A PROV encoding for provenance analysis using deductive rules (Datalog)

Paolo Missier
Newcastle University, UK

Khalid Belhajjame
University of Manchester, UK

IPAW'12
Santa Barbara, CA, June 2012

The W3C PROV effort

- A set of specifications -- to be finalized by end of 2012

The W3C PROV effort

- A set of specifications -- to be finalized by end of 2012

W3C

PROV-DM: The PROV Data Model

Post internal release (WD6) (Diffs since last release)

W3C Editor's Draft 20 June 2012

W3C

The Provenance Notation

l release (Diffs since last release)

s Draft 20 June 2012

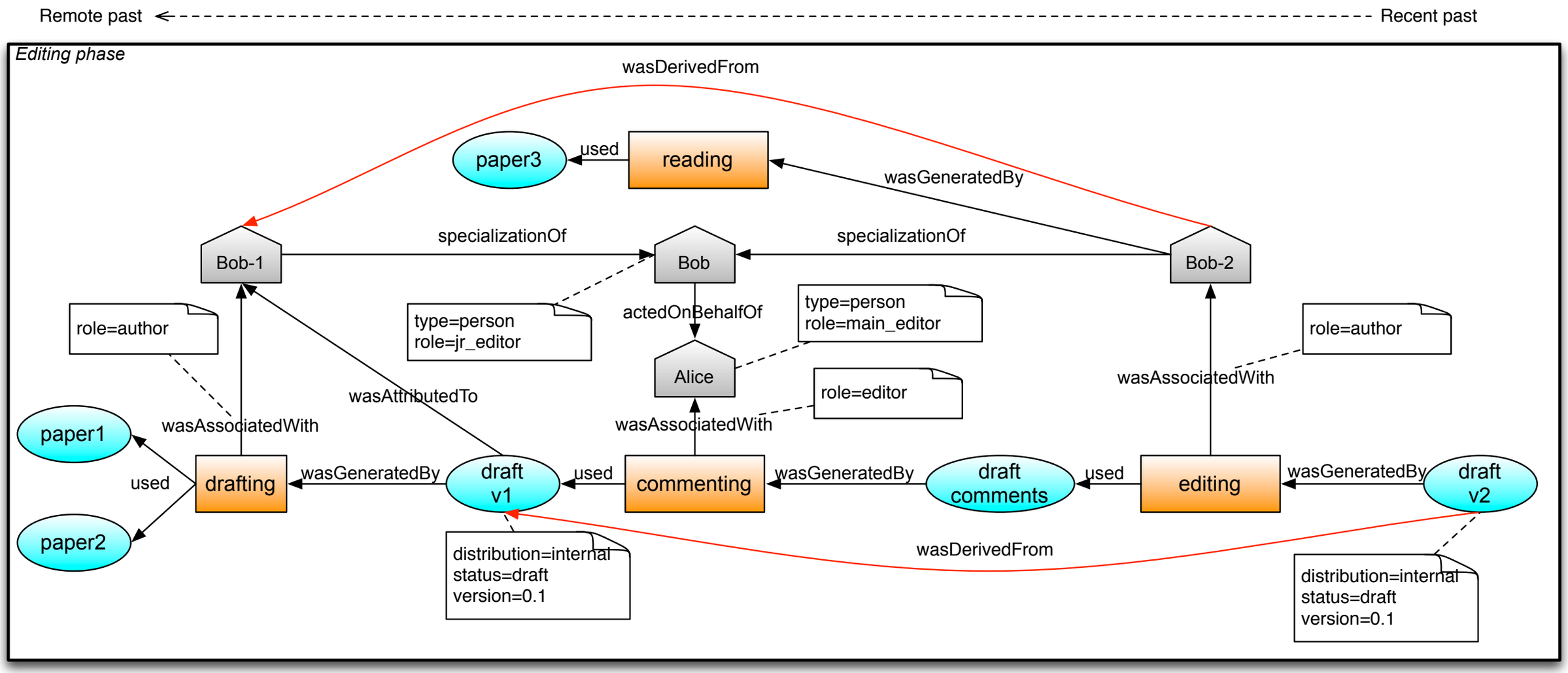
W3C

Constraints of the Provenance Data Model

towards second working draft

W3C Editor's Draft 20 June 2012

A PROVenance graph



PROV Notation:

```
entity(draftV1, ["distribution"="internal",
                 "status"="draft", "version"="0.1"])
entity(draftComments)
activity(commenting, comment_start, comment_end)
used(u1; commenting, draftV1, comm_d1_use)
wasGeneratedBy(g1; draftComments, commenting, comm_dc_gen)
```

PROV-N and Datalog encoding

PROV follows a relational data model

```
entity(draftV1, ["distribution"="internal",
                 "status"="draft", "version"="0.1"])
entity(draftComments)
activity(commenting, comment_start, comment_end)
used(u1; commenting, draftV1, comm_d1_use)
wasGeneratedBy(g1; draftComments, commenting, comm_dc_gen)
```

