RABAC : Role-Centric Attribute-Based Access Control

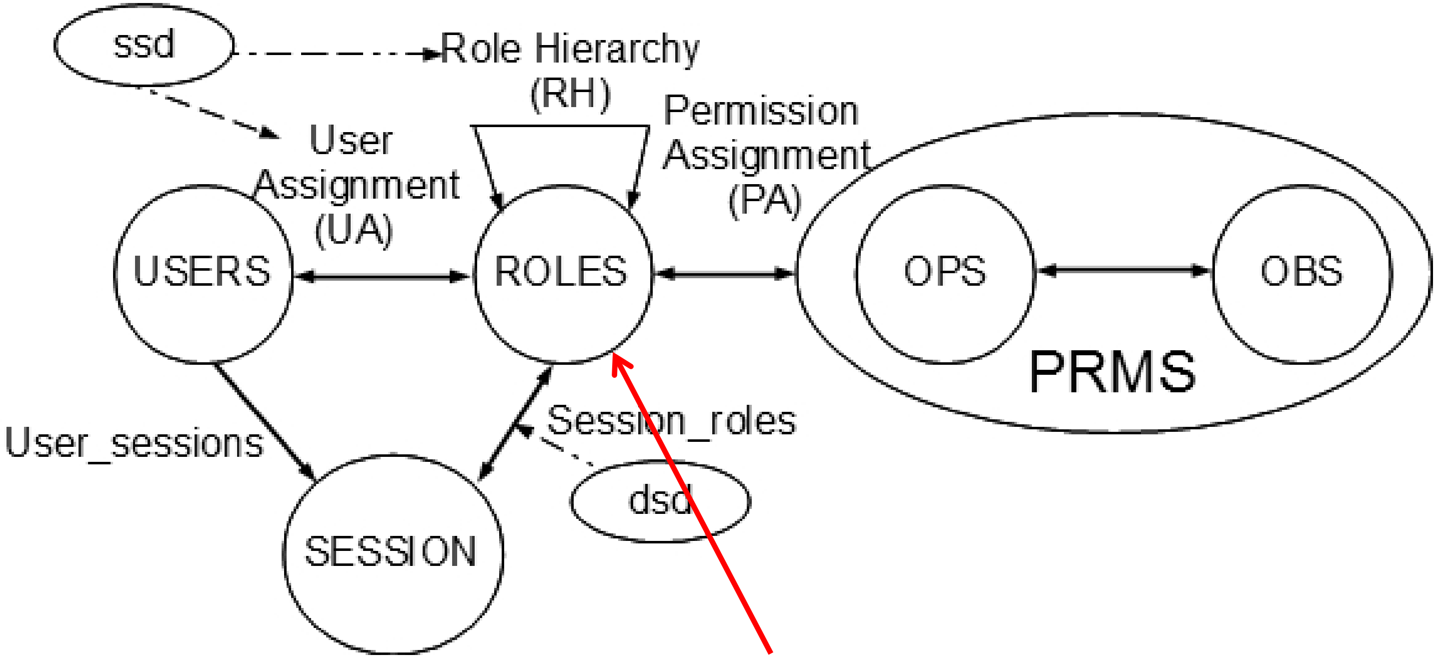
MMM-ACNS 2012

Xin Jin, Ravi Sandhu, Ram Krishnan
University of Texas at San Antonio
San Antonio, TX, USA

OUTLINE

- Motivation
- Proposed Model
- XACML Profile
- Conclusion

Role Based Access Control

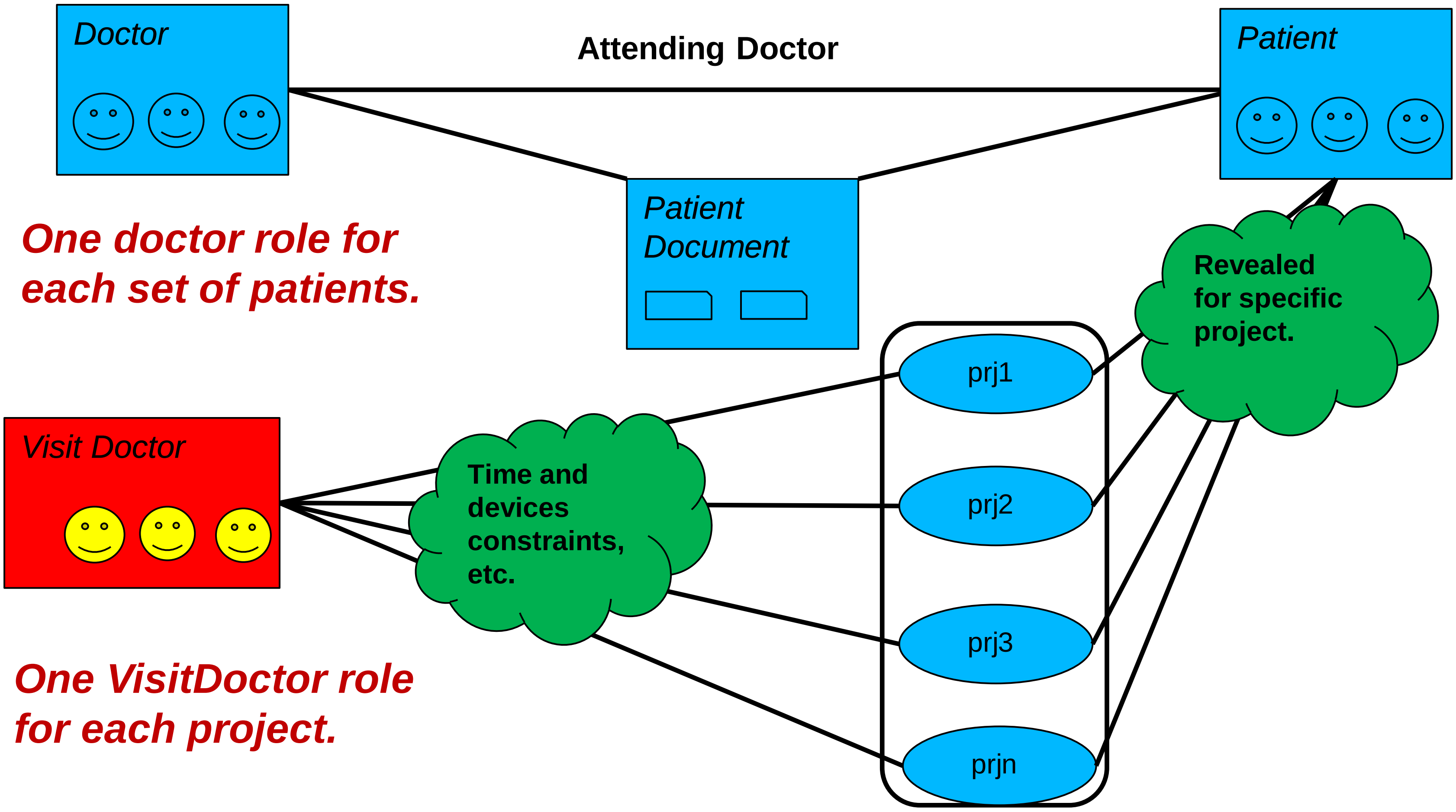


Role Explosion

Role number is supposed to be much than users.

