A Provenance-based Access Control Model for Dynamic Separation of Duties

July 10, 2013
PST 2013

Dang Nguyen, Jaehong Park, and Ravi Sandhu
Institute for Cyber Security
University of Texas at San Antonio

Separation of Duties (SoD)

- Duties
  - The responsibilities required for accomplishing a certain task
  - Example: washing dishes, flying airplane, saving the world, etc.
  - Responsibilities are assigned to people (or users)
- Conflicting Duties
  - Too many responsibilities = corrupted power
  - Example: “One Ring to rule them all”
- Essentially an Access Control Problem
  - Who can have which responsibility?

RBAC Approach for SoD

- Roles as semantic constructs
  - Various responsibilities can be encapsulated within a specific role.
  - Example: Professor is responsible for assigning and grading homework.
  - *Responsibilities* are mapped to *roles*, which are then assigned to *users*.
- Conflicting Roles
  - Two main approaches: **Static** and **Dynamic**.

Static Separation of Duties

- Mainly deals with role assignment
  - No two conflicting roles can be assigned to the same user.
  - Example: A user should not be assigned both police and thief roles.
- Narrow scope
  - Unable to address SoD concerns in dynamic environment.

Dynamic Separation of Duties

- Utilizes the Role Activation concept
  - Two conflicting roles can be assigned to the same user, just not activated at the same time (or under other constraints)..
- Variations of DSOD
  - Expressing different concerns.
  - Each concern features unique characteristic.

DSOD Variations + Features

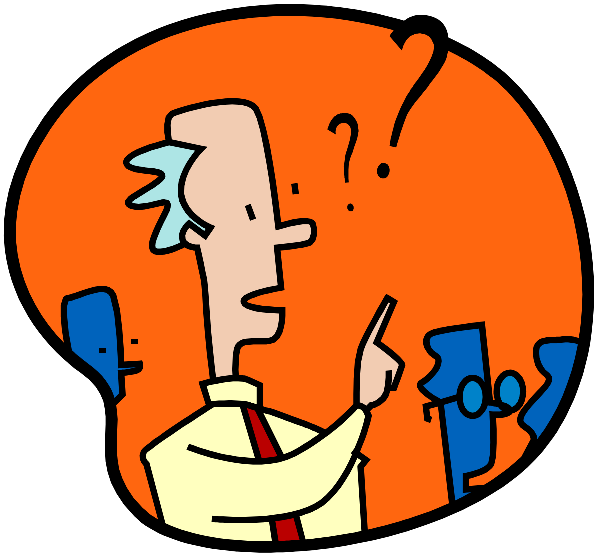
Features	Simple DSOD	Obj-DSOD	Ops-DSOD	HDSOD	TCE
Per Role	√	√	√	√	√
Per Action		√	√	√	√
Per Object		√		√	√
Task-aware			√	√	√
Order-aware				√	√
Weighted Action-aware					√

DSOD Examples

- Scenario: Homework Grading System
  - Students can **upload/replace/submit** a homework to the system.
  - Once it is **submitted**, the homework can be **reviewed** by other students or designated graders until it is **graded** by the teaching assistant (TA).
  - The Professor holds the highest authority.

- Variations of DSOD constraints:
  - Cannot activate roles *Reviewer* and *Student* at the same time – Simple DSOD
  - Can activate roles *Reviewer* and *Student*, but cannot **review** the homework **submitted** – Object-based DSOD
  - Cannot activate roles *TA* and *Student*, if permitted actions cover *Professor's* – Operational DSOD
  - Cannot **grade** a homework before it is **submitted** – History-based DSOD
  - Cannot **grade** a homework unless reviews' combined weights exceeds 3 – TCE

PBAC Approach to DSOD



PBAC Approach to DSOD

- Naturally provide history information
  - Existing approaches assume ready availability for usages.
- Expressive control unit (dependency names)
  - Facilitate policy specification and convenient enforcement.
- Enables new DSOD concerns
  - Capable of capturing more interesting behavior from system events.
- Easily incorporated with other AC mechanisms
  - **RBAC** and more

DSOD Variations + Features

Features	Simple DSOD	Obj-DSOD	Ops-DSOD	HDSOD	TCE	DSOD in PBAC
Per Role	√	√	√	√	√	√
Per Action		√	√	√	√	√
Per Object		√		√	√	√
Task-aware				√	√	√
Order-aware			√	√	√	√
Weighted Action-aware					√	√

DSOD Variations + Features

Features	Simple DSOD	Obj-DSOD	Ops-DSOD	HDSOD	TCE	DSOD in PBAC
Per Role	√	√	√	√	√	√
Per Action		√	√	√	√	√
Per Object		√		√	√	√
Task-aware				√	√	√
Order-aware			√	√	√	√
Weighted Action-aware					√	√
Dependency Path Patterns- aware						√

DSOD Variations + Features

Features	Simple DSOD	Obj-DSOD	Ops-DSOD	HDSOD	TCE	DSOD in PBAC
Per Role	√	√	√	√	√	√
Per Action		√	√	√	√	√
Per Object		√		√	√	√
Task-aware				√	√	√
Order-aware			√	√	√	√
Weighted Action-aware					√	√
Dependency Path Patterns- aware						√
Past Attribute-aware						√

Provenance Data

- Information of operations/transactions performed against data objects and versions
  - Actions that were performed against data
  - Acting Users/Subjects who performed actions on data
  - Data Objects used for actions
  - Data Objects generated from actions
  - Additional Contextual Information of the above entities

- Directed Acyclic Graph (DAG)
- Causality dependencies between entities (acting users / subjects, action processes and data objects)
- Dependency graph can be traced for the discovery of Origin, usage, versioning info, etc.