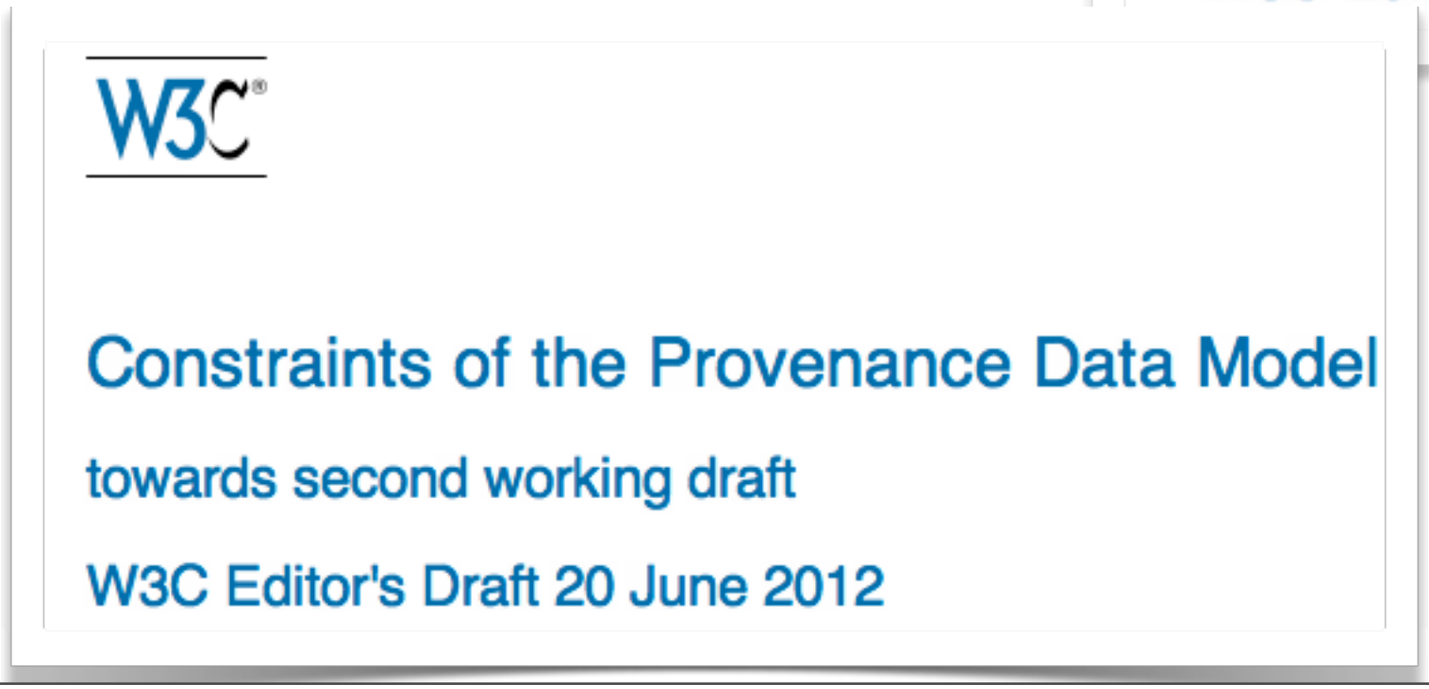
The corresponding Datalog EDB is straightforward:

```
entity(draftV1, draftV1Attrs).
attrList(draftV1Attrs, "distribution", "public").
attrList(draftV1Attrs, "status", "draft").
attrList(draftV1Attrs, "release", "1.0").
entity(draftComments, nil).
activity(commenting, comment_start, comment_end, nil).
used(commenting, draftV1, comm_d1_use, nil).
wasGeneratedBy(draftComments, commenting, comm_dc_gen, nil).
```

Parser implementation in the ProvToolbox (gitHub)

(Thanks to Luc Moreau for the master PROV-N parser code)

PROV Constraints



PROV Constraints

- PROV-N provides a syntax
- PROV comes with a set of rules for the semantics of the model
  1. deductive rules

Inference 9 (inference-trace)

Given two identifiers `e2` and `e1` for entities, the following statements hold:

1. IF `wasDerivedFrom(e2,e1,a,g2,u1)` holds, for some `a`, `g2`, `u1`, THEN `tracedTo(e2,e1)` also holds.
2. IF `wasDerivedFrom(e2,e1)` holds, THEN `tracedTo(e2,e1)` also holds.
3. IF `wasAttributedTo(e2,ag1,aAttr)` holds, THEN `tracedTo(e2,ag1)` also holds.
4. IF `wasAttributedTo(e2,ag2,aAttr)`, `wasGeneratedBy(-;e2,a,-,gAttr)`, and `actedOnBehalfOf(ag2,ag1,a,rAttr)` hold, for so
5. IF `wasGeneratedBy(e2,a,gAttr)` and `wasStartedBy(a,e1,sAttr)` hold, for some `a`, `gAttr` , `sAttr` then `tracedTo(e2,e1)` holds.
6. IF `tracedTo(e2,e)` and `tracedTo(e,e1)` hold for some `e`, THEN `tracedTo(e2,e1)` also holds.

2. constraints: they effectively define *consistent provenance*

Inference 14 (specialization-antisymmetric)

For any entities `e1`, `e2`, it is not the case that `specializationOf(e1,e2)` and `specializationOf(e2,e1)`.