Role Explosion : Different roles have to be defined for slightly different sets of permissions.

Example



→ ***One doctor role for each set of patients.***

→ ***One VisitDoctor role for each project.***

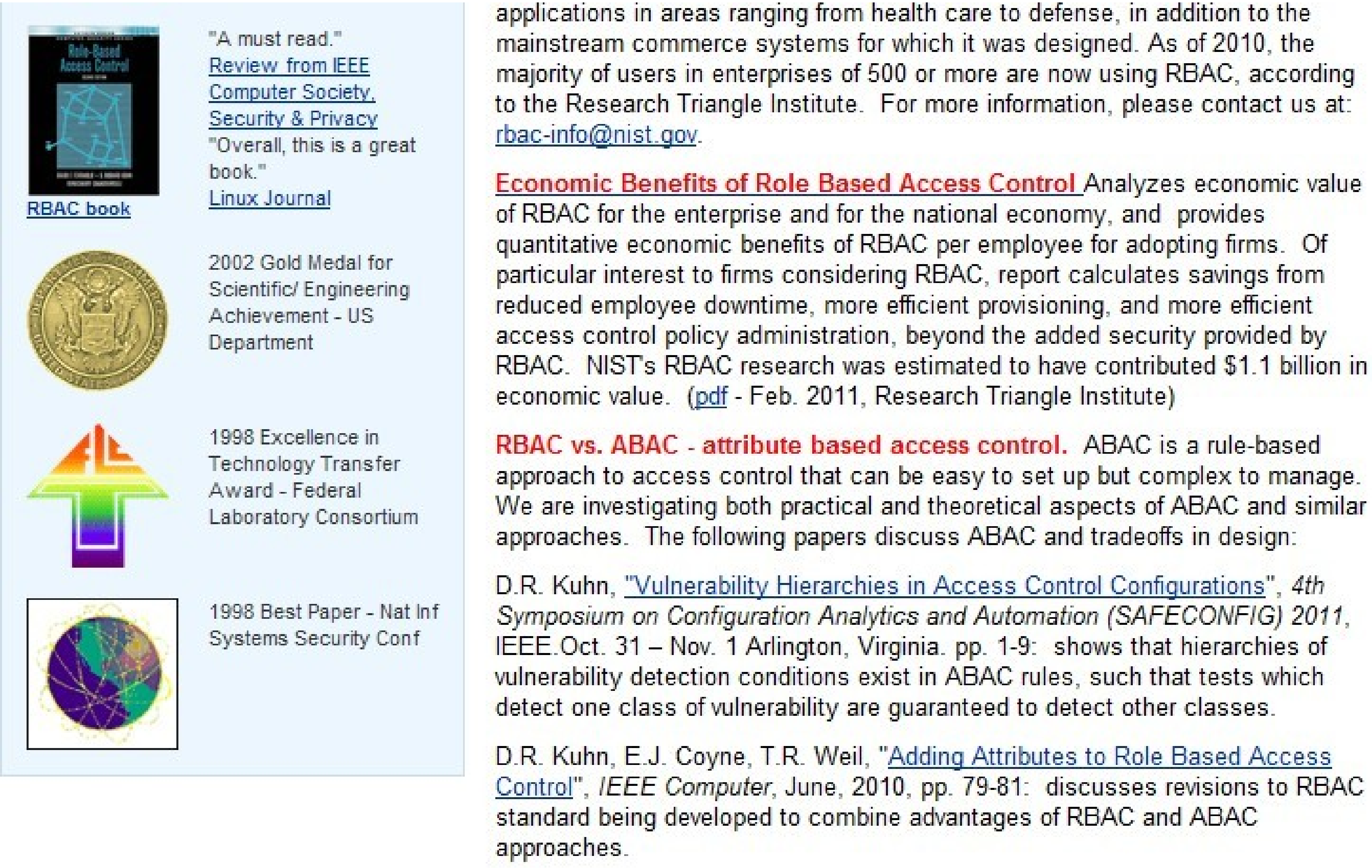
Related Work

- Role Template, Parameterized Role, Attributed role, etc
- Two level RBAC (SACMAT 12)
- Environment Role, Object Role
- Automatic user-role assignment, TrustBAC
- Relationship based access control (ReBAC)
- Role and organization based access control (ROBAC)

They need modification in user-role and role-permission assignment.
Role engineering is the most costly work in constructing RBAC system.

Why can't we design a solution which can be enforced with least impact to current deployment?

Motivation



"A must read."
Review from IEEE Computer Society, Security & Privacy
"Overall, this is a great book."
Linux Journal

RBAC book

2002 Gold Medal for Scientific/ Engineering Achievement - US Department

1998 Excellence in Technology Transfer Award - Federal Laboratory Consortium

1998 Best Paper - Nat Inf Systems Security Conf

applications in areas ranging from health care to defense, in addition to the mainstream commerce systems for which it was designed. As of 2010, the majority of users in enterprises of 500 or more are now using RBAC, according to the Research Triangle Institute. For more information, please contact us at: rbac-info@nist.gov.

Economic Benefits of Role Based Access Control Analyzes economic value of RBAC for the enterprise and for the national economy, and provides quantitative economic benefits of RBAC per employee for adopting firms. Of particular interest to firms considering RBAC, report calculates savings from reduced employee downtime, more efficient provisioning, and more efficient access control policy administration, beyond the added security provided by RBAC. NIST's RBAC research was estimated to have contributed $1.1 billion in economic value. (pdf - Feb. 2011, Research Triangle Institute)

RBAC vs. ABAC - attribute based access control. ABAC is a rule-based approach to access control that can be easy to set up but complex to manage. We are investigating both practical and theoretical aspects of ABAC and similar approaches. The following papers discuss ABAC and tradeoffs in design:

D.R. Kuhn, "Vulnerability Hierarchies in Access Control Configurations", *4th Symposium on Configuration Analytics and Automation (SAFECONFIG) 2011*, IEEE.Oct. 31 – Nov. 1 Arlington, Virginia. pp. 1-9: shows that hierarchies of vulnerability detection conditions exist in ABAC rules, such that tests which detect one class of vulnerability are guaranteed to detect other classes.

D.R. Kuhn, E.J. Coyne, T.R. Weil, "Adding Attributes to Role Based Access Control", *IEEE Computer*, June, 2010, pp. 79-81: discusses revisions to RBAC standard being developed to combine advantages of RBAC and ABAC approaches.