Provenance-aware Systems

- Capturing provenance data
- Storing provenance data
- Querying provenance data

Provenance Data Model

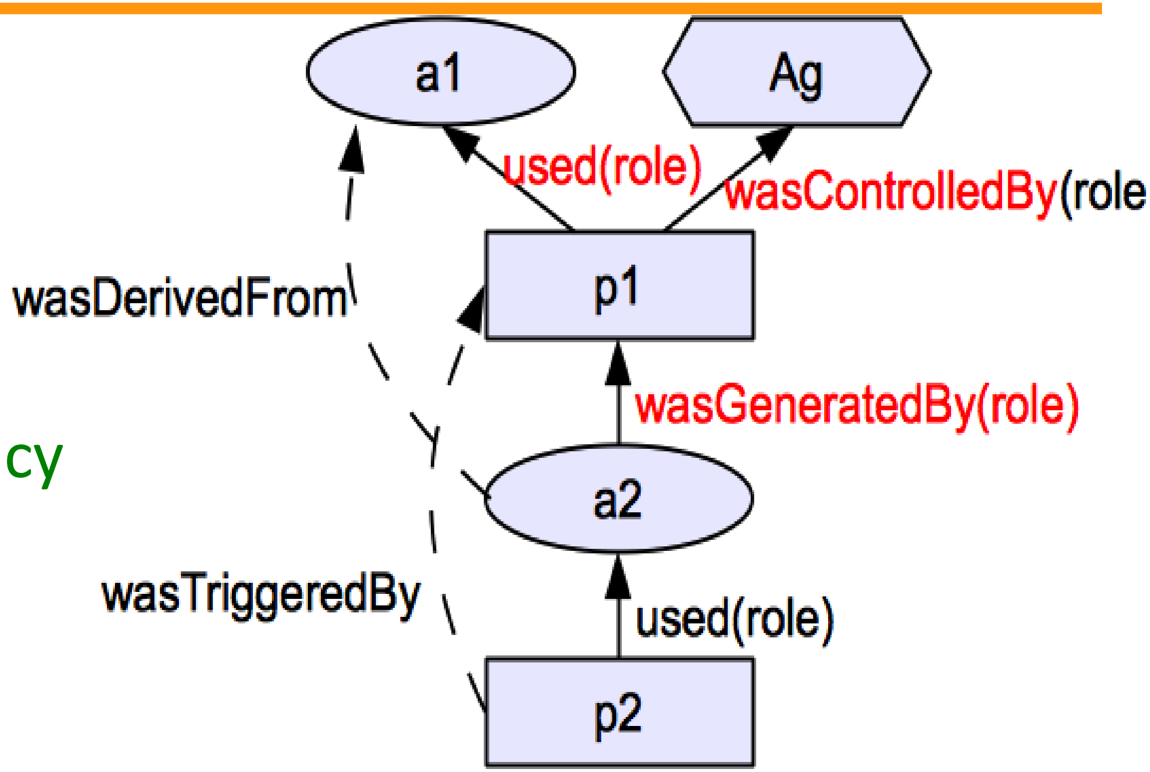
- Using provenance data
- Securing provenance data

Access Control

From Open Provenance Model (OPM)

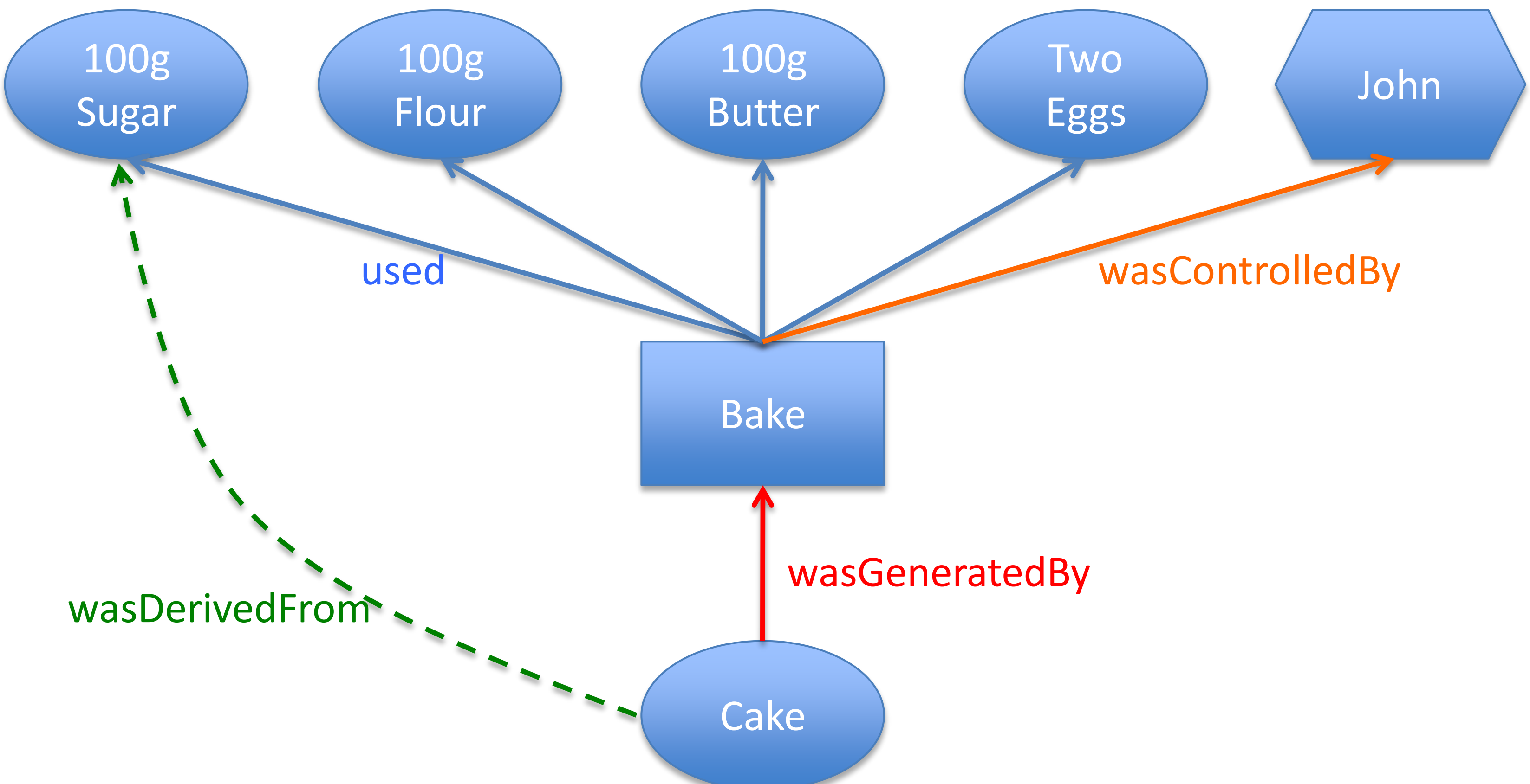
- 3 Node Types
  - Object (Artifact)
  - Action (Process)
  - User/Subject (Agent)

- 5 Causality dependency edge Types (not a dataflow)