---

note:
these constraints are still in flux at the time of this presentation

PROV constraints as Datalog rules

Goal of this work:

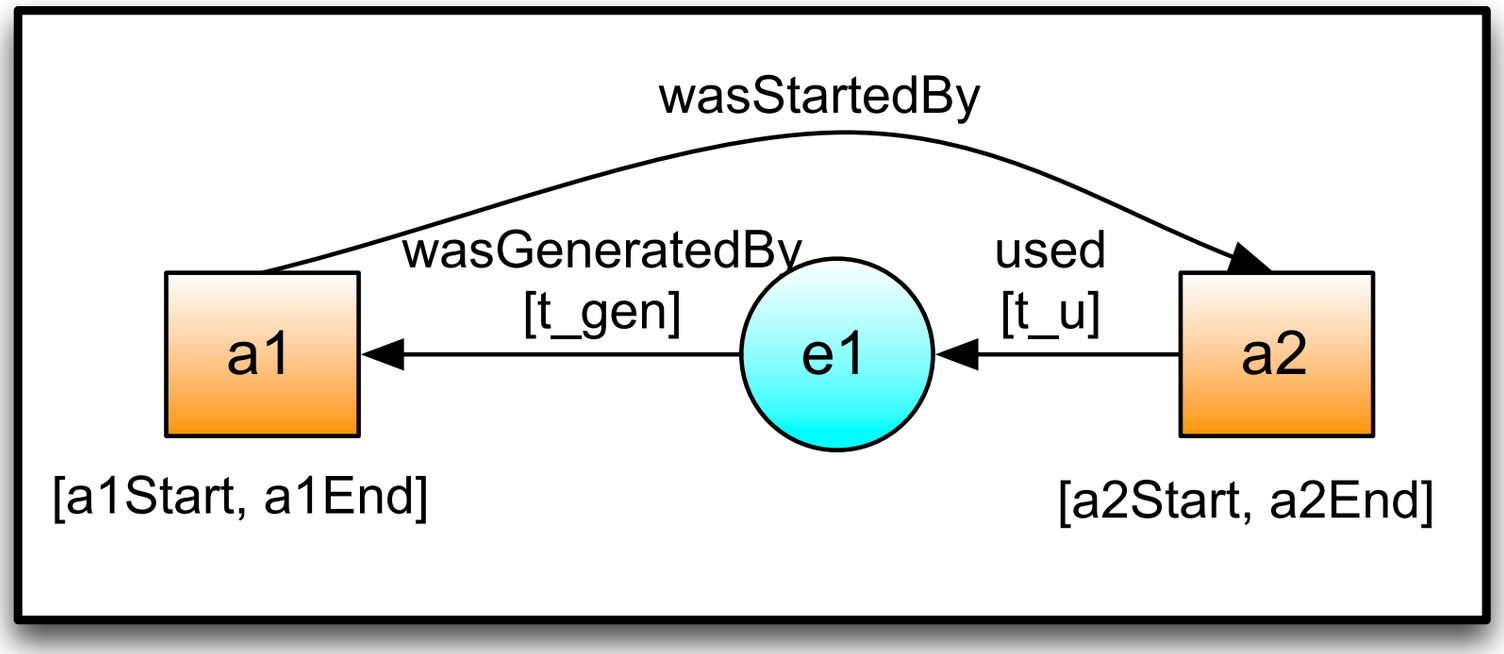
- to encode most PROV constraints as Datalog rules
  - (with some exceptions)
- Benefits:
  - A declarative specification with a deductive inference model
  - Therefore, a validator for PROV graphs
  - With a well-understood query model
  - Useful for rapid prototyping of graph traversal algorithms for provenance analysis

Note: the implementation is done using the DLV: http://www.dlvsystem.com

PROV constraints as Datalog rules

Goal of this work:

- to encode most PROV constraints as Datalog rules
  - (with some exceptions)
- Benefits:
  - A declarative specification with a deductive inference model
  - Therefore, a validator for PROV graphs
  - With a well-understood query model
  - Useful for rapid prototyping of graph traversal algorithms for provenance analysis



Note: the implementation is done using the DLV: http://www.dlvsystem.com

Example of deductive rules: traceability

Inference 9 (inference-trace)

Given two identifiers `e2` and `e1` for entities, the following statements hold:

1. IF `wasDerivedFrom(e2,e1,a,g2,u1)` holds, for some `a, g2, u1`, THEN `tracedTo(e2,e1)` also holds.
2. IF `wasDerivedFrom(e2,e1)` holds, THEN `tracedTo(e2,e1)` also holds.
3. IF `wasAttributedTo(e2,ag1,aAttr)` holds, THEN `tracedTo(e2,ag1)` also holds.
4. IF `wasAttributedTo(e2,ag2,aAttr), wasGeneratedBy(-;e2,a,-,gAttr)`, and `actedOnBehalfOf(ag`
5. IF `wasGeneratedBy(e2,a,gAttr)` and `wasStartedBy(a,e1,sAttr)` hold, for some `a, gAttr , sAttr` th
6. IF `tracedTo(e2,e)` and `tracedTo(e,e1)` hold for some `e`, THEN `tracedTo(e2,e1)` also holds.

```
[1,2] tracedTo(E2, E1):-  wasDerivedFrom(E2,E1,_,_).

[3]  tracedTo(E, Ag)  :-  wasAttributedTo(E,Ag,_,_).

[4] tracedTo(E2, Ag1) :-  wasGeneratedBy(E2,A,_,_),
                          wasAttributedTo(E2,Ag2,_,_),
                          actedOnBehalfOf(Ag2,Ag1,A,_).

[5] tracedTo(E2, E1)  :-  wasStartedBy(A,E1,_),
                          wasGeneratedBy(E2,A,_,_).

[6] tracedTo(E3, E1)  :-  tracedTo(E3, E2), tracedTo(E2,E1).
```