Motivation

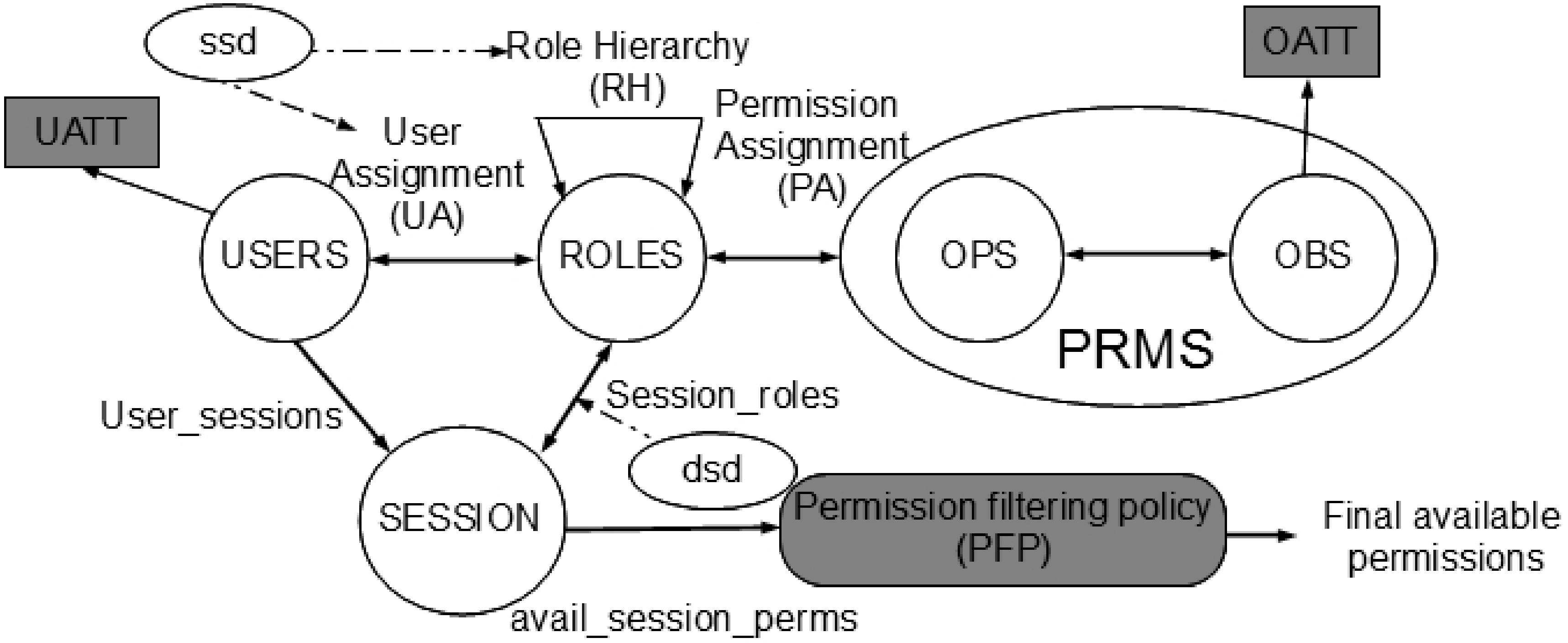
- NIST proposed three alternative revisions to RBAC standard
  - Attribute Centric
    - Totally attribute based, role as a user attribute
    - Related work: ABAC–alpha model [Jin, DBSEC12], etc
  - Dynamic Roles
    - Automatically user-role assignment [Kahtani & Sandhu],etc
  - Role Centric RBAC
    - Not too much research.

With previous work in ABAC-alpha, We provide a formal model for Role-Centric attribute based access control.

