Engineering Access Control Policies for Provenance-aware Systems

Lianshan Sun[12], Jaehong Park[2] and Ravi Sandhu[2]

1. Shaanxi University of Science and Technology (SUST), Xi'an, Shaanxi, China, 710021
2. University of Texas at San Antonio (UTSA), San Antonio, Texas, USA, 78249
sunlianshan@gmail.com, jae.park@utsa.edu, ravi.sandhu@utsa.edu

Engineering access control policies for provenance-aware systems

- Background
  - What is provenance
  - Provenance-aware systems
  - Provenance-aware access control policies
- Motivations
- Solution and Case Study
  - Typed Provenance Model (TPM)
  - A TPM-Centric Process for engineering Access Control Polices
  - A case study on Homework Grading System (HGS)
- Conclusion

What is provenance

- Provenance is information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness.

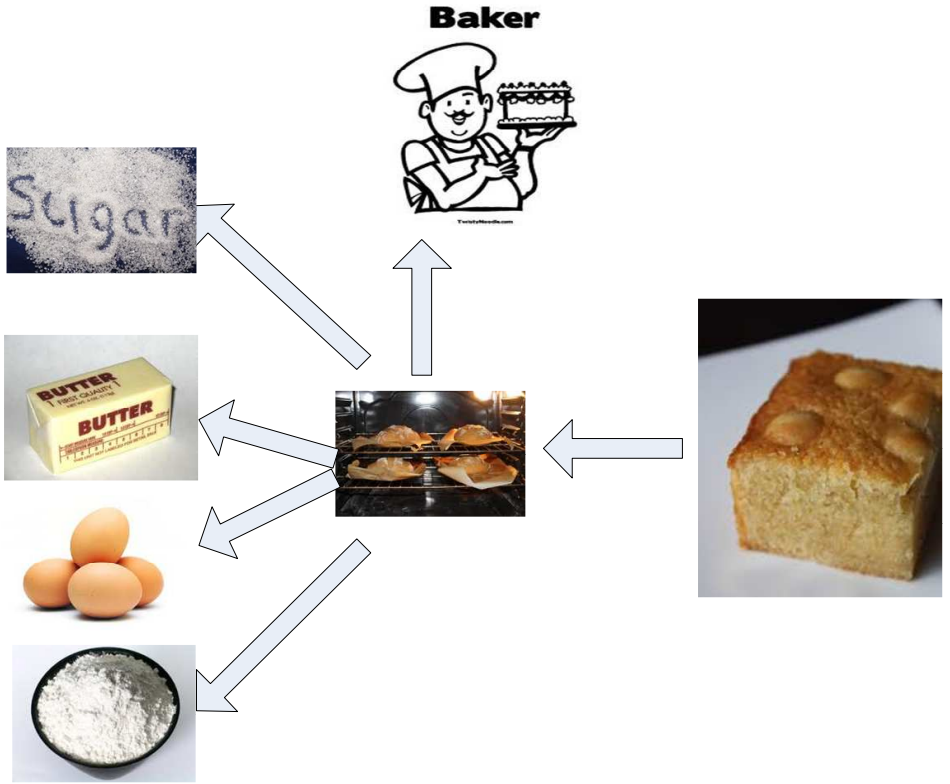

Figure: The provenance of a piece of cake

A Running Example – Homework Grading System (HGS)

- Students upload, replace, and submit their homework;
- Professors as well as some students on behalf of professors review the submitted homework;
- Professors grade a homework to generate a grade report having some of existing reviews of the homework as appendix.