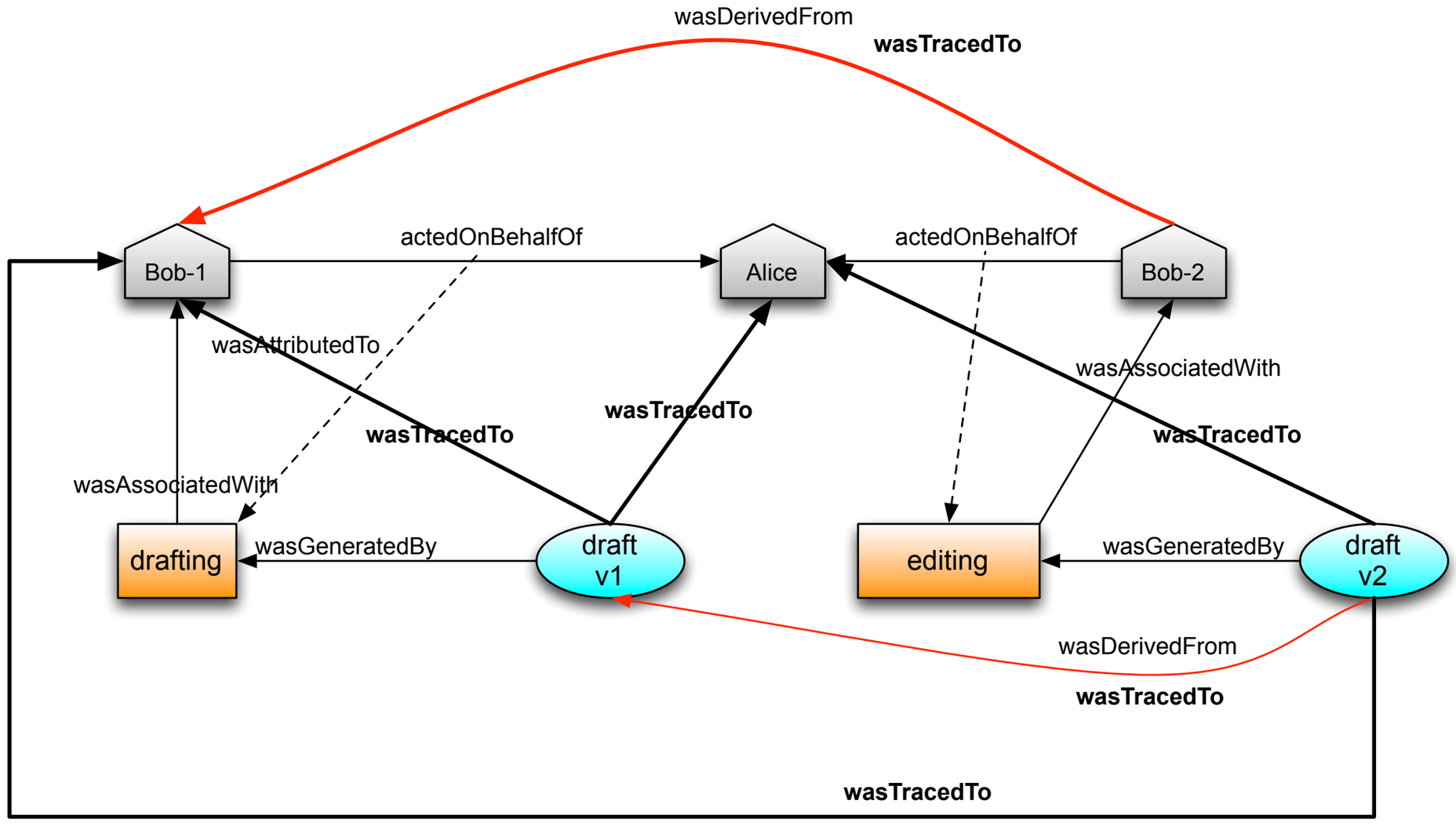
Computing the induced graph

query:

tracedTo(E, E1) ?

model:

tracedTo(draftV1, alice) [3] attribution, delegation
tracedTo(draftV1, bob_1) [2] (attribution)
tracedTo(draftV2, draftV1) [1] (derivation)
tracedTo(draftV2, alice) [5] transitivity
tracedTo(draftV2, bob_1) [5] transitivity
tracedTo(bob_2, bob_1) [1] (derivation)



Constraints: example

Inference 14 (specialization-antisymmetric)

For any entities `e1`, `e2`, it is not the case that `specializationOf(e1,e2)` and `specializationOf(e2,e1)`.

% anti-symmetry of specialization

false :- specializationOf(E3,E2), specializationOf(E2,E3), E2 != E3.

Interpretation:
a Datalog program that satisfies the body of the rule has no model

Constraints vs. inference

"IF `wasGeneratedBy(-;e, -, t1)` and `wasGeneratedBy(-;e, -, t2)` hold, THEN `t1=t2`."