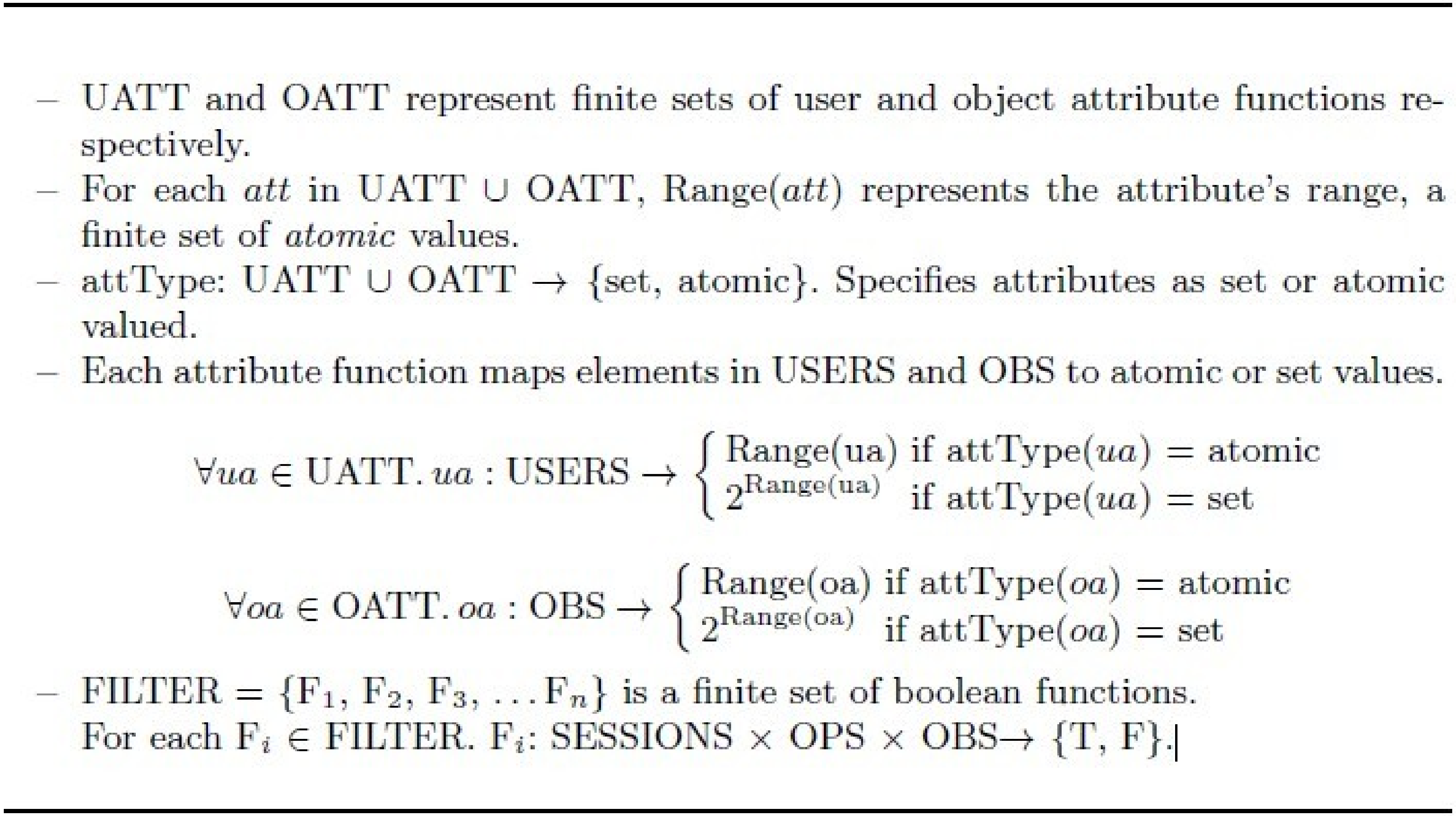
OUTLINE

- Motivation
- Proposed Model
- XACML Profile
- Conclusion

Model Components



Model Components



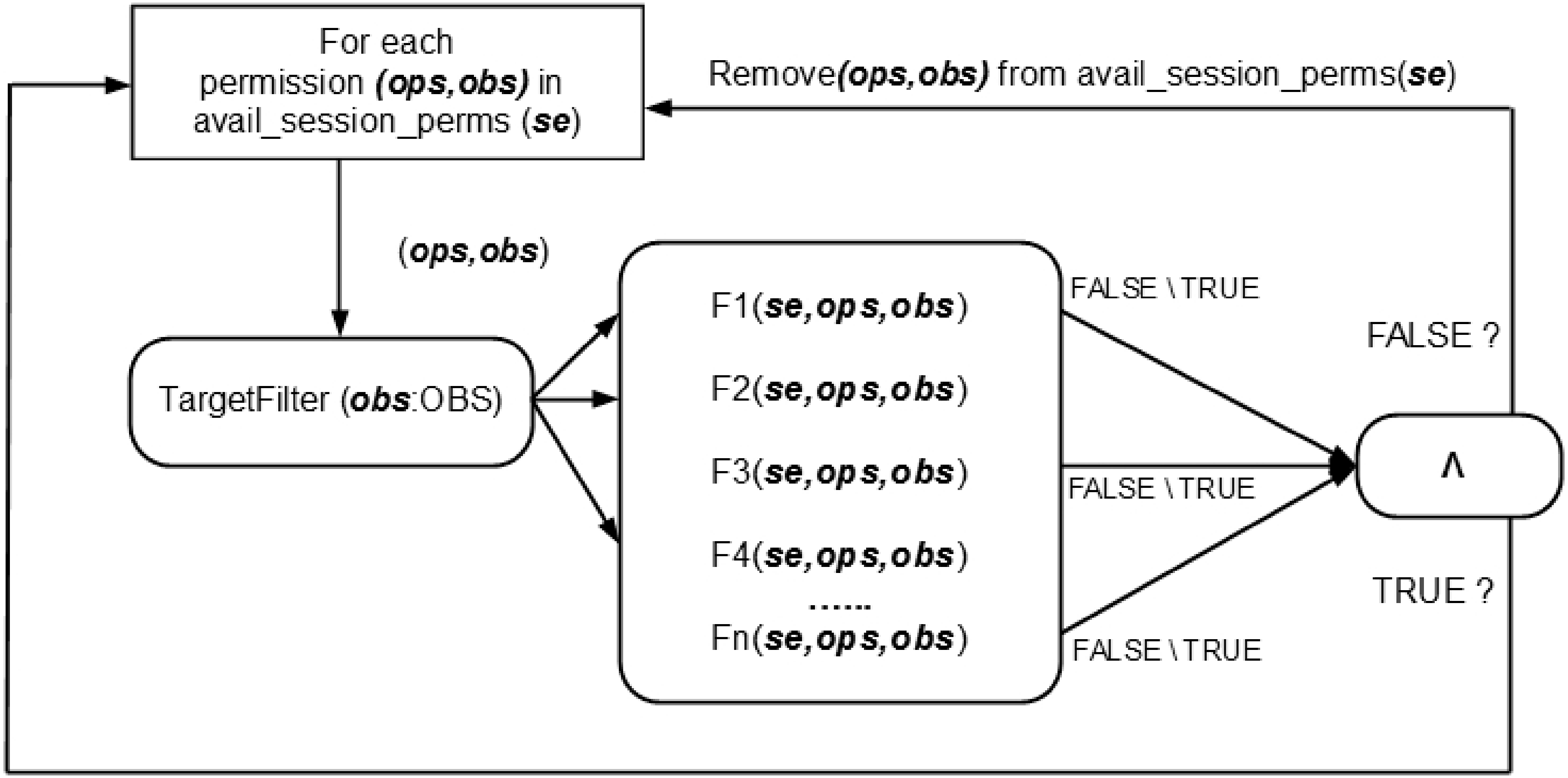
- UATT and OATT represent finite sets of user and object attribute functions respectively.
- For each *att* in UATT ∪ OATT, Range(*att*) represents the attribute's range, a finite set of *atomic* values.
- attType: UATT ∪ OATT → {set, atomic}. Specifies attributes as set or atomic valued.
- Each attribute function maps elements in USERS and OBS to atomic or set values.

$$\forall ua \in \mathrm{UATT}.\ ua : \mathrm{USERS} \rightarrow \begin{cases} \mathrm{Range(ua)} & \text{if } \mathrm{attType}(ua) = \text{atomic} \\ 2^{\mathrm{Range(ua)}} & \text{if } \mathrm{attType}(ua) = \text{set} \end{cases}$$

$$\forall oa \in \mathrm{OATT}.\ oa : \mathrm{OBS} \rightarrow \begin{cases} \mathrm{Range(oa)} & \text{if } \mathrm{attType}(oa) = \text{atomic} \\ 2^{\mathrm{Range(oa)}} & \text{if } \mathrm{attType}(oa) = \text{set} \end{cases}$$