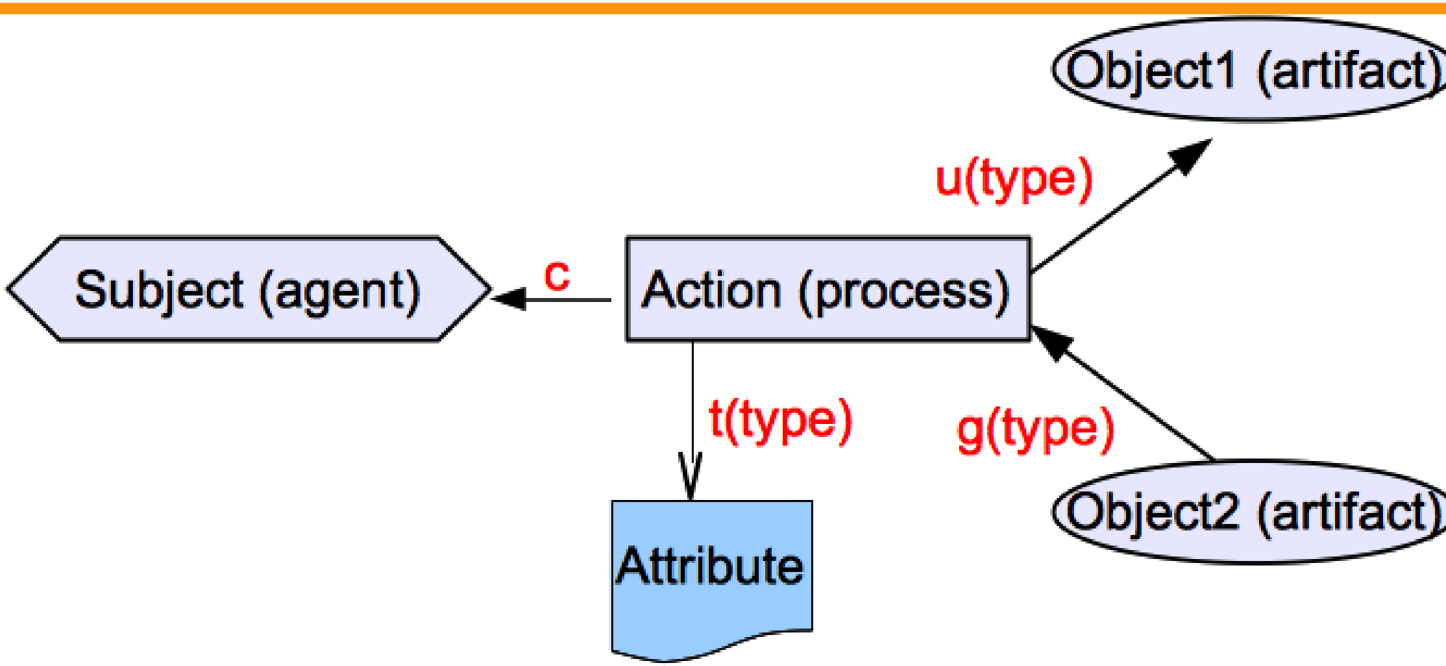


- Provenance data: a set of 2 entities & 1 dependency
  - E.g., (ag,p1,a1,a2): <p1,ag,c>,<p1,a1,u>,<a2,p1,g>

OPM Example


100g
Sugar
100g
Flour
100g
Butter
Two
Eggs
John
used
wasControlledBy
Bake
wasGeneratedBy
wasDerivedFrom
Cake

Provenance Data Model



c wasControlledby
u used
g wasGeneratedBy
t hasAttributeOf

Attrb. edge
Dep. edge

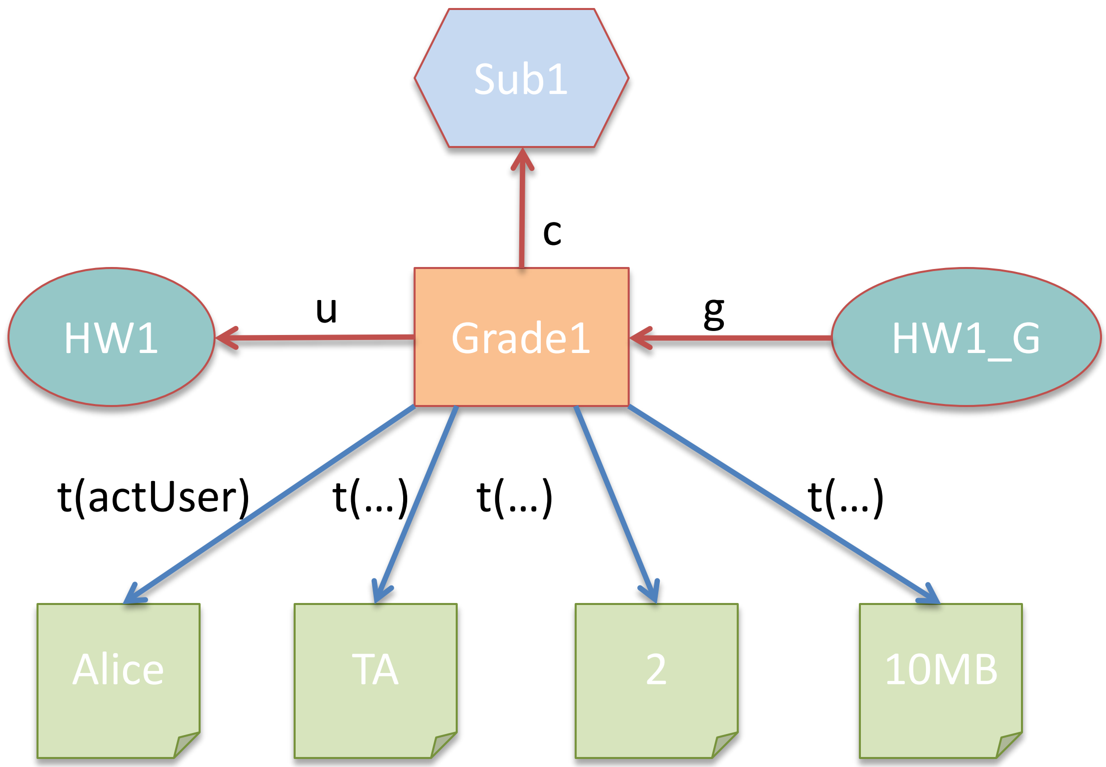
Capturing Provenance Data

(Subject1, Grade1, HW1, GradedHW1, ContextualInfoSet-Grade1)

(Grade1, u, HW1)
(Grade1, c, Subject1)
(GradedHW1, g, Grade1)

(Grade1, t[actingUser], Alice)
(Grade1, t[activeRole], TA)
(Grade1, t[weight], 2)
(Grade1, t[object-size], 10MB)