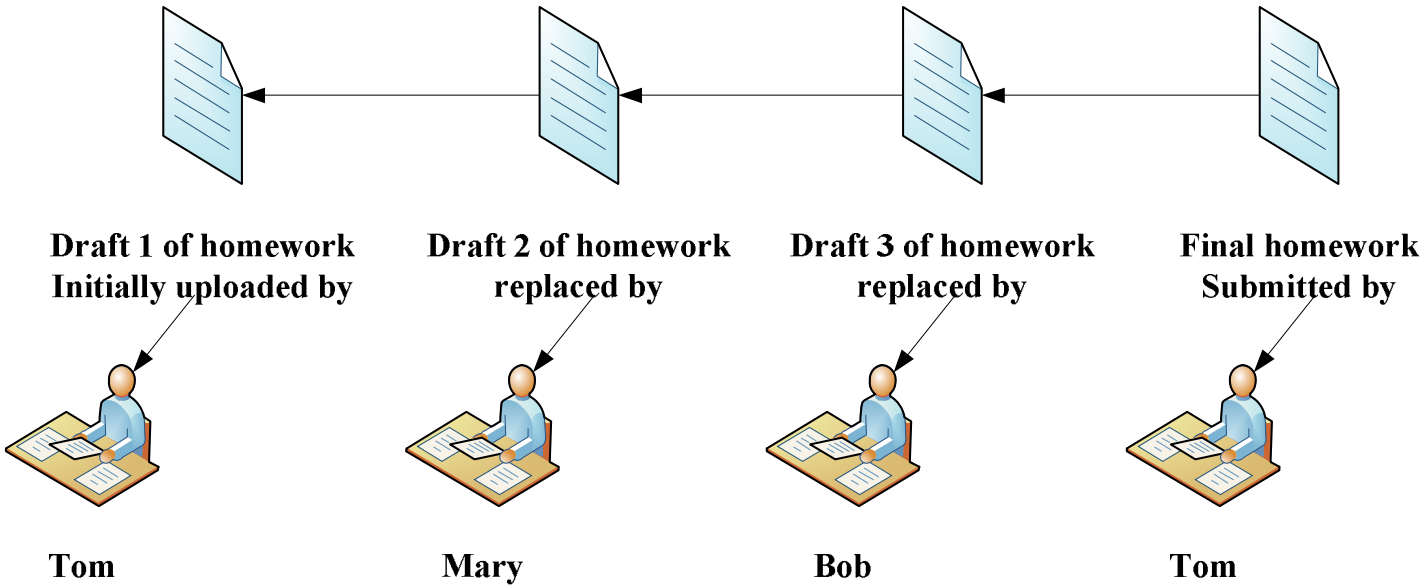

Figure: The provenance of a submitted homework.

Provenance-aware systems

- A provenance-aware system generates, stores, processes, and disseminates provenance to answer various provenance questions.
  - Key issues in building provenance aware systems include provenance collection, storage, and retrieval.
  - A provenance data model defines the scheme of provenance to be captured and is the conceptual basis of building provenance aware systems.
- A public provenance data model – Open Provenance Model (OPM).
  - A directed graph captures entities and casuality dependencies among entities.
  - Entities: artifact, process, agent.
  - Casuality dependency : $e \rightarrow f$ means e is caused by f.
  - Dependency types: direct (u, g, c), indirect (d, t).

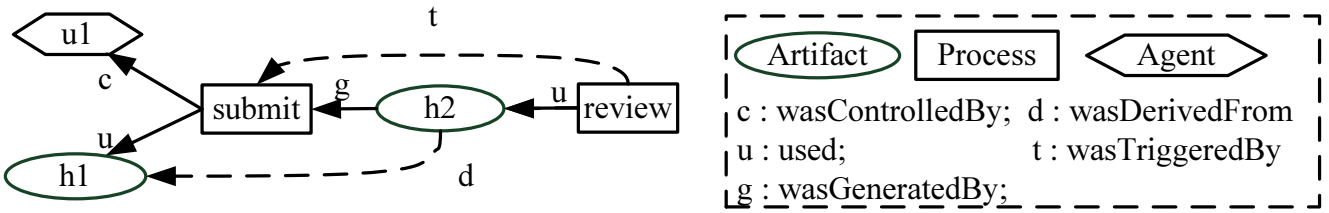

Figure: An OPM graph.

Access control in provenance-aware systems

- Provenance-aware systems need to deploy some access control facilities to protect both normal data items and their provenance.
- Provenance differs from traditional data and meta-data in that it is an immutable directed acyclic graph called provenance graph and can only be captured at run-time.
- Some subgraphs of a provenance graph as a unit may show meaningful provenance semantics and could be treated as sensitive resources or be used to adjudicate access requests.

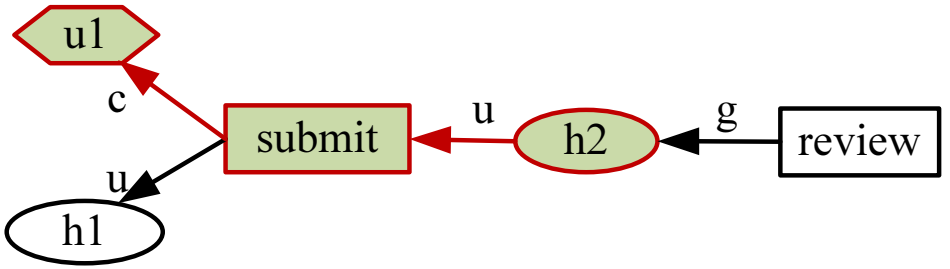

Figure: A subgraph of provenance.

Access control in provenance-aware systems

- Traditional access control models, policy languages do not work well in provenance aware systems.
- Researchers have proposed some provenance-aware access control models and corresponding policy languages.
  - Provenance access control, **PAC**
    - Protecting sensitive provenance.
    - A reviewer cannot see who has submitted a homework. prov: $(h \rightarrow submit \rightarrow u)$.
  - Provenance-based access control, **PBAC**
    - Protecting both sensitive provenance and sensitive data items with provenance by using provenance to adjudicate access requests.
    - Only a submitted homework can be reviewed. prov: $(h \rightarrow submit)$

Provenance-aware Access Control Policies

- A provenance-aware policy may be either a PAC policy, a PBAC policy, or the combination of both, which may refer to provenance answering certain provenance questions
  - A user u can see the owner of a homework h if u has started to grade h.
  - $u \in GradedBy(h) \quad \Leftarrow \quad \mathbf{P}(u, query, OwnedBy(h))$.
- Here, both $GradedBy(h)$ and $OwnedBy(h)$ are two provenance questions against the homework h whose semantics can be easily understood by users without technical knowledge.