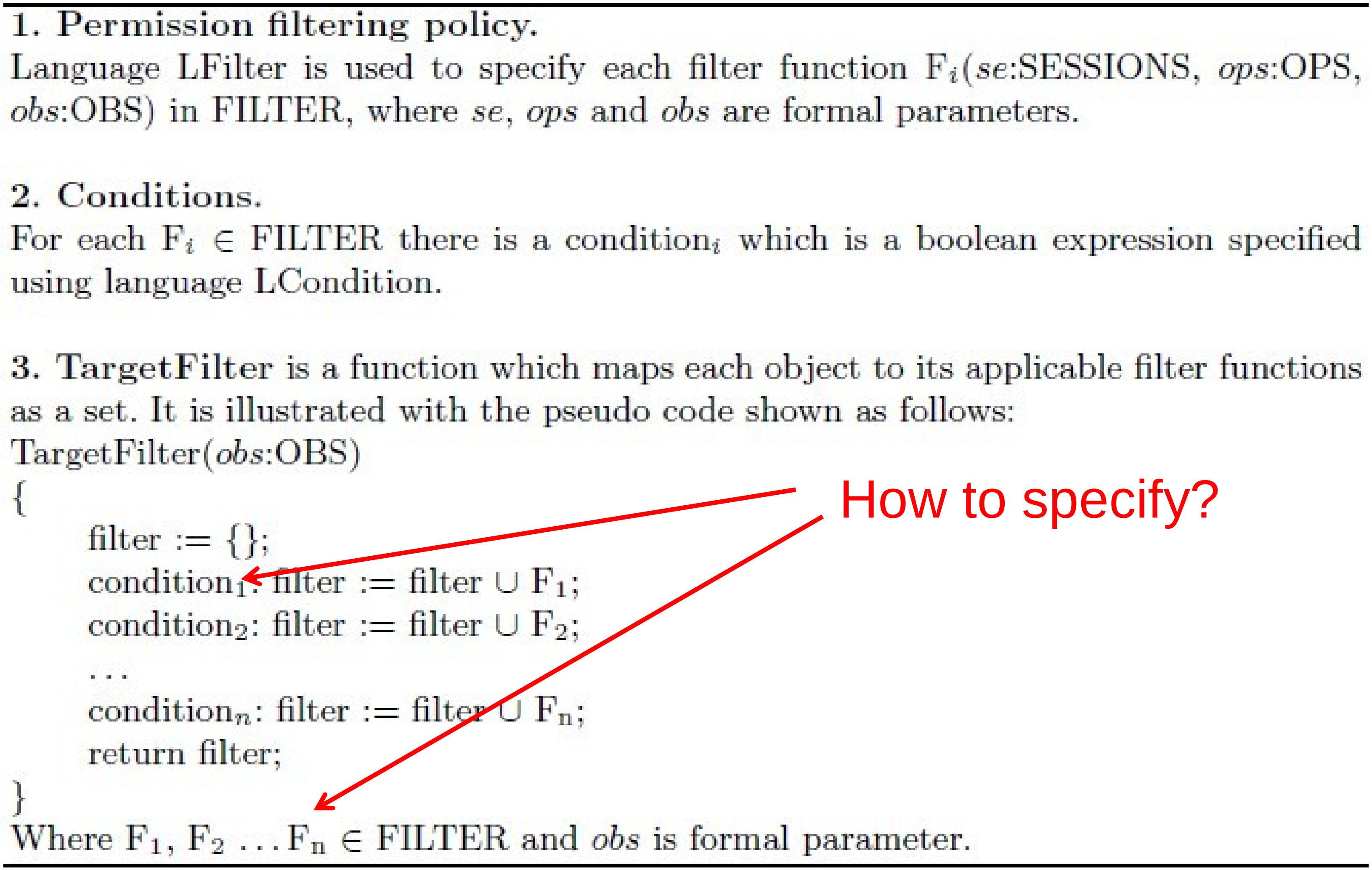
- FILTER = $\{F_1, F_2, F_3, \ldots F_n\}$ is a finite set of boolean functions. For each $F_i \in$ FILTER. F_i: SESSIONS × OPS × OBS→ {T, F}.

Filtering Policy



World-Leading Research with Real-World Impact!

Filtering Policy



1. Permission filtering policy.
Language LFilter is used to specify each filter function $F_i(se{:}\text{SESSIONS}, ops{:}\text{OPS}, obs{:}\text{OBS})$ in FILTER, where *se*, *ops* and *obs* are formal parameters.

2. Conditions.
For each $F_i \in$ FILTER there is a condition_i which is a boolean expression specified using language LCondition.

3. TargetFilter is a function which maps each object to its applicable filter functions as a set. It is illustrated with the pseudo code shown as follows:

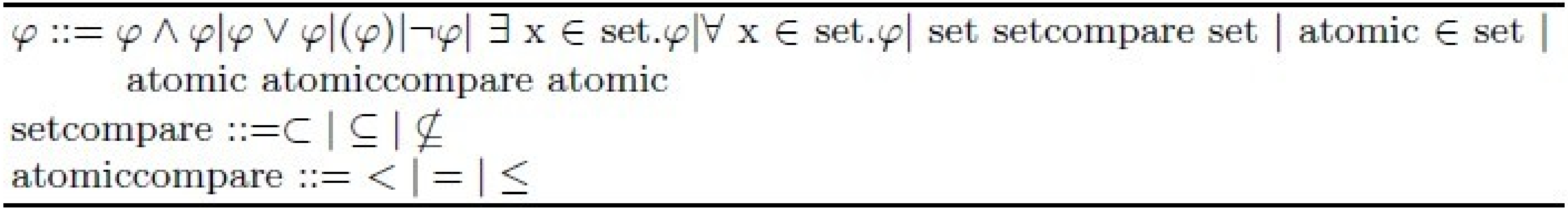
```
TargetFilter(obs:OBS)
{
        filter := {};
        $\text{condition}_1$: filter := filter $\cup$ $F_1$;
        $\text{condition}_2$: filter := filter $\cup$ $F_2$;
        . . .
        $\text{condition}_n$: filter := filter $\cup$ $F_n$;
        return filter;
}
```

Where $F_1, F_2 \ldots F_n \in$ FILTER and *obs* is formal parameter.

How to specify?

Language for Policy

Common Policy Language (CPL) :



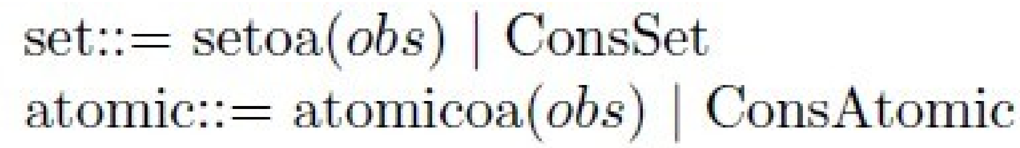
$$\varphi ::= \varphi \wedge \varphi | \varphi \vee \varphi | (\varphi) | \neg\varphi | \ \exists \text{ x} \in \text{set}.\varphi | \forall \text{ x} \in \text{set}.\varphi | \text{ set setcompare set } | \text{ atomic} \in \text{set } |$$

$$\qquad \text{atomic atomiccompare atomic}$$

$$\text{setcompare} ::= \subset | \subseteq | \not\subset$$

$$\text{atomiccompare} ::= < | = | \leq$$

LCondition, used to specify each condition, is an instance of CPL where:



$$\text{set} ::= \text{setoa}(obs) \mid \text{ConsSet}$$

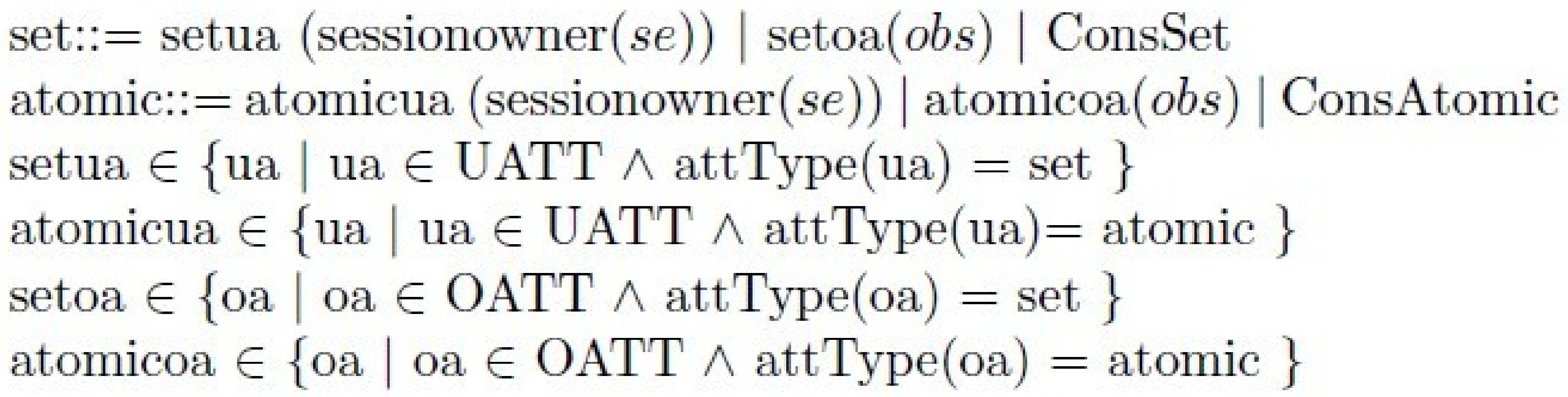
$$\text{atomic} ::= \text{atomicoa}(obs) \mid \text{ConsAtomic}$$

Example:

type(o) = studentrecord ⋀ (owner(o) ∊ GameClub ⋁ (∃ reader ∊ reader(o). reader = user3))

Language for Policy

LFilter, used to specify each filter, is an instance of CPL where:



$$\text{set}::= \text{setua}\ (\text{sessionowner}(se))\ |\ \text{setoa}(obs)\ |\ \text{ConsSet}$$

$$\text{atomic}::= \text{atomicua}\ (\text{sessionowner}(se))\ |\ \text{atomicoa}(obs)\ |\ \text{ConsAtomic}$$

$$\text{setua} \in \{\text{ua}\ |\ \text{ua} \in \text{UATT} \wedge \text{attType}(\text{ua}) = \text{set}\ \}$$

$$\text{atomicua} \in \{\text{ua}\ |\ \text{ua} \in \text{UATT} \wedge \text{attType}(\text{ua}) = \text{atomic}\ \}$$

$$\text{setoa} \in \{\text{oa}\ |\ \text{oa} \in \text{OATT} \wedge \text{attType}(\text{oa}) = \text{set}\ \}$$

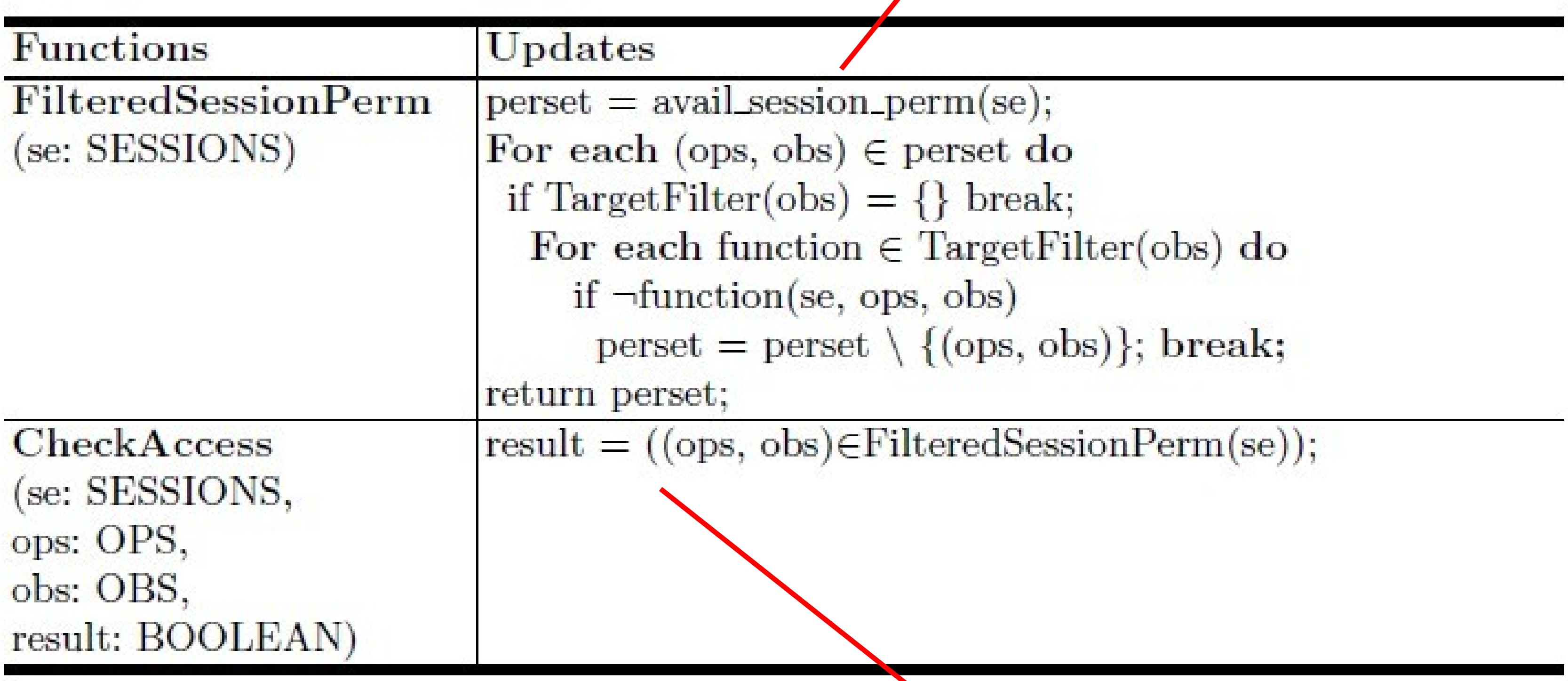
$$\text{atomicoa} \in \{\text{oa}\ |\ \text{oa} \in \text{OATT} \wedge \text{attType}(\text{oa}) = \text{atomic}\ \}$$