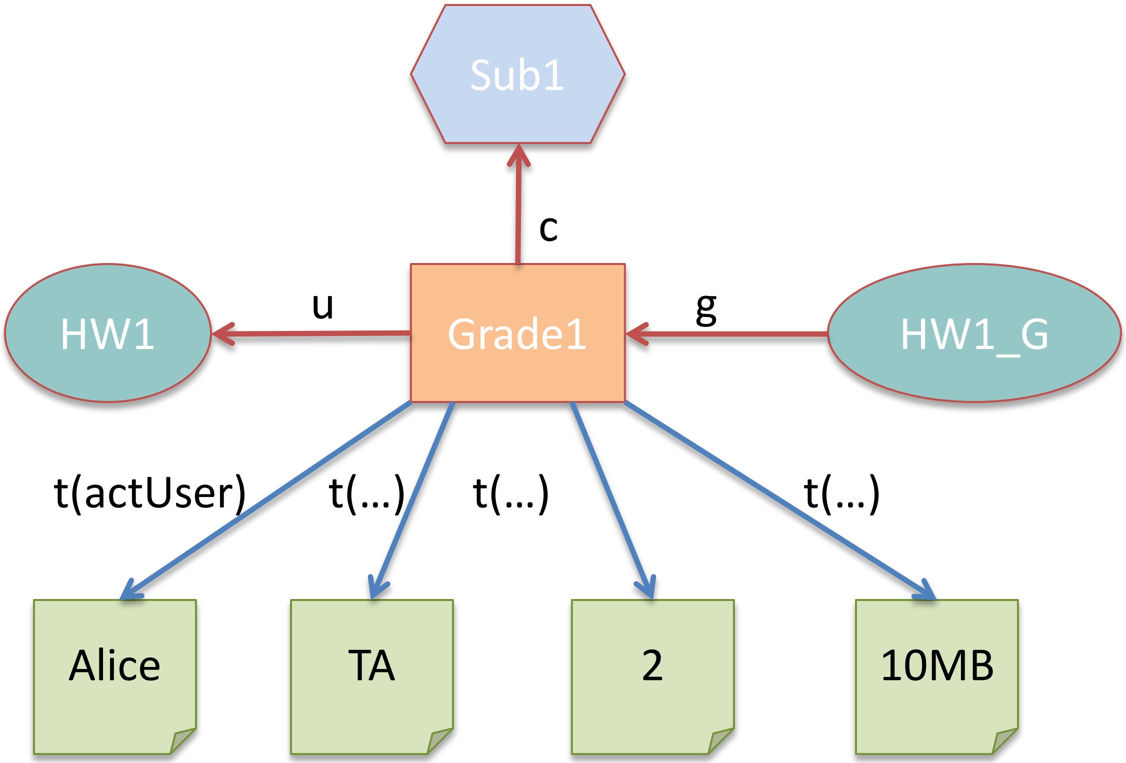
Provenance Graph



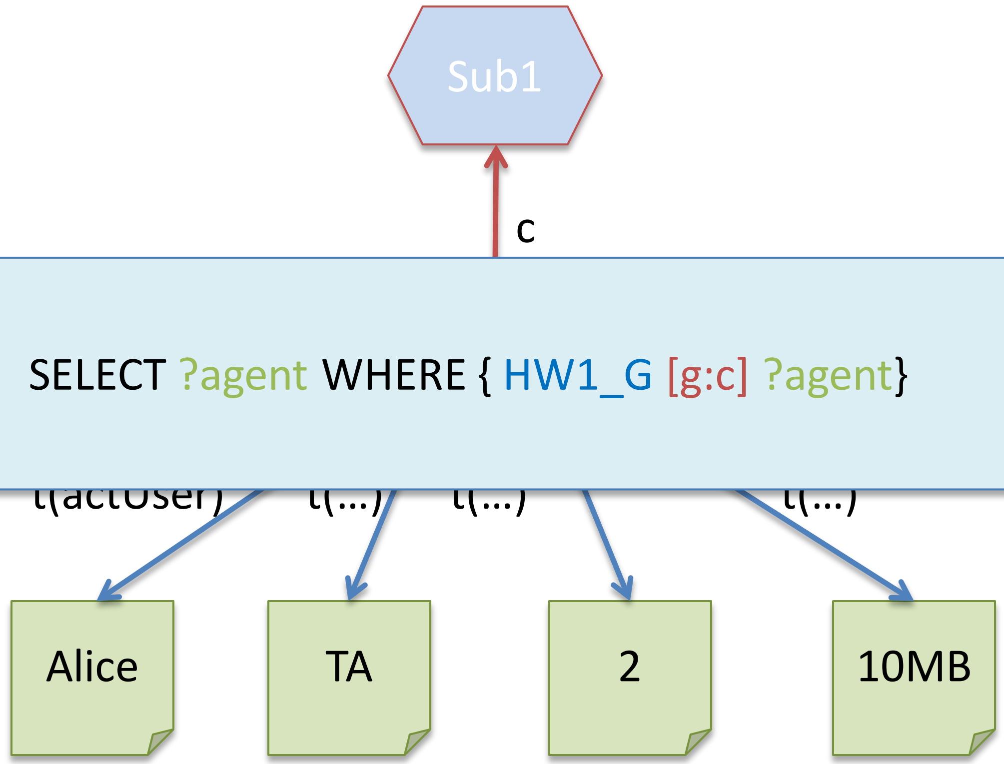
Storing and Querying Provenance Data

- Resource Description Framework (RDF) provides natural representation of triples.
- RDF-format triples can be stored in databases.
- Utilizes SPARQL Protocol and RDF Query Language for extracting useful provenance information.
  - Starting Node: any entities (not attribute nodes)
  - A matching path pattern: combination of dependency edges