Provenance-aware Access Control Policies

- A provenance-aware policy may be either a PAC policy, a PBAC policy, or the combination of both, which may refer to provenance answering certain provenance questions
  - A user u can see the owner of a homework h if u has started to grade h.
  - $u \in GradedBy(h) \quad \Leftarrow \quad \mathbf{P}(u, query, OwnedBy(h))$.
- Here, both $GradedBy(h)$ and $OwnedBy(h)$ are two provenance questions against the homework h whose semantics can be easily understood by users without technical knowledge.
- Although there are provenance-aware policy languages, it is far from straightforward for developers to specify provenance-aware policies due to various reasons.

Motivations

- First, it is very difficult to specify provenance-aware policies due to the complexity of provenance graph.
- For example, policy architects need to identify one or more subgraphs in a provenance graph in defining provenance-aware policies.
- $u \in GradedBy(h) \quad \Leftarrow \quad \mathbf{P}(u, query, OwnedBy(h))$

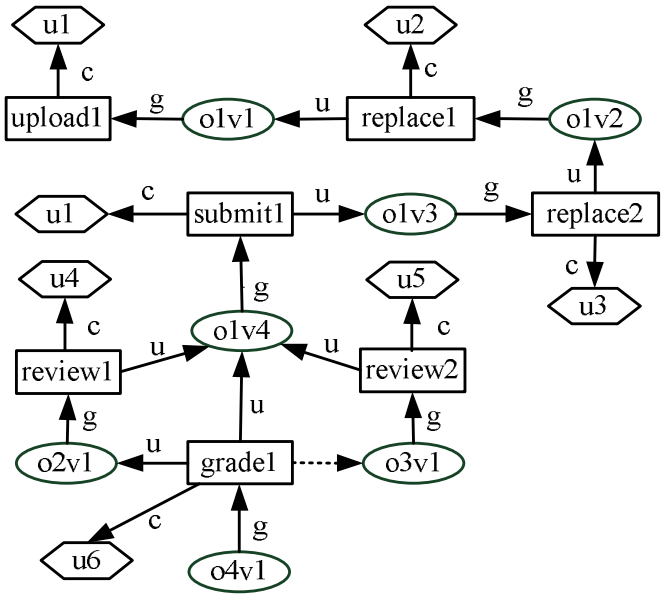

Figure: Provenance Graph of HGS.

Motivations

- First, it is very difficult to specify provenance-aware policies due to the complexity of provenance graph.
- For example, policy architects need to identify one or more subgraphs in a provenance graph in defining provenance-aware policies.
- $u \in GradedBy(h) \quad \Leftarrow \quad \mathbf{P}(u, query, OwnedBy(h))$

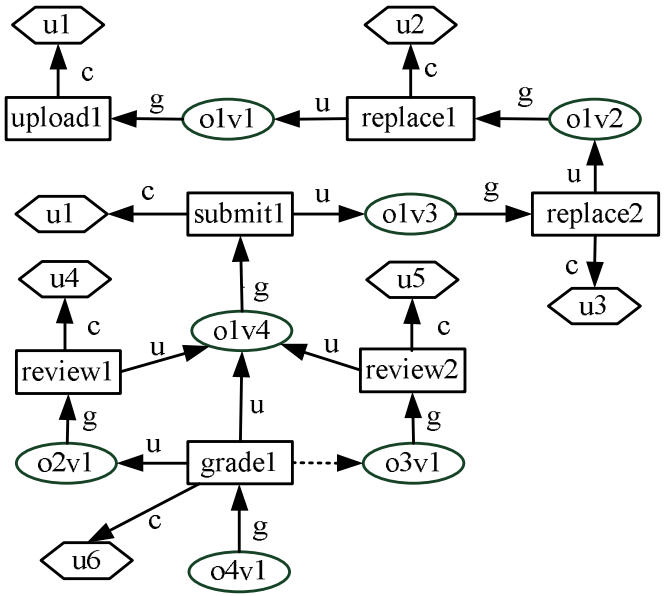

Figure: Provenance Graph of HGS.

We need some mechanisms to abstract complex provenance graph into user-comprehensible and meaningful controlling units that can be used to efficiently define provenance-aware policies at development time when the provenance graph is even not available.