Example:

major(u) = major(o) ⋀ (location(u)= utsa ⋁ ∃ project∈ involvedprj(u). project=proj(o))

Access Checking

Apply policy and get final available permissions in session



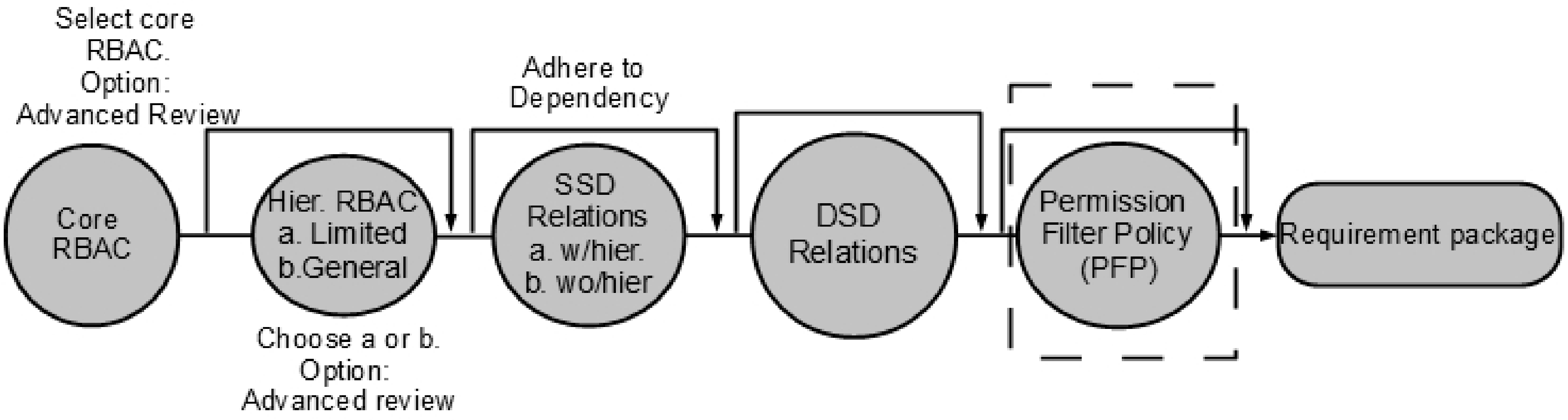
Functions	Updates

FilteredSessionPerm
(se: SESSIONS)
	perset = avail_session_perm(se);
	For each (ops, obs) $\in$ perset **do**
	  if TargetFilter(obs) = {} break;
	    For each function $\in$ TargetFilter(obs) **do**
	      if $\neg$function(se, ops, obs)
	        perset = perset $\setminus$ {(ops, obs)}; **break;**
	return perset;

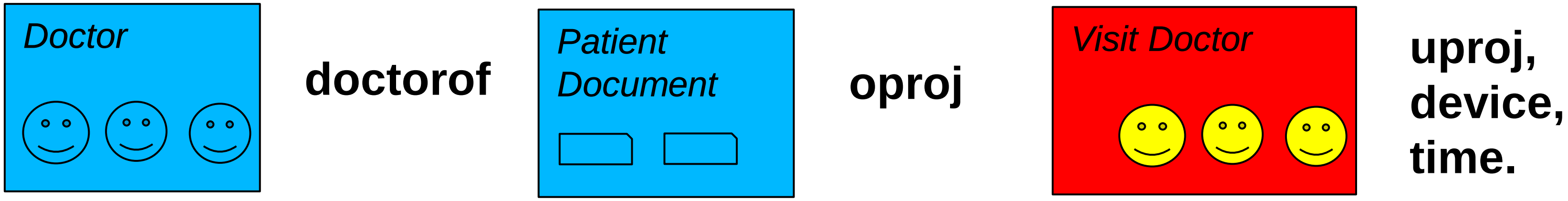
CheckAccess (se: SESSIONS, ops: OPS, obs: OBS, result: BOOLEAN)	result = ((ops, obs)$\in$FilteredSessionPerm(se));