⇓

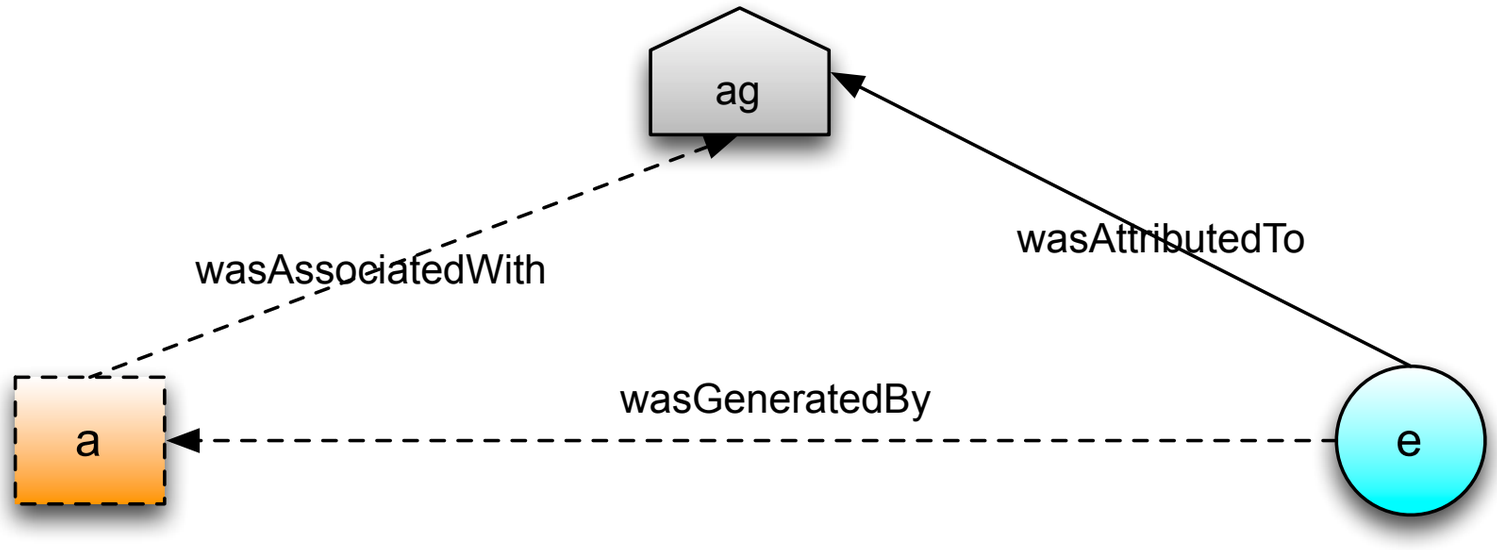
false :- wasGeneratedBy(E,_,T1,_), wasGeneratedBy(E,_,T2,_), T1 != T2.

Limitations of constraints mapping

Inference 8 (inference-attribution)

IF `wasAttributedTo(-;e,ag)` holds for some identifiers `e` and `ag`, THEN there exists an activity with some identifier `a` such that

```
activity(a, -, -)
wasGeneratedBy(-; e, a, -)
wasAssociatedWith(-; a, ag, -)
```



- The rule above generates a **set** of relations
- Existential quantification on **a**

- Also, attributes from relations in the body are not merged into new attributes for the head:

```
% derivation-use
used(A,E1, nil, T) :- wasDerivedFrom(E2, E1,_, Attrs1),
                      wasGeneratedBy( E2, A, Attrs2, T).
```

Ad hoc provenance analysis -- examples

Find all pairs of agents, along with the length of each of the paths amongst them

- an embrionic form of “distance” amongst agents to express how strongly they are related

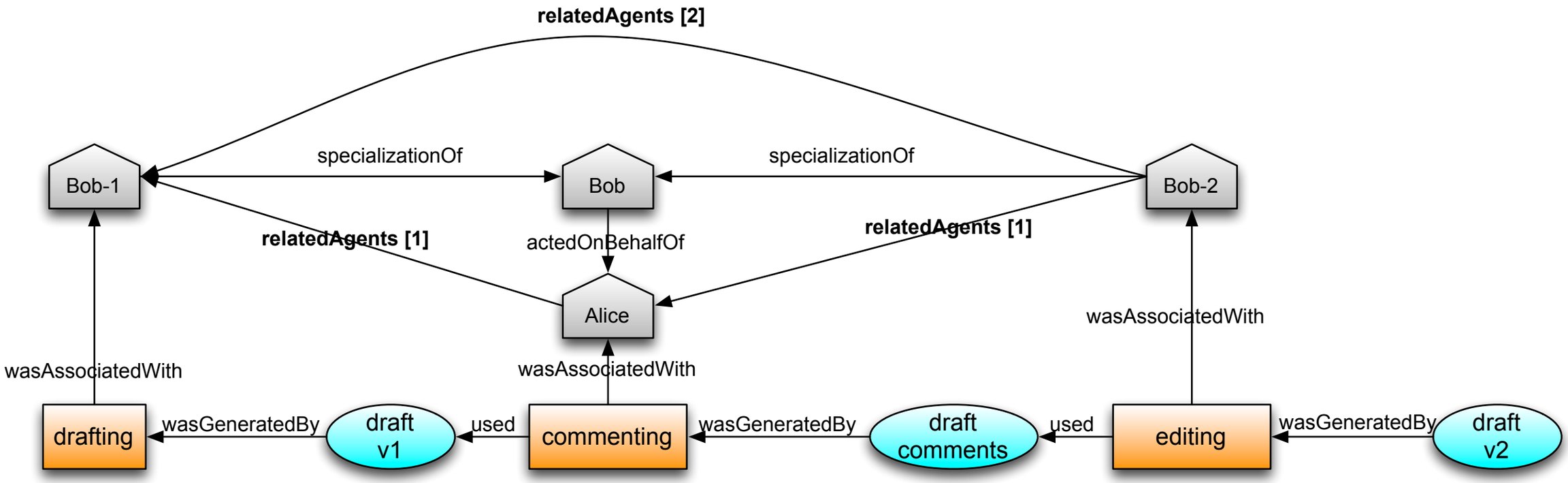
```
wasInformedBy(A2, A1,nil) :- wasGeneratedBy( E, A1, _, _),
                             used( A2, E, _, _).

relatedAgents0(Ag2, Ag1)    :- wasInformedBy(A2, A1,_),
                               wasAssociatedWith(A2,Ag2,_,_),
                               wasAssociatedWith(A1,Ag1,_,_).

relatedAgents(Ag2, Ag1, 1) :- relatedAgents0(Ag2, Ag1).

relatedAgents(Ag3, Ag1, N) :- relatedAgents0(Ag3, Ag2),
                              relatedAgents(Ag2, Ag1, M),
                              #succ(M,N).
```