Motivations

- Second, implications on software architecture
  - Provenance impacts software architecture and makes some traditional functional requirements possibly be implemented as provenance-aware policies.
    - An activity A can start only after another activity B is finished
    - Only users who did not review a homework before can review the homework.
  - Developers need to decide which requirements can be and should be modeled as provenance-aware requirements from the beginning of software development.

Motivations

- Second, implications on software architecture
  - Provenance impacts software architecture and makes some traditional functional requirements possibly be implemented as provenance-aware policies.
    - An activity A can start only after another activity B is finished
    - Only users who did not review a homework before can review the homework.
  - Developers need to decide which requirements can be and should be modeled as provenance-aware requirements from the beginning of software development.
- So it is conducive to take some engineering solutions in developing provenance-aware policies.
  - Modeling provenance in abstractions
  - Designing process to guide the identification, specification, and refinement of provenance aware policies.

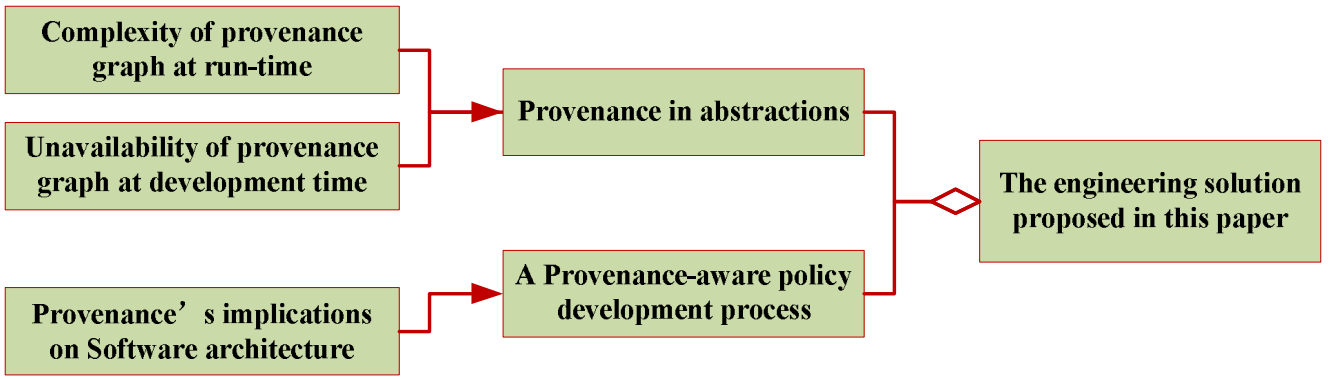

Figure: Motivations.

Typed Provenance Model

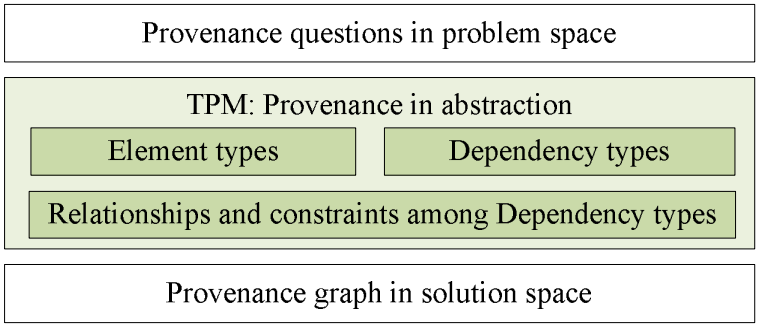

Figure: Provenance abstractions.

- An entity type is a class that is instantiated into nodes in a provenance graph
  - Artifacts: *Homework, Review, Grade*
  - Processes: *upload, replace, submit, review, grade*
  - Agents: *Student, Professor*
- A dependency type is a class of causality dependencies with similar provenance semantics
  - $T := N(E, C)$, e.g *T := ReviewedBy(Homework, User)*
  - $ReviewedBy(Hw_1, u_1)$ instantiated from T means that the homework Hw_1 was reviewed by the user u_1.
  - $ReviewedBy(Hw_1, u_1)$ can also be denoted as $u_1 \in ReviewedBy(Hw_1)$.
- Primitive dependency types and complex dependency types.

Primitive dependency types

- Each primitive dependency type abstracts the semantics of a set of edges in a provenance graph and could be from a process type to either an artifact type or an agent type, or from an artifact type to a process type.
- Primitive dependency types related to the same process type can be grouped together to form a process-centered directed graph of provenance in abstractions, called provenance type graph.

