Policy Based Access Control for RDF stores

Pavan Reddivari, Tim Finin and Anupam Joshi

UMBC
AN HONORS UNIVERSITY IN MARYLAND

ebiquity

http://ebiquity.umbc.edu/paper/html/id/334/

Motivation: RDF for Knowledge Sharing

- A 90s vision was to achieve information interoper-ability by turning sources into agents speaking the same language
  - KIF + shared ontologies for content
  - KQML for speech acts and protocols
- RDF is the new KIF and SPARQL the new KQML
  - But they must be enhanced to support updates, access control, etc.
- RAP explores some of these issues
  - Via an implemented prototype built on Jen

RAP in a Nutshell

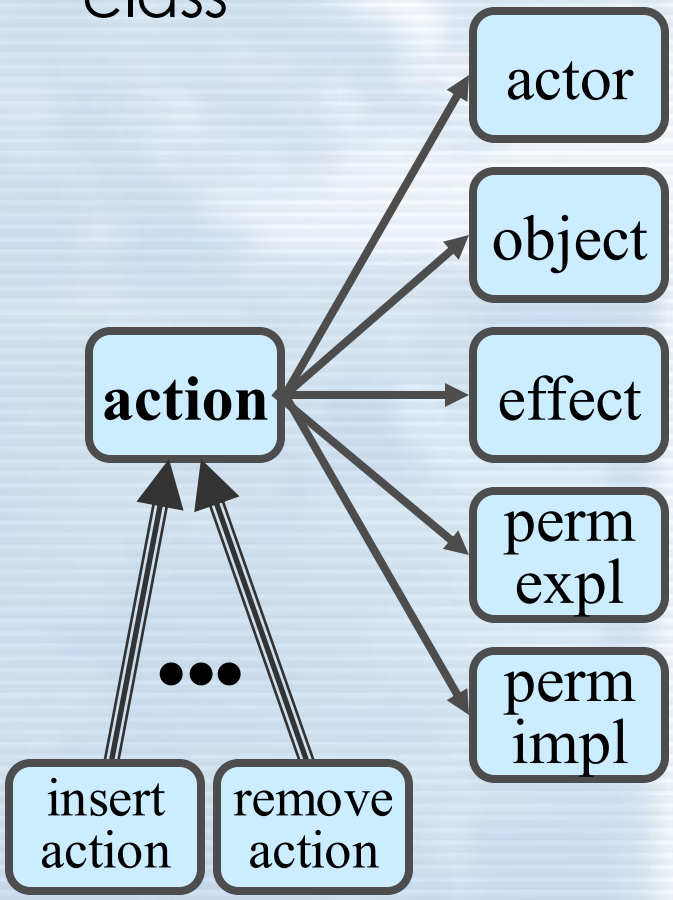
- RAP models the actions that an agent uses to modify or query an RDF triple store
- RAP supports policy rules that constrain the actions a given agent is permitted to do
- Policy rules can involve any information in the a triple store, including:
  - data
  - provenance
  - information

Example Policy Rules

- Only agents designated as editors can insert/delete triples
- An agent can only delete triples it previously inserted
- An agent can only add properties to classes it introduced
- No agent may see any values of the SSN property
- No agent may insert a triple that allows any agent to infer a patient's 'HIV status'
- An agent may modify any data about itself
- An agent may not add a foaf:Person instance without also providing a foaf:name property and either a

RAP Ontology: Actions

RAP has a simple RDFS ontology of which action is a key class



Nine action subtypes

- actor: The agent requesting the action
- object: The object that is the action's target
- effect: An implicit action caused by the action
- perm expl: The action's computed explicit permission
- perm impl: The action's computed implicit permission

Insertion Actions

- An agent can directly insert a triple or set of triples into the store's graph
  - **Insert(A,T):** A directly inserts triple T into the graph
  - **InsertSet(A, {Tc}):** A inserts the set {Tc} together
- An agent can also perform the implicit action of inserting a triple into the store's model
  - **InsertModel (A,T):** A InferInsert triple T If A Insert (A,T1) and triple T can be inferred after inserting T1

    insert(A,[usp:bush,foaf:mbox,bush@wh.gov]) has the effect
    insertModel(A,[usp:bush,rdf:type,foaf:Person])