Provenance Graph

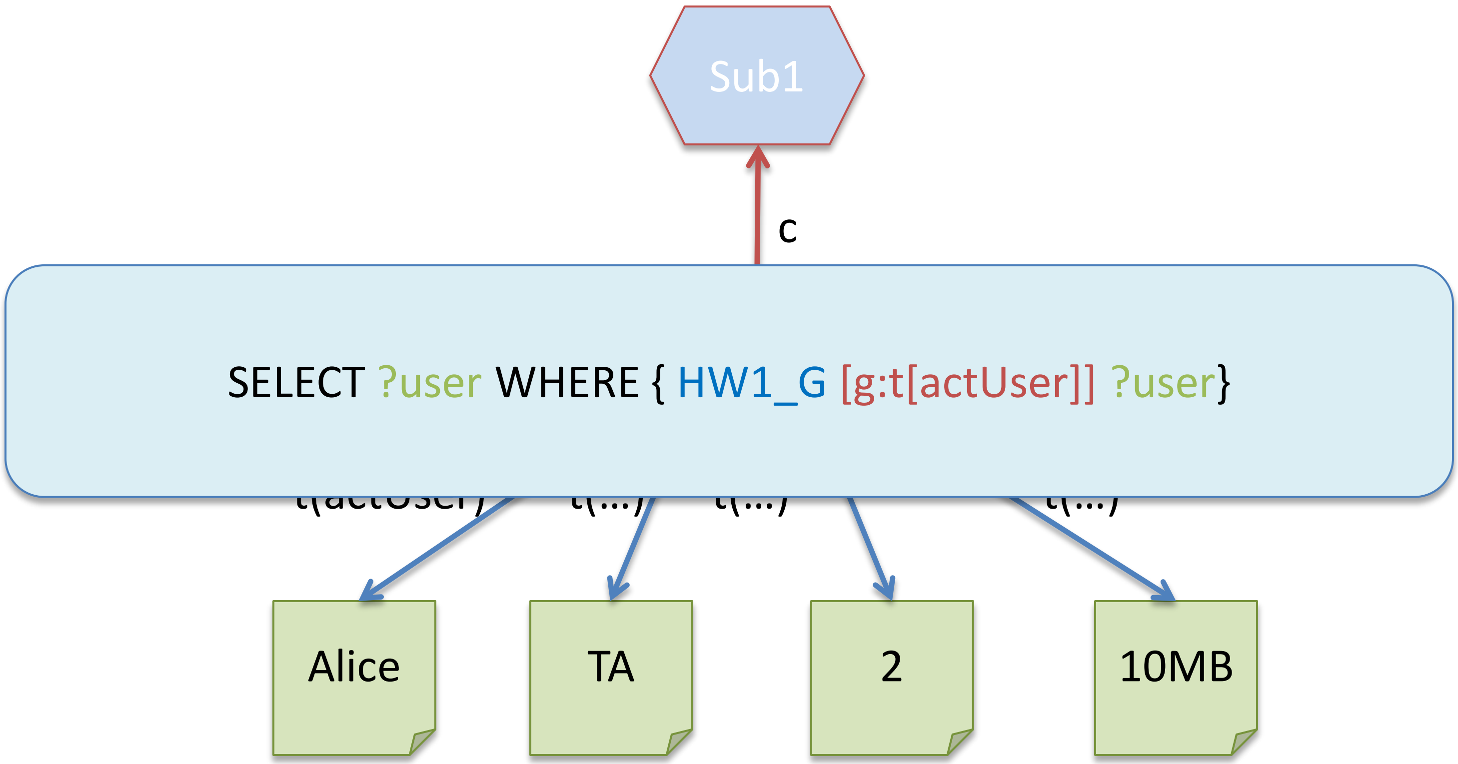


Provenance Graph

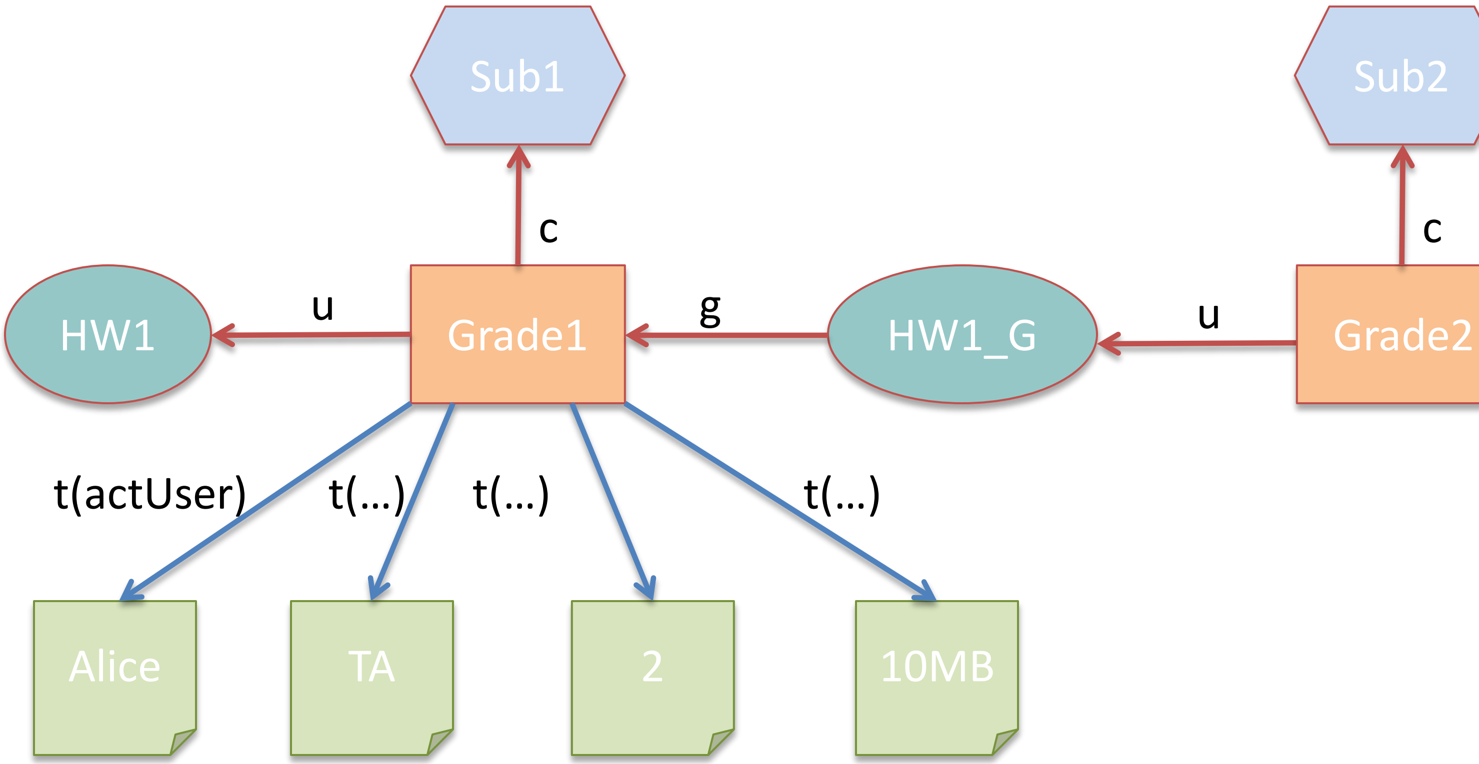


SELECT ?agent WHERE { HW1_G [g:c] ?agent}

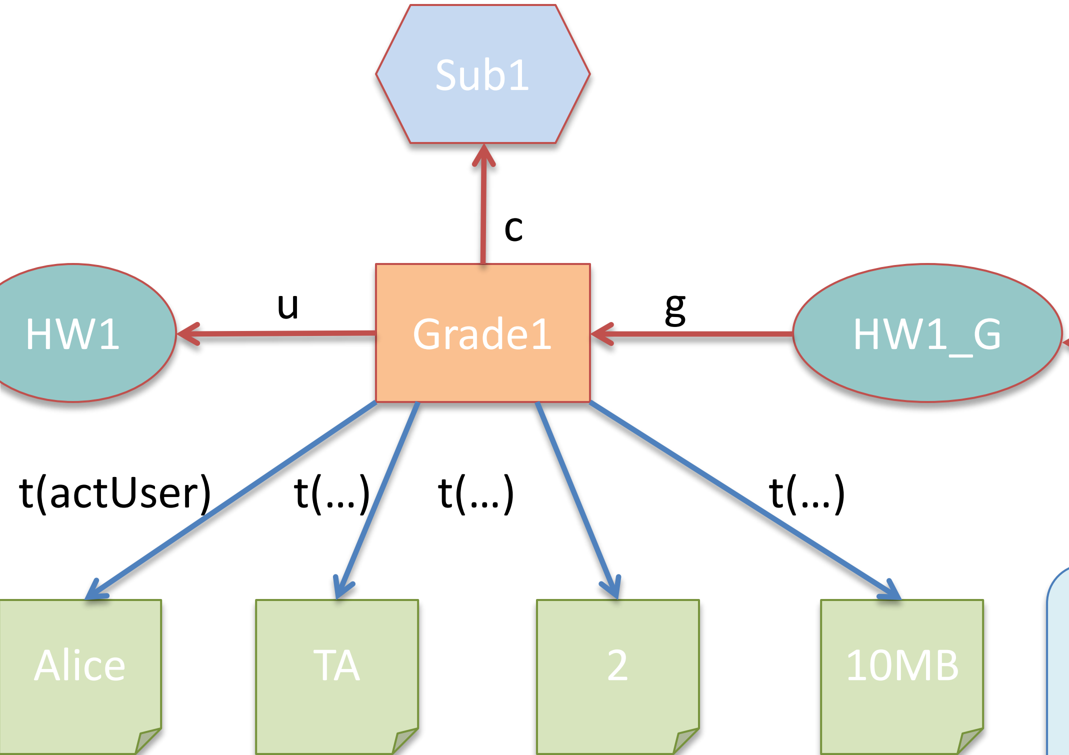
Provenance Graph



Provenance Graph



Provenance Graph



SELECT ?user
WHERE
{ HW1_G’ [g:u:g:c] ?user}

{ HW1_G’ ***[[g:u]*:g:c]*** ?user}

Provenance-aware Systems

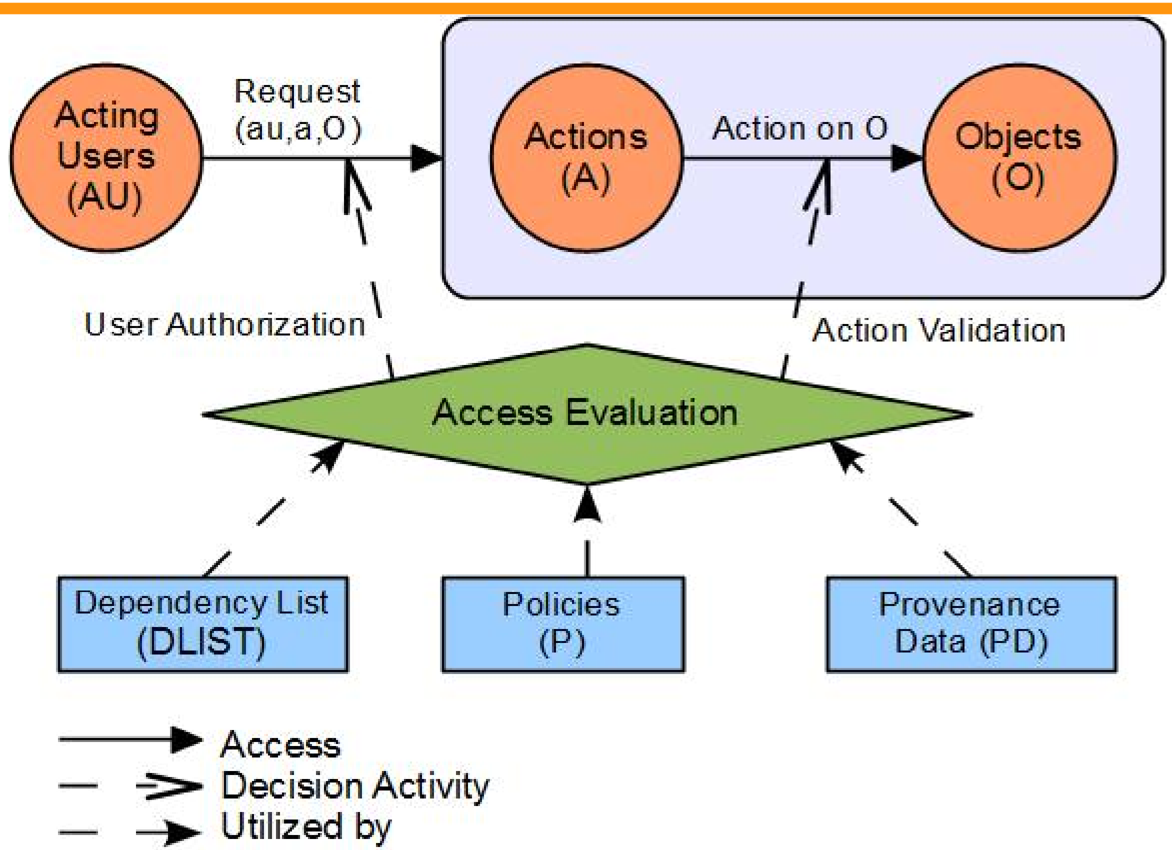
Using provenance data

PBAC

PAC

Securing provenance data

PBAC Model Components



Dependency List

- Object Dependency List (DL_O): A set of identified dependencies that consists of pairs of
  - Dependency Name: abstracted dependency names (DNAME) and
  - regular expression-based object dependency path pattern (DPATH)

- System-computable (complex) dependency instances