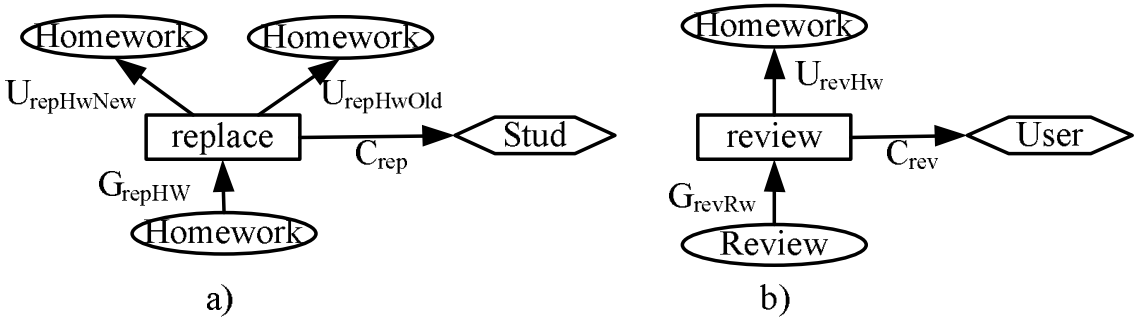

Figure: Primitive dependency types.

$$T_1 := U_{repHwOld}(rep, Hw), \qquad T_2 := U_{repHwNew}(rep, Hw),$$
$$T_3 := G_{repHW}(Hw, rep), \qquad T_4 := C_{rep}(rep, Stud).$$

Complex dependency types

- A complex dependency type encapsulates complex provenance semantics that cannot be carried by individual primitive dependency types.
- Each CDT can be defined as a composition of primitive dependency types.

The *concatenation* operator '$\cdot$'.

$$T_5 := G_{upHw}(Hw, up); T_6 := C_{up}(up, \mathit{Stud})$$
$$T_7 := UploadedBy(Hw, \mathit{Stud}) := T_5 \cdot T_6,$$

The *inversion* operation '$^{-1}$'.

$$T_8 := G_{revRw}(Rw, rev),$$
$$T_9 := U_{revHw}(rev, Hw),$$
$$T_{10} := C_{rev}(rev, \mathit{User}),$$
$$T_{11} := ReviewOn(Rw, Hw) := T_8 \cdot T_9,$$
$$T_{12} := {T_{11}}^{-1} := ReviewOn^{-1}(Hw, Rw)$$
$$:= {T_9}^{-1} \cdot {T_8}^{-1}.$$

The *regular expression* operators '$*|+|?$'.

$$T_{13} := G_{subHw}(Hw, sub), T_{14} := U_{subHw}(sub, Hw),$$
$$T_{15} := SubmisionOn(Hw, Hw) := T_{13} \cdot T_{14} \cdot (T_3 \cdot T_1)*.$$

The *conjunctive and disjunctive* operators '$\wedge|\vee$'.

$$T_{16} := ReplacedBy(Hw, \mathit{Stud}) := ((T_3 \cdot T_1)*) \cdot T_3 \cdot T_4;$$
$$T_{17} := SubmittedBy(Hw, \mathit{Stud}) := T_{13} \cdot C_{sub}(sub, \mathit{Stud});$$
$$T_{18} := OwnedBy(Hw, \mathit{Stud})$$
$$:= T_{17} \vee (T_{15}? \cdot T_{16}) \vee (T_{15}? \cdot (T_3 \cdot T_1)* \cdot T_7);$$
$$T_{19} := ReviewedBy(Hw, \mathit{User}) := {T_9}^{-1} \cdot T_{10};$$
$$T_{20} := ReviewedCOI(Hw, \mathit{User}) := T_{18} \wedge T_{19},$$

Composition constraints among dependency types

C_1 : A dependency type should has a unique name.

C_2 : A dependency type could be instantiated into multiple instances with not only unique identifiers but unique cause element or effect element;

C_3 : A dependency type can only defined on given element types;

C_4 : A dependency type T_a can concatenate with (using "$\cdot$" operator) another type T_b only if the cause element type of T_a is same to the effect element type of T_b;

C_5 : A dependency type can be combined with another type via the operator $\vee$ or $\wedge$ only if they have the same cause element type and effect element type.

C_6 : A dependency type cannot precede another type. In the HGS, a dependency type T_r denotes a homework was a replacement of another homework and T_s denotes a homework is a submitted version of another homework. $T_r \cdot T_s$ is syntactically right while semantically wrong because the $replace$ activity on a homework cannot be activated after the $submit$ process on the same homework happened.