Deletion Actions

- An agent can directly delete a triple or set of triples from the store's graph
  - **Remove(A,T):** A directly removes T from the graph
  - **RemoveSet(A, {Tc}):** A removes the set {Tc} together
- An agent can also perform the implicit action of removing a triple from the store's model
  - **RemoveModel (A,T):** A InferInsert triple T If A Insert (A,T1) and triple T can be inferred after inserting T1

    remove(A,[usp:bush,foaf:mbox,bush@wh.gov]) may have the effect removeModel(A,[usp:bush,rdf:type,foaf:Person])

Access Actions

- **See (A,T):** Agent A sees triple T if it returned in the response to one of A's queries.
- **Use (A,T):** Agent A uses triple T if it is used in answering one of A's queries.
- **Update(A,T1,T2):** Agent A directly replaces triple T1 with T2

Explicit policies

- Policy Representation:

  Modality(Action(A,T)) :- Condition

  Modality: Permitted or Prohibited

  A : Agent

  T: Triple

  Condition: Combination of simple constraints expressed as RDF triple

- Metadata Specific Conditions: Conditions in the policy can be based on metadata of the triples

  permit(insert(A,(?,rdfs:type,C))) :- createdNode(A,C)

Explicit policies

- Conditions in the policy can be based on kind of triple on which the action is being performed
  - *No agent can see any salaries*
  - prohibit(see(A,(?,emp:salary,?))
- Conditions in the policy can be based on Agent and its properties
  - *Supervisors can update the salaries of their supervisees*
  - permit(update(A,(P,emp:salary,?),(P,emp:salary,?)) :-

    existTriple(A,emp:Supervisor,P)
- Conditions in the policy can also be a combination of

Meta Policies

- Before performing an action, RAP tries to prove that it is permitted and that it prohibited
- RAP handles cases where neither or both proofs success with meta-policies for:
  - **default policy**: A default policy specifies the permission in cases where there is no explicit policy to prove a permission.
  - **modality preference**: A modality preference policy is used to the deal with the conflict situation.

Proven Prohibited \ Proven Permitted	No	Yes
No	?	Permitted
Yes	Prohibited	Conflict

RAP Ontology: metadata

RAP's ontology supports a many metadata properties useful in defining policies

- **isTripleOwner(A,T):** This predicate determines ownership of the triple. It returns true if agent A created the triple T.
- **isNodeOwner(A,N):** This predicate determines ownership of the node in the RDF graph. It returns true if agent A was first to create the node N in the RDF graph.
- **isSchemaPredicate(P):** This predicate would return true if P is a predicate used to define RDF schema level information (e.g., rdfs:subClass, rdfs:domain,etc).
- **isSubProperty(P1,P2):** true if P1 is a Sub-Property of P2

RAP rule compilation

The current implementation compiles RAP's policy into executable form as Jena forward chaining rules