- note: this simple version assumes no cycles

Related agents

```
relatedAgents(Ag2, Ag1, N) ?
```

```
alice, bob_1, 1
bob_2, alice, 1
bob_2, bob_1, 2
...
```

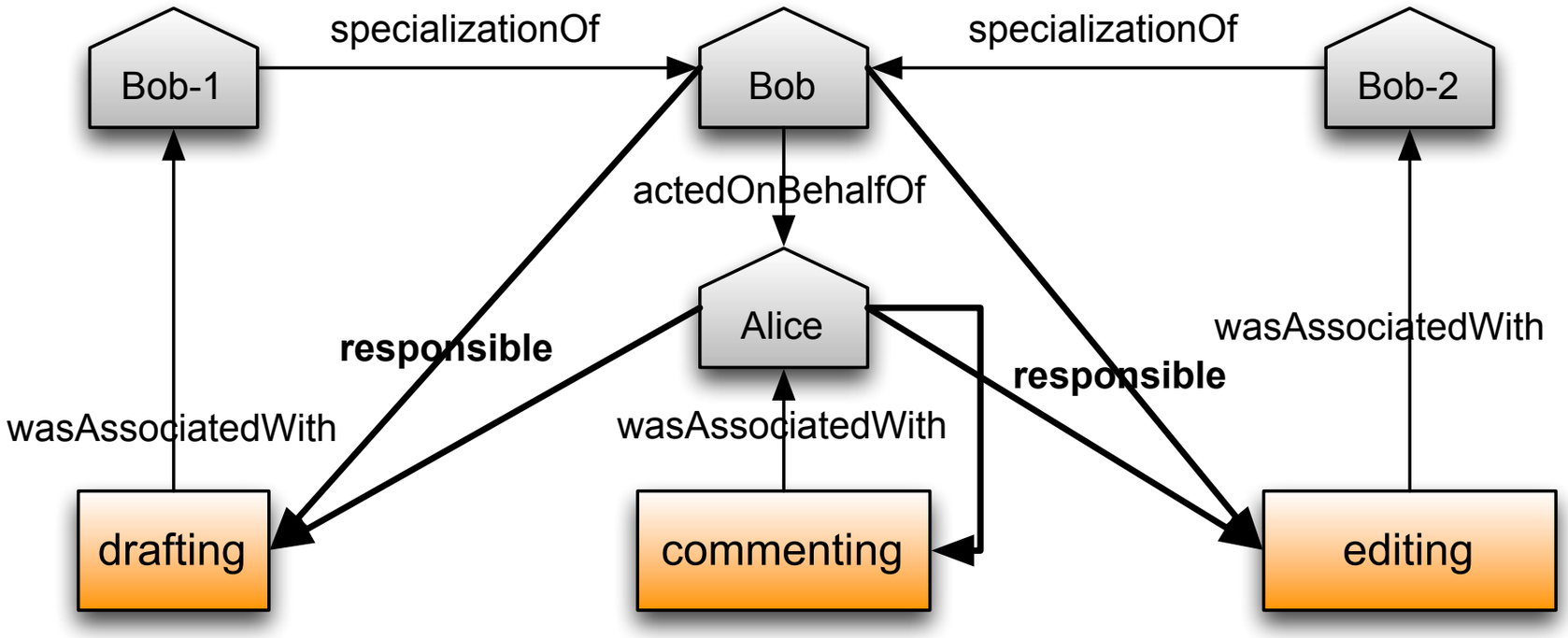


Chain of responsibility

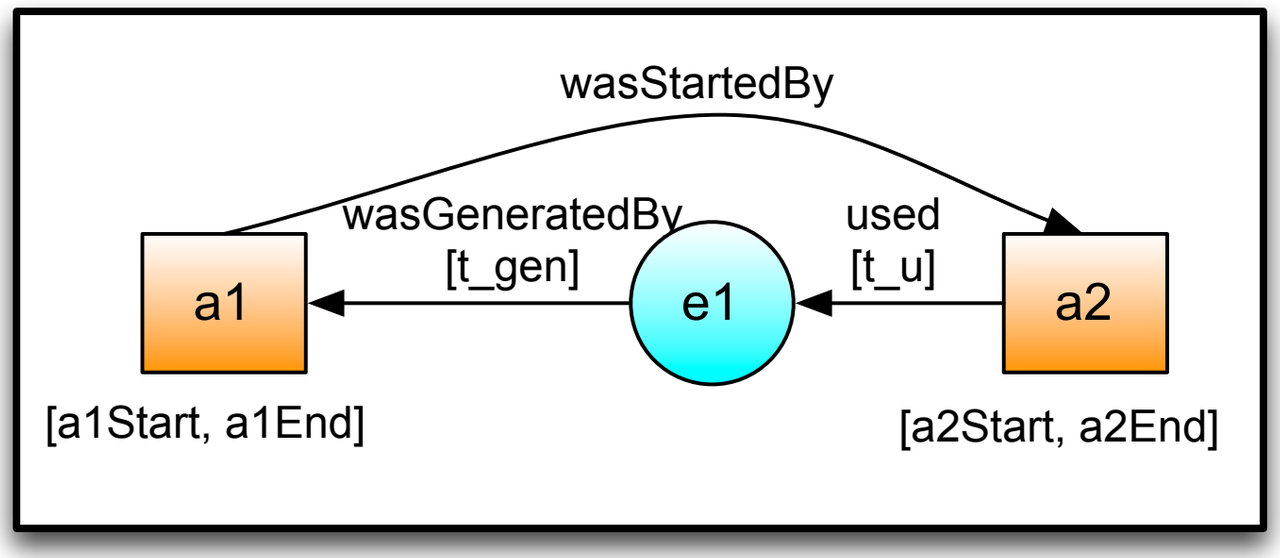
```
responsible(Ag, Act) :-  wasAssociatedWith(Act,Ag,_,_).

responsible(Ag1, Act) :- actedOnBehalfOf(Ag,Ag1,_,_),
                         responsible(Ag, Act).
```