  - using pre-defined dependency names and matching dependency path patterns in DL (and querying base provenance data)
- User-declared (complex) dependency instances
  - using pre-defined dependency names in DL

- Examples
  - < wasSubmittedVof, $g_{submit}.u_{input}$ >
  - < wasAuthoredBy, wasSubmittedVof?.wasReplacedVof $*.g_{upload}.c$ >

$PBAC_B$: A Base Model

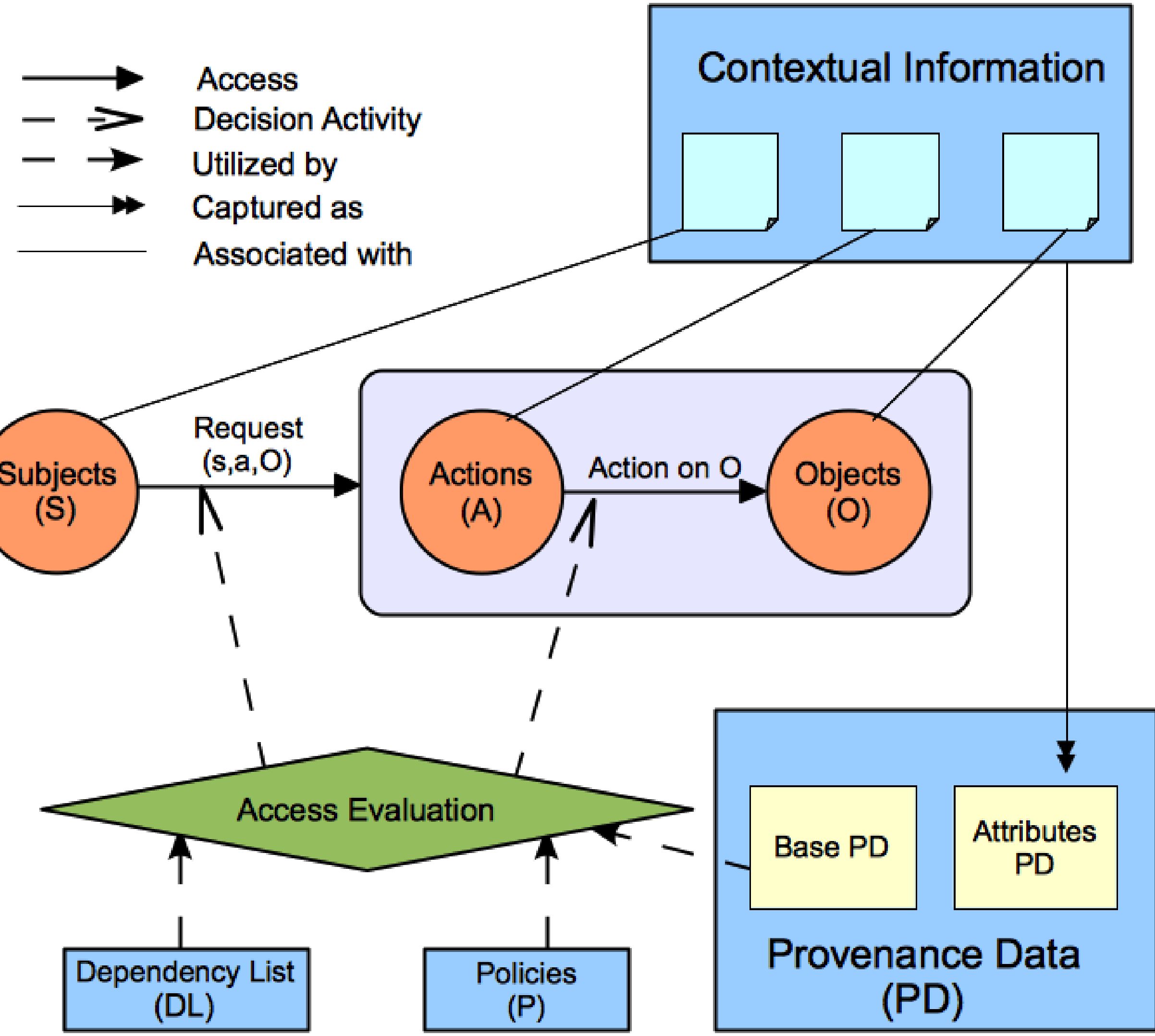
- System-captured Base Provenance Data only
  - Using sub-types of 3 direct dependencies (u, g, c)
  - No user-declared provenance data
- Object dependency only
- Supports Simple and effective policy specification and access control management
- Supports DSOD, workflow control, origin-based control, usage-based control, object versioning, etc.