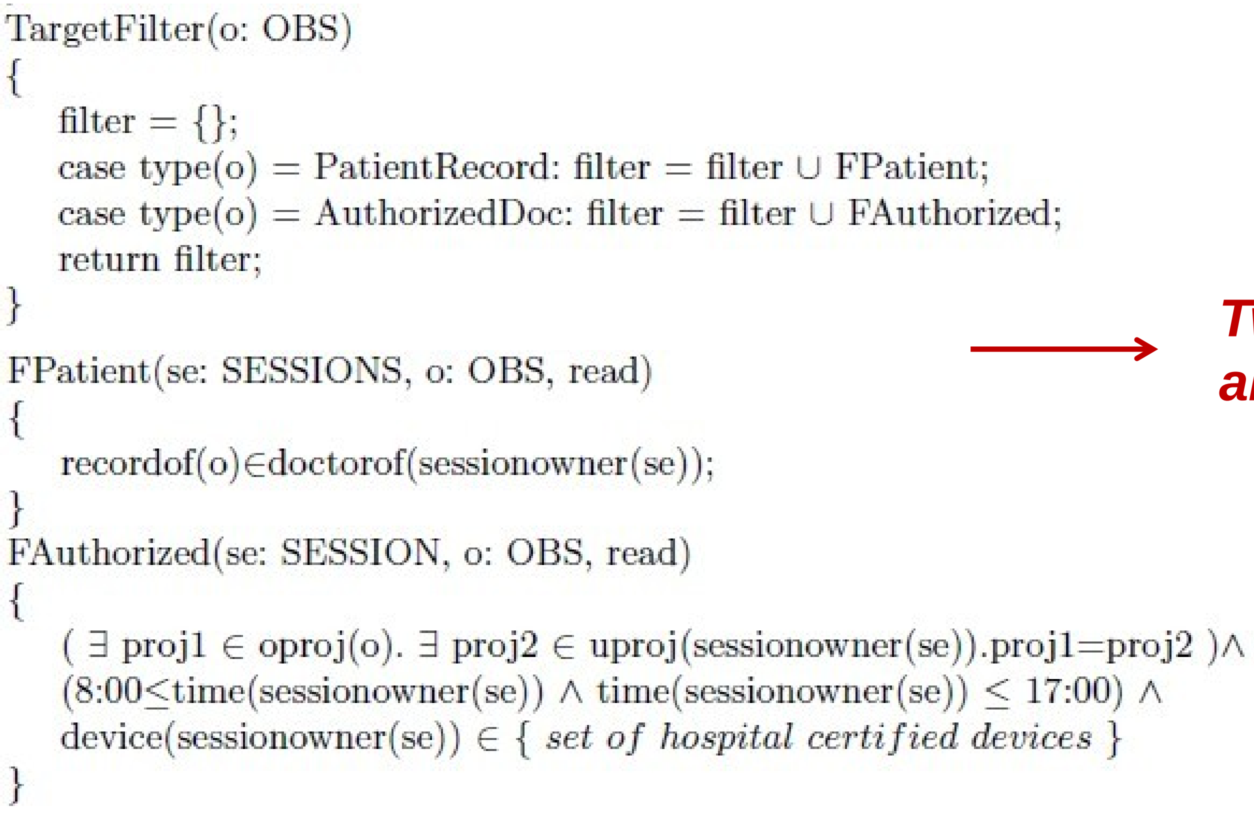
Check against user request

Package Building Path



Advantage





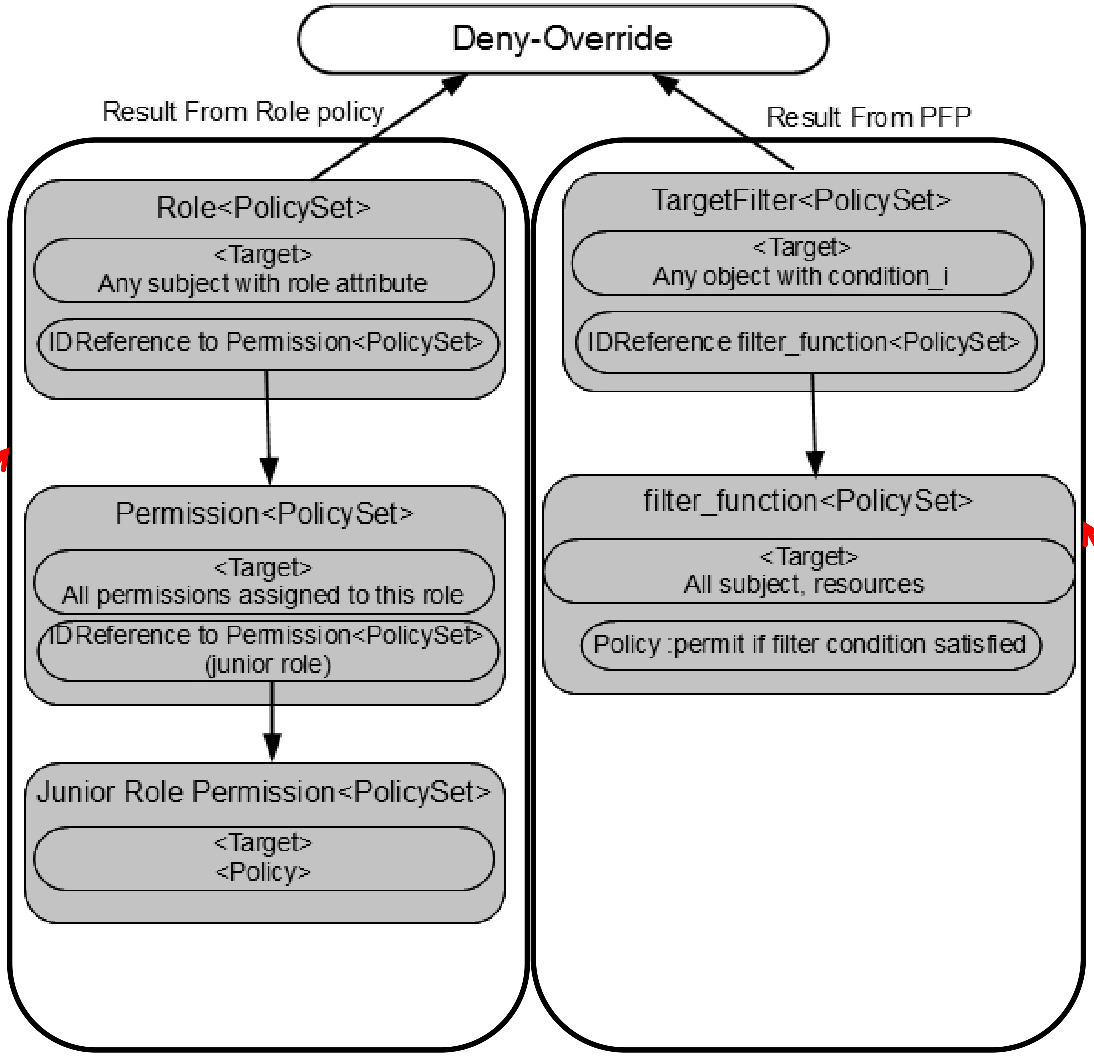
```
TargetFilter(o: OBS)
{
    filter = {};
    case type(o) = PatientRecord: filter = filter $\cup$ FPatient;
    case type(o) = AuthorizedDoc: filter = filter $\cup$ FAuthorized;
    return filter;
}
FPatient(se: SESSIONS, o: OBS, read)
{
    recordof(o)$\in$doctorof(sessionowner(se));
}
FAuthorized(se: SESSION, o: OBS, read)
{
    ( $\exists$ proj1 $\in$ oproj(o). $\exists$ proj2 $\in$ uproj(sessionowner(se)).proj1=proj2 )$\wedge$
    (8:00$\leq$time(sessionowner(se)) $\wedge$ time(sessionowner(se)) $\leq$ 17:00) $\wedge$
    device(sessionowner(se)) $\in$ { $set\ of\ hospital\ certified\ devices$ }
}
```

→ ***Two role definitions are enough.***

OUTLINE

- ➢ Motivation
- ➢ Proposed Model
- ➢ XACML Profile
- ➢ Conclusion

XACML-Profile for RABAC


Deny-Override
Result From Role policy
Result From PFP
Role<PolicySet>
<Target>
Any subject with role attribute
IDReference to Permission<PolicySet>
Permission<PolicySet>
<Target>
All permissions assigned to this role
IDReference to Permission<PolicySet>
(junior role)
Junior Role Permission<PolicySet>
<Target>
<Policy>
TargetFilter<PolicySet>
<Target>
Any object with condition_i
IDReference filter_function<PolicySet>
filter_function<PolicySet>
<Target>
All subject, resources
Policy :permit if filter condition satisfied

XACML Profile for RBAC

XACML express permission filtering policy

OUTLINE

- Motivation
- Proposed Model
- Use Case
- XACML Profile
- Conclusion

Conclusion

- Main contribution
  - RABAC model: Extension to RBAC with filtering policy
  - Define languages for specifying policy
  - Modify functions for access checking
- Advantages
  - Without modification to original deployment while mitigating role explosion problem.
  - Retains the administration convenience of RBAC
  - Offer flexibility and administration convenience.
- Future work
  - Distinguish user attribute and session attribute.
  - Enhance policy language.

Thanks

Any Questions?