responsible(Ag, Act)?

```
alice, drafting
alice, commenting
alice, editing
bob, drafting
bob, editing
bob_1, drafting
bob_2, editing
```



Encoding temporal constraints



Encoding temporal constraints

false :- **precedes**(T1,T2), **precedes**(T2,T1), T1 != T2. % anti-symmetry

precedes(T1,T3) :- **precedes**(T1,T2), **precedes**(T2,T3). % transitivity

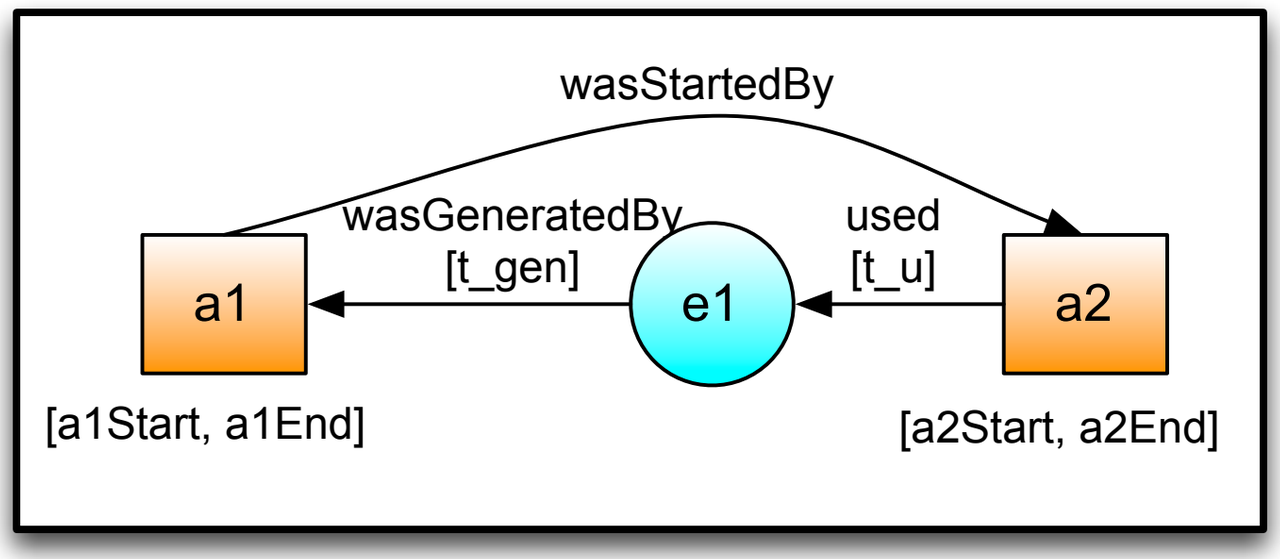
% Generation-precedes-usage

precedes(T2,T1) :- used(_, E, _,T1), wasGeneratedBy(E, _, _, T2), T1 != nil, T2 != nil.

precedes(T1, UT) :- activity(A, T1, _, _), used(A,_, _,UT), T1 != nil, UT != nil.

precedes(UT, T2) :- activity(A, _, T2, _), used(A,_, _,UT), T2 != nil, UT != nil.

precedes(ST1, ST2) :- wasStartedBy(A2,A1,_), activity(A1, ST1,_,_), activity(A2, ST2, _, _).