Limitations of $PBAC_B$

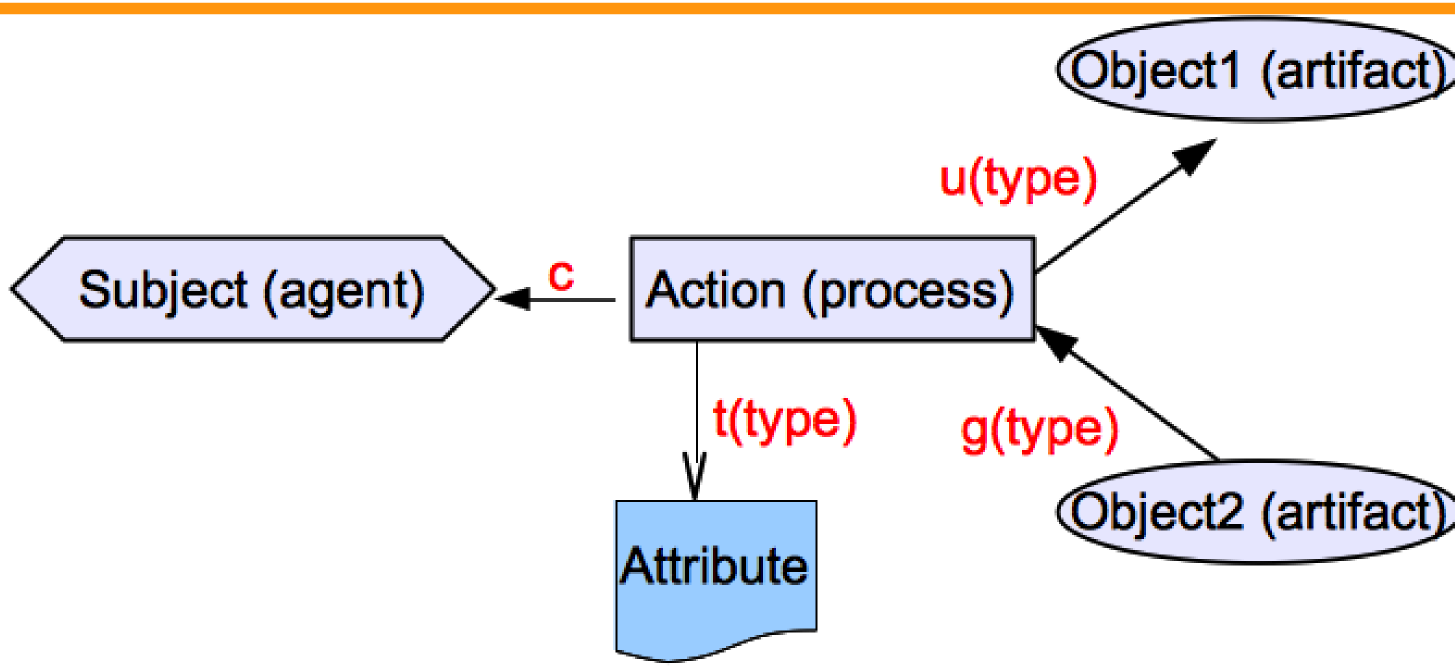
- Simplified data model
  - Does not capture contextual information
  - Unable to address advanced DSOD
  - Access evaluation restrained to User Verification and Action Validation

- $PBAC_C$: extending the base model

$PBAC_C$: $PBAC_B$ + Contextual Info.



Provenance Data Model



c wasControlledby
u used
g wasGeneratedBy
t hasAttributeOf

→ Attrb. edge
→ Dep. edge

Provenance Data Model

- A new type of entity, **Attribute**, to capture all contextual information.
- A new type of edge (can be considered dependency), t, that connects an entity and the associated attribute.
- Notice all attribute types (regardless of association) is concentrated in **Action** entities.
  - Action instances define system events.

$PBAC_C$: $PBAC_B$ + Contextual Info.

- Introduce **Subject** entities
- Incorporate *contextual information* associated with the main entities (**Users**, **Subjects**, etc.)
- Enable more variations of *dependency*
- *Access evaluation* now utilizes attributes
- Enable enhanced ***traditional*** and ***new*** features of DSOD
- More **flexible policy specification** (startNode = (S, A, or O))

Enhanced DSOD Features

- Awareness of Past-Action attribute.
  - Context information of action varies in different states in time
  - Past context information may potentially be significant for current state
  - Example: policy can specify decision rules based on either past or current assigned weight to action types

- Dependency Path Pattern-based DSOD.
  - More expressive control units
  - Can achieve wide variety of path patterns
  - Combinations of actions, versioning, etc.

Policies

- An *informal policy language* is used to specify access decision rules based on dependency name control units
- Example ObjDSOD:
  - **English Policy**: requires the requesting subject on replacing a homework object to be activated by the same acting user who activated the subject on uploading it.
  - **Informal Policy**:

    allow(sub,replace,o) =>
    (sub,hasPerformedActions:hasAttributeOf(actingUser)) ϵ (o,wasUploadedBy) *and* count(o,wasSubmittedVof) = 0.
- Smooth conversion to XACML policy language
  - Can be easily enforced
  - A proof-of-concept prototype is implemented

Sample XACML policy

```
<Policy PolicyId="replacePolicy"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rulecombining-
algorithm:ordered-permit-overrides">
<Target>
...
<Actions>
    <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0 :function:string-equal">
    <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">replace</AttributeValu
    e>
    <ActionAttributeDesignator
    AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id“
    DataType="http://www.w3.org/2001/XMLSchema#string" />
    </ActionMatch>
    </Action>
</Actions>
```

Sample XACML policy

…

<Rule RuleId="ReplaceRule" Effect="Permit">

<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">

<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">

<Apply FunctionId="**provenance-query-SPARQL">**

<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">

<SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"DataType="http://www.w3.org/2001/XMLSchema#string" /></Apply>

<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">**hasPerformedActions:hasAttributeOf(actingUser)**

</AttributeValue>

</Apply>

<Apply FunctionId="**provenance-query-SPARQL">**

<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">

<ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resourceid" DataType="http://www.w3.org/2001/XMLSchema#string" />