A TPM-Centric Process for Engineering ACPs

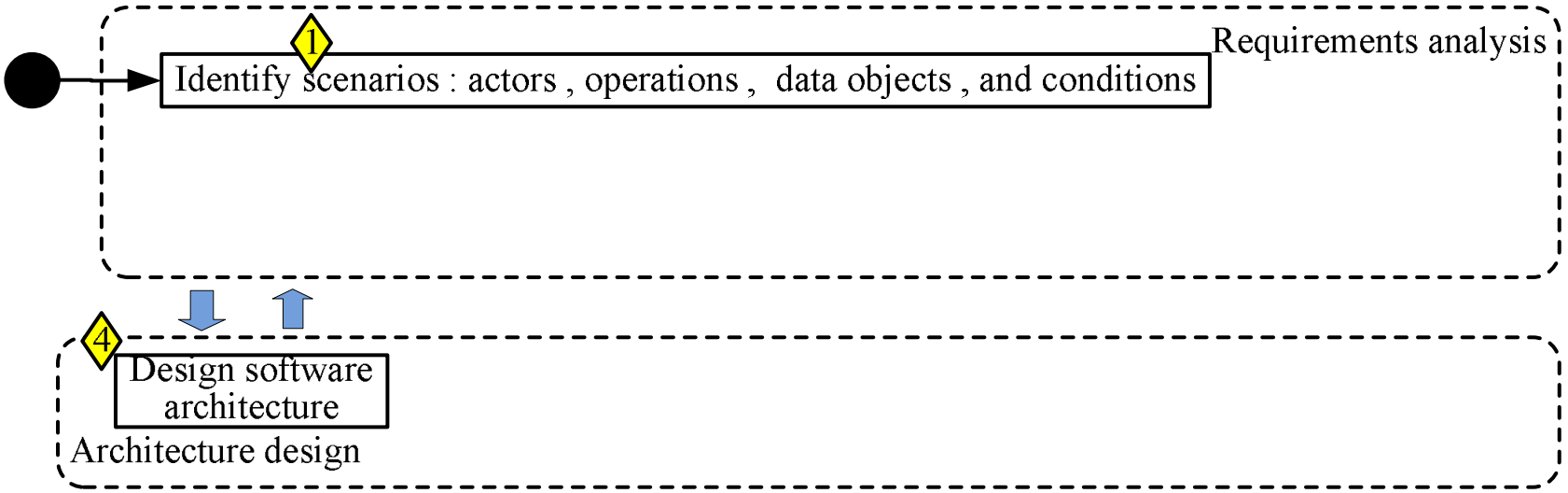

Figure: A TPM-Centric Process for Engineering ACPs.

A TPM-Centric Process for Engineering ACPs

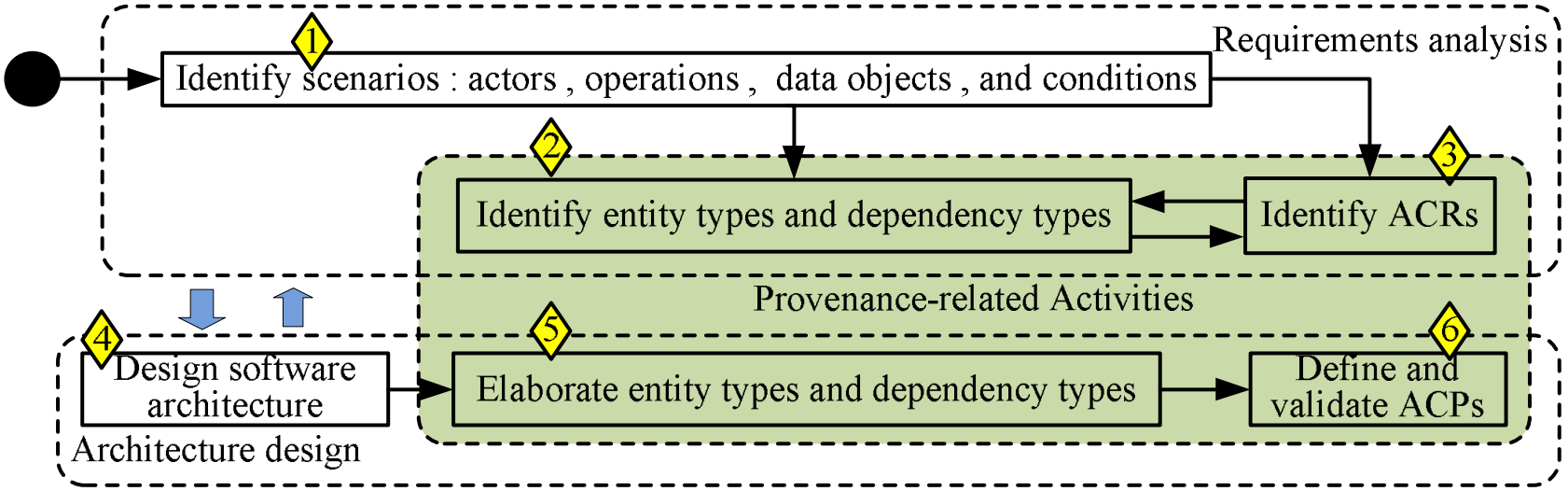

Figure: A TPM-Centric Process for Engineering ACPs.

- This process emphasizes the role of typed provenance model in identifying and specifying provenance-aware policies in overall software development process.
- This process itself is intuitive and does not promise to produce a good enough set of policies, which heavily depends on the expertise of policy architects.

Case Study

Table: A list of scenarios of the HGS.