Encoding temporal constraints

false :- **precedes**(T1,T2), **precedes**(T2,T1), T1 != T2. % anti-symmetry

precedes(T1,T3) :- **precedes**(T1,T2), **precedes**(T2,T3). % transitivity

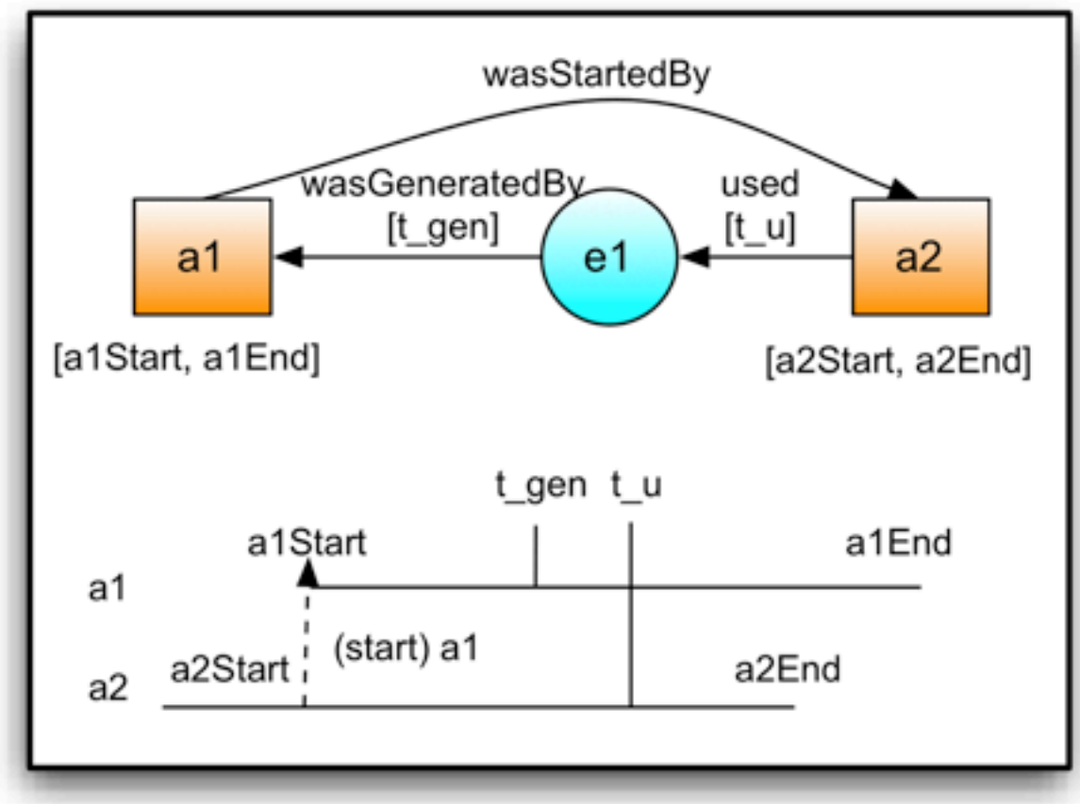
% Generation-precedes-usage

precedes(T2,T1) :- used(_, E, _,T1), wasGeneratedBy(E, _, _, T2), T1 != nil, T2 != nil.

precedes(T1, UT) :- activity(A, T1, _, _), used(A,_, _,UT), T1 != nil, UT != nil.

precedes(UT, T2) :- activity(A, _, T2, _), used(A,_, _,UT), T2 != nil, UT != nil.

precedes(ST1, ST2) :- wasStartedBy(A2,A1,_), activity(A1, ST1,_,_), activity(A2, ST2, _, _).



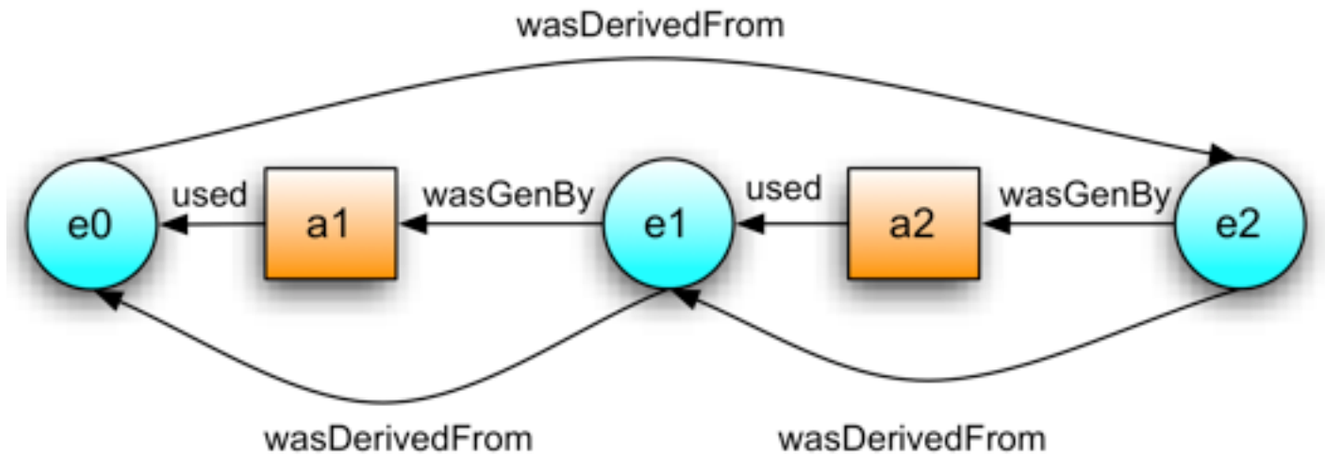
precedes(A,B) ?

a1Start <= a2End
a1Start <= t_u
a1Start <= t_gen
a2Start <= a1Start
a2Start <= a1End
a2Start <= t_u
a2Start <= t_gen
t_u <= a2End
t_gen <= a1End
t_gen <= a2End
t_gen <= t_u

Detecting illegal cycles

Derivation cycles are not allowed:

```
derivable(E2, E1) :-  wasDerivedFrom(E2, E1,_,_).
derivable(E2, E1) :-  derivable(E2, E0),  derivable(E0, E1).

% no-cycles constraint
false :- derivable(E2, E1), derivable(E1, E2), E1 != E2.
```



Query: `derivable(A,B) ?`
returns no model

Summary

- A Datalog encoding for PROVenance graphs
  - PROV-N mapped to a database of facts
  - PROV constraints mapped to Datalog rules
  - implemented using DLV, a former research system with a startup home

- Why this is appealing:
  - straightforward mapping from PROV-N
  - most constraints easily encoded
  - well-understood declarative style, well-established computational model
  - rapid prototyping of graph traversal rules and queries for provenance analysis

- Still only a proof of concept
  - constraints will evolve, W3C Note to be issued with a final encoding
  - small scale examples. Relies on DLV optimizations, which are untested

- Potential for a stronger implementation
  - DLV can be embedded into Java
  - comes with a variety of front-end reasoners, e.g. constraint solvers