</Apply>

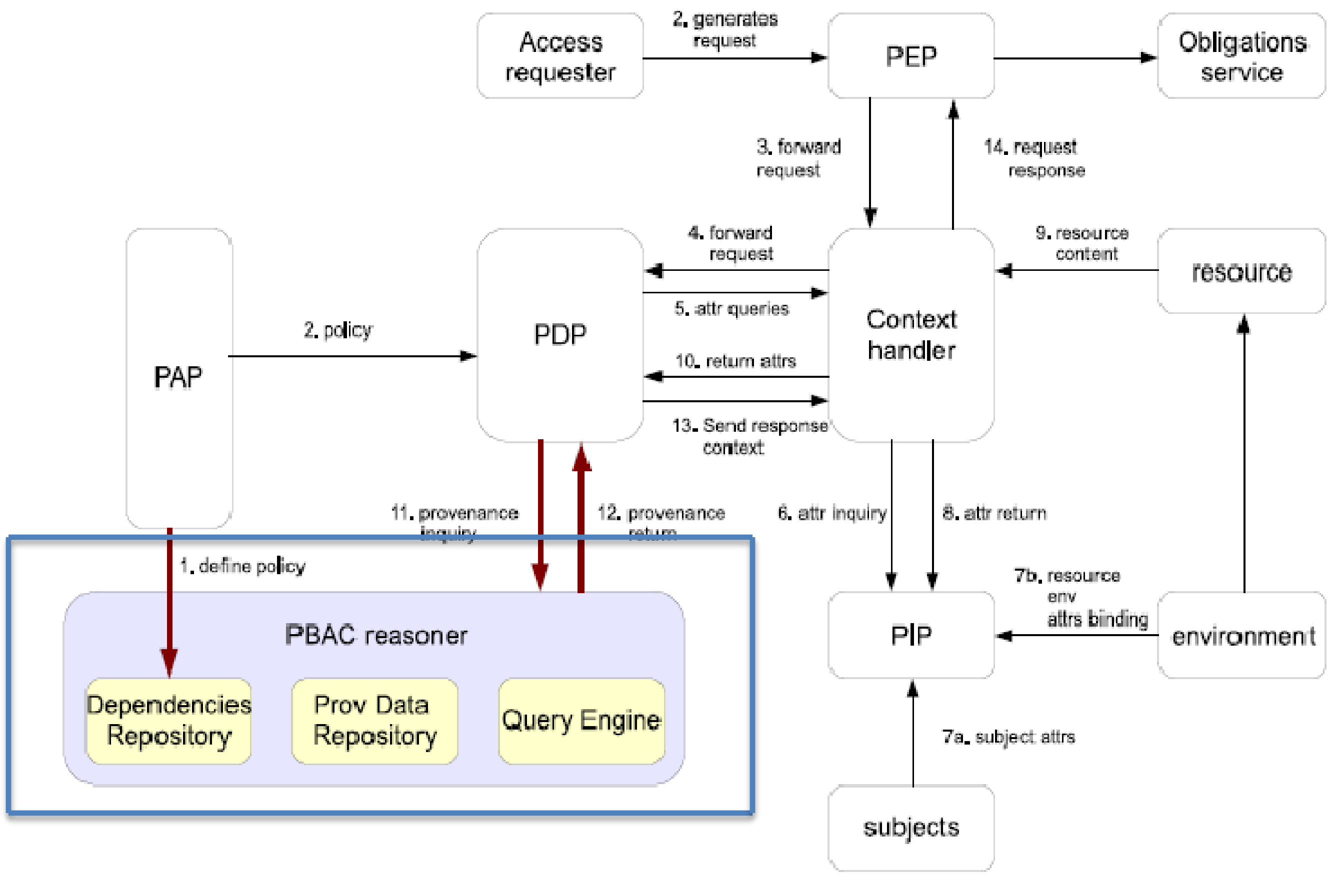
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">**wasUploadedBy**</AttributeValue>

</Apply>

</Apply>

…

Extended XACML Architecture



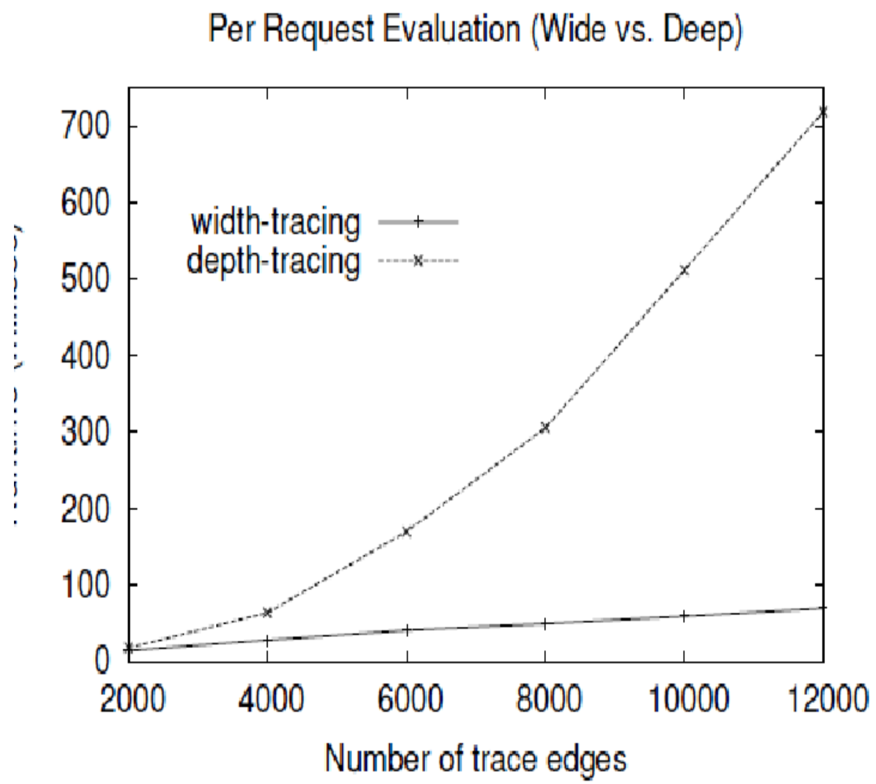
PBAC Reasoner Implementation

- Dependency Repository ← MySQL
- Provenance Data Repository ← Jena
- Query Engine ← ARQ

- *Extend OASIS XACML*
  - *Utilize top-of-the-shelf toolkits*

Experiment and Performance

- System
  - Ubuntu 12.10 image with 4GB Memory and 2.5 GHz quad-core CPU running on a Joyent SmartData center (ICS Private Cloud).
- Mock Data simulating HGS scenario
  - Different shapes of provenance graph
  - Extreme depth and width settings
- Results for tracing 2k/12k edges
  - 0.017/0.718 second per deep request
  - 0.014/0.069 second per wide request



Throughput Evaluation

- Results for tracing 2k/12k edges
  - 0.0096/0.154 second per deep request
  - 0.035/0.04 second per wide request

FEASIBLE !!!

Conclusion

- Propose a PBAC approach for traditional and enhanced DSOD variations
- Extend the base PBAC model to capture contextual information
- Proof-of-concept prototype on XACML architecture extension
- An access control foundation for secure provenance computing!

Thank you!!!

- Questions and Comments?