No.	actor	operation	data objects	conditions
1.1	Student	upload	a **homework**	C0: Any student in the HGS
1.2	Student	replace/submit	a **homework** to get a new or submitted **homework**	C1: Owner of homework C2: Homework was non-submitted.
2	Student or Professor	review	a **homework** to produce a **review**	C3: (Not C1) Not Owner of the homework C4: (Not C2) The homework was submitted C5: Not reviewed the homework before C6: The homework was not graded C7: Number of reviews of the homework < 3
3	Professor	grade	a **homework** to produce a **grade**	C8: Number of reviews of the homework ≥ 2
4	Student	query-prov-of	a **homework** on its reviewed times and graded state	C9: (C1) Owner of the homework
5	Student or Professor	query-prov-of	a **review** as part of a grade	C10: Owner of the review
6	Professor	query-prov-of	a **homework** on its author, reviewers, and whether it is reviewed by a user involved in conflict of interests.	C11: Any professor in the HGS

Table: Entity types of the HGS.

ET	$:= UT \mid AT \mid PT.$
UT	$:= User \mid Stud \mid Prof.$
AT	$:= Homework \mid Review \mid Grade.$
PT	$:= upload \mid replace \mid submit$
	$\mid review \mid grade \mid$ query-prov-of.

Table: Dependency types from prov. questions.

No	Dependency types	Question (type description)
1	$U_{revHw}(rev, Hw)$	4 (review used Hw)
2	$U_{grdHw}(grd, Hw)$	4 (grade used Hw)
3	$U_{grdRw}(grd, Rw)$	5 (grade used Rw)
4	$ReviewedBy(Hw, User)$	6 (Hw was ReviewedBy user)
5	$OwnedBy(Hw, Stud)$	6 (Hw was OwnedBy stud)
6	$ReviewCOI(Hw, User)$	6 (Hw was Reviewedby owner)

Case study

Table: Dependency types from ACRs.

No	Dependency types used to specify	conditions in Table 1
1	$OwnedBy(Hw, Stud)$	C1, C3, C9
2	$G_{subHw}(Hw, sub)$	C2, C4
3	$ReviewedBy(Hw, User)$	C5
4	$U_{grdHW}(grd, Hw)$	C6
5	$ReviewOn(Rw, Hw)$	C7, C8
6	$ReviewOf(Rw, User)$	C10

$$T_{18} := OwnedBy(Hw, Stud)$$
$$:= T_{17} \vee (T_{15}? \cdot T_{16}) \vee (T_{15}? \cdot (T_3 \cdot T_1) * \cdot T_7);$$

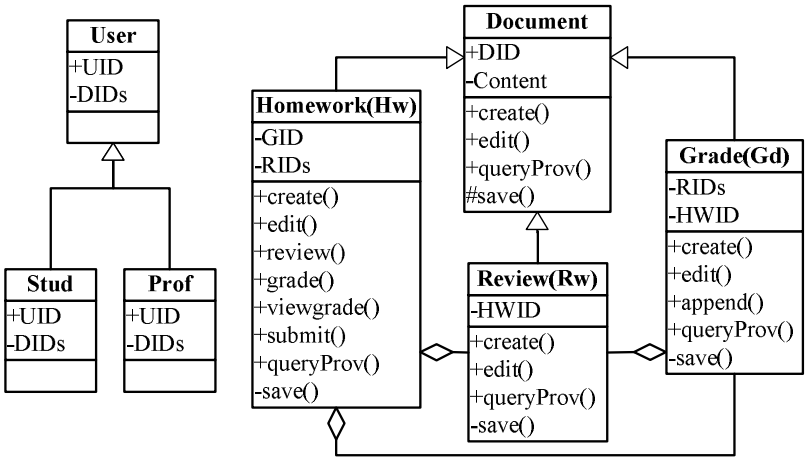

Figure: The class diagram of the HGS.

Table: Access Control Policies of HGS.

No.	Policies
1.2	$\forall u \forall h\ (u \in OwnedBy(h) \wedge\ Submittedby(h) = \phi\ ) \Leftarrow \mathbf{P}(u, replace/submit, h)$
2	$\forall u \forall h \left(\begin{array}{l} u \in OwnedBy(h)\ \wedge\ G_{subHW}(h) \neq \phi \wedge u \notin ReviewedBy(h) \wedge \\ {U_{grdHw}}^{-1}(h) = \phi \wedge \lvert ReviewOn^{-1}(h) \rvert < 3 \end{array} \right) \Leftarrow \mathbf{P}(u, review, h)$
3	$\forall u \forall h\ (\lvert ReviewOn^{-1}(h) \rvert \geq 2) \Leftarrow \mathbf{P}(u, grade, h)$
4	$\forall u \forall h\ (u \in OwnedBy(h)) \Leftarrow \mathbf{P}\left(u, \text{query-prov-of}, \{\lvert {U_{revHw}}^{-1}(h) \rvert, {U_{grdHw}}^{-1}(h)\}\right)$
5	$\forall u \forall r\ (u \in ReviewOf(r)) \Leftarrow \mathbf{P}(u, \text{query-prov-of}, {U_{grdRw}}^{-1}(r))$
6	$\forall u \forall h\ (u\ \in\ Prof \Leftarrow \mathbf{P}(u, \text{query-prov-of}, \{OwnedBy(h), ReviewedBy(h), ReviewedCOI(h)\}))$

Case study

Table: Dependency types from ACRs.

