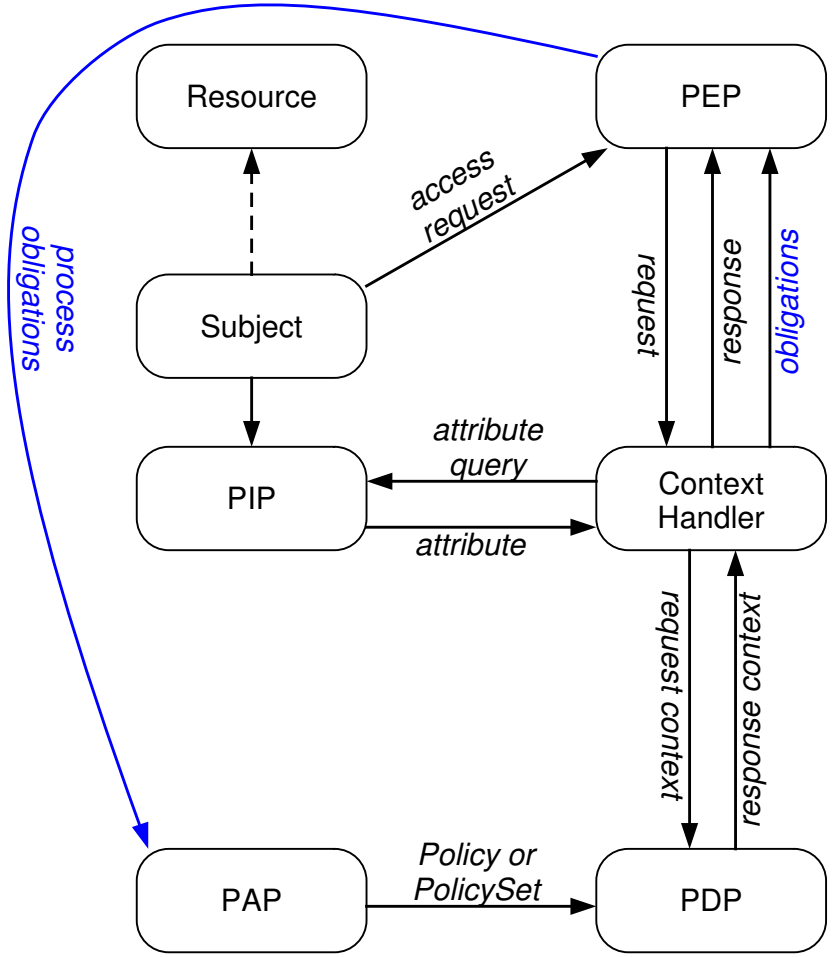
Basic idea is to exploit the `<Obligations>` mechanism to update blacklists

- The permission to create purchase orders must include an `<Obligation>` element
- The `<Obligation>` element requires that the PEP write a new rule to the appropriate blacklist `<Policy>`
  - If a request by `Jason` to create a purchase order is permitted ...
  - ...then a new `<Rule>` must be added to `Jason`'s blacklist

Putting it all together (2)

```
<PolicySet ... PolicySetId="Permission:Set" ... >
  <Policy ... PolicyId="permission:po:create" ... >
    <Rule ... >
      <Target> ... purchase orders ... create </Target>
    </Rule>
    <Obligations ... >
      <Obligation FulfillOn="Permit" ... >
        add permission (this-purchase-order,sign) to subject's blacklist
      </Obligation>
    </Obligations>
  </Policy>
  :
</PolicySet>
```

Putting it all together (3)



Future work

- Liaise with XACML TC
  - Explore the advantages of the alternative formulation of RBAC using XACML
  - Include separation of duty in XACML RBAC profile
- Investigate the extent to which obligations, XACML policies. and role-based administrative models can be used to manage XACML RBAC policies
- Investigate to what extent SAML and XACML can inter-operate to support
  - Attribute-based role assignment
  - Discovery of distributed credentials and credential chains