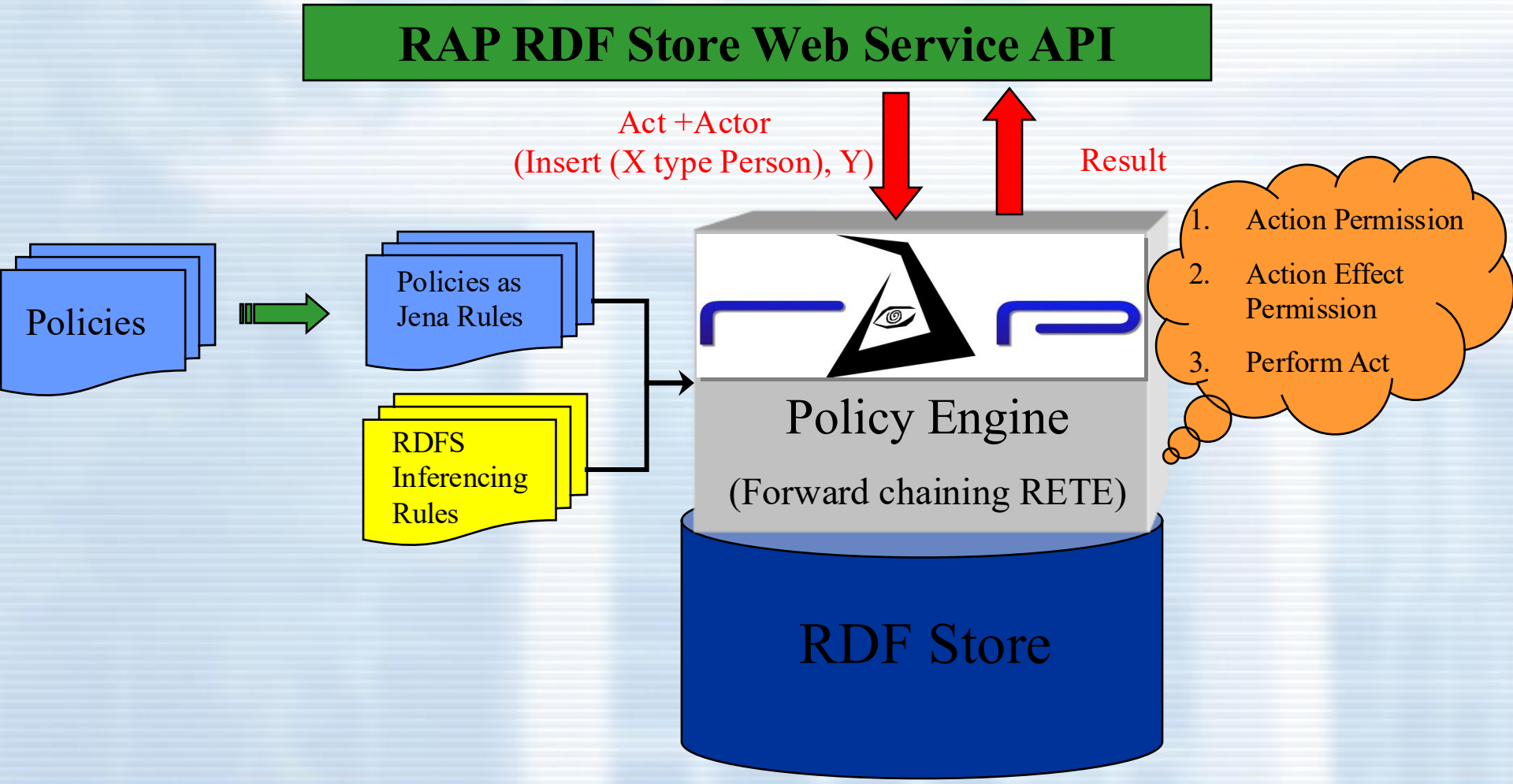
Gloss: "A supervisor of a Person can update that Person's salary"

RAP rule: permit(update(A,(P,emp:salary,?),(P,emp:salary,?)) :-
existTriple(A,emp:Supervisor,P)

Jena rule:

(?x rdf:type rap:Update_Action) (?x rap:Actor ?y
(?x ap:oldTriple_object ?z1) (?x newTriple_object ?z2)
(?z1 rdf:subject ?s1)(?z1 rdf:predicate emp: Salary)
(?z2 rdf:subject ?s1)(?z2 rdf:predicate emp:Salary)
(?y rap:Supervisor ?s1)
->
(?x rap:explicitPermission rap:Permitted)

Version 1.0


RAP RDF Store Web Service API
Act +Actor
(Insert (X type Person), Y)
Result
Policies
Policies as Jena Rules
RDFS Inferencing Rules
Policy Engine
(Forward chaining RETE)
RDF Store
1. Action Permission
2. Action Effect Permission
3. Perform Act

Version 1.0

- Current RAP prototype is being build using the JENA generic rule based Engine.
- The rule engine is used in the RETE forward chaining engine mode.
- The RAP Polices and the RDFS inferencing Polices are given as rules to the Engine
- This Enables RAP to check for modality of actions and their effects as in case of insertModel and removeModel actions.

RAP Query Approach

RAP 1.0 uses the following process to answer a query

- Run the RDQL query to get result set
- For each result, get all triples used to prove this result.
- Remove any of the triples in the result that the agent is not permitted to see.
- Using Jena's justification mechanism, remove any results that depend on triples that the agent is not permitted to use.
  - Only the derivation used to infer the triple is noted as there could more paths to inference the same triple

Contribution and Future Work

Contributions

- Prototype implementation demonstrating feasibility of a simple policy system for an RDF store.
- Policies can be defined over data, metadata, provenance an d usage history
- Increase usage of RDF stores in knowledge based application

Future work

- Moving RAP to a standard rule language such as SWRL or RuleML
- Integrating RAP ideas into SPARQL and into other RDF stores
- Expanding RAPs ontology to include additional actions (e.g., delegation) and predicates
- Extending RAP to include (some) OWL vocabulary

UMBC
AN HONORS UNIVERSITY IN MARYLAND

http://ebiquity.umbc.edu/

RAP Store

- Domain Knowledge and Policies are bound.