No	Dependency types used to specify	conditions in Table 1
1	$OwnedBy(Hw, Stud)$	C1, C3, C9
2	$G_{subHw}(Hw, sub)$	C2, C4
3	$ReviewedBy(Hw, User)$	C5
4	$U_{grdHW}(grd, Hw)$	C6
5	$ReviewOn(Rw, Hw)$	C7, C8
6	$ReviewOf(Rw, User)$	C10

$$T_{18} := OwnedBy(Hw, Stud)$$
$$:= T_{17} \vee (T_{15}? \cdot T_{16}) \vee (T_{15}? \cdot (T_3 \cdot T_1) * \cdot T_7);$$

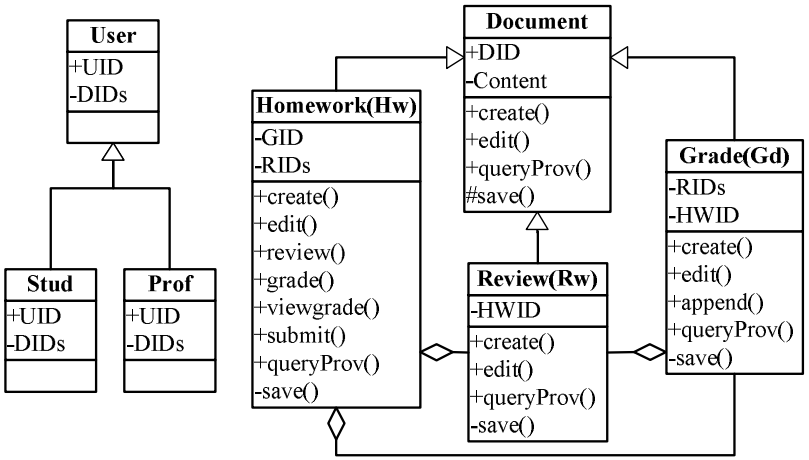

Figure: The class diagram of the HGS.

Table: Access Control Policies of HGS.

No.	Policies
1.2	$\forall u \forall h\ (u \in OwnedBy(h) \wedge\ Submittedby(h) = \phi\ ) \Leftarrow \mathbf{P}(u, replace/submit, h)$
2	$\forall u \forall h \left(\begin{array}{l} u \in OwnedBy(h)\ \wedge\ G_{subHW}(h) \neq \phi \wedge u \notin ReviewedBy(h) \wedge \\ {U_{grdHw}}^{-1}(h) = \phi \wedge \lvert ReviewOn^{-1}(h) \rvert < 3 \end{array} \right) \Leftarrow \mathbf{P}(u, review, h)$
3	$\forall u \forall h\ (\lvert ReviewOn^{-1}(h) \rvert \geq 2) \Leftarrow \mathbf{P}(u, grade, h)$
4	$\forall u \forall h\ (u \in OwnedBy(h)) \Leftarrow \mathbf{P}\left(u, query\text{-}prov\text{-}of, \{\lvert {U_{revHw}}^{-1}(h) \rvert, {U_{grdHw}}^{-1}(h)\}\right)$
5	$\forall u \forall r\ (u \in ReviewOf(r)) \Leftarrow \mathbf{P}(u, query\text{-}prov\text{-}of, {U_{grdRw}}^{-1}(r))$
6	$\forall u \forall h\ (u\ \in\ Prof \Leftarrow \mathbf{P}(u, query\text{-}prov\text{-}of, \{OwnedBy(h), ReviewedBy(h), ReviewedCOI(h)\}))$

Conclusion

- It is increasingly difficult to build an access control system in a specific application because access control itself is increasingly relying on the complex application-level concepts, such as role and provenance.
- Our research interest is to explore the engineering solutions in building access control systems on the basis of existing achievements in security field, including access control models, policy languages, and underlying enforcement mechanisms.
- Contributions of this paper.
  - Identified issues motivating the research on solutions to engineer access control policies in provenance-aware systems.
  - Introduced the typed provenance model to model semantically meaningful provenance for more efficiently defining and managing access control policies.
  - Designed a TPM-centric process to discipline the identification, specification, and refinement of access control policies.
  - Illustrated our achievements in a homework grading system.
- We are working on an enforcement architecture for provenance-aware policies specified using TPM.


Q
A