Rein : Where Policies Meet Rules in the Semantic Web

Lalana Kagal and Tim Berners-Lee

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
{lkagal,timbl}@csail.mit.edu

Abstract. Rein is a decentralized framework for representing and reasoning over distributed policies in the Semantic Web. Policies in Rein use information defined in and inferences made by other policies and web resources forming interconnected policy networks. Rein allows policies to be represented in different policy ontologies and requires the use of N3 rules, a semantic web rule language, for defining the connections in these networks. Reasoning over these networks to obtain policy decisions is done using cwm, an N3 reasoner. Rein consists of three main components - a high level ontology for describing policy networks, mechanisms in N3 for using information and inferences from both policies and web resources, and an engine for inferring policy decisions from policy networks. As part of our future work we would like to look into the use of SWRL and RuleML to develop similar functionality as the Rein framework.

1 INTRODUCTION

The Semantic Web provides mechanisms for searching through and making inferences over potentially millions of data with well defined semantics. In this open and dynamic environment, it is important to control how Web agents (including people, web services, and software agents) behave in terms of what resources (e.g. services, pictures, and webpages) they access (security), how their information is used by others (privacy), whether they are reliable (trust), how they establish and fulfill social and business obligations and contracts (obligation management), and how they make put information together to make inferences (data mining). Policies are norms of ideal behavior that influence how Web agents act. They are the ideal mechanism for behavior management on the Semantic Web because they provide the flexibility, openness, and automation required.

Our earlier work dealt with developing specific policy languages to describe policies in a rule language defined over OWL [1, 2]. However, our current work is aimed at providing a unifying framework that allows users to define policies in their own taxonomies and provides mechanisms in N3 rules that enable these policies to access and reason over each other and the web. Rein (**Rei** and **N**3) is a distributed framework for policy specification and reasoning, which exploits the inherently decentralized and open nature of the Semantic Web. Rein allows policies to be described using ontologies in RDF-S [3] and OWL [4], and serialized in RDF/XML [5] or N3 [6]. It also allows policies to use information and

inferences about entities and relationships defined in other policies and resources using mechanisms described in N3 rules. These policies and resources are implicitly connected through access links and form Rein policy networks. This leads to a modular architecture where a policy need not 'understand' the ontology or reasoning used by another policy or web resource in order to use it within its own rules. Policy networks allow the definition of policies in terms of other policies and information distributed on the Semantic Web.

Rein **does not propose a single policy ontology or language** for policy specification. It allows every policy to potentially have its own policy ontology and if required, a reasoner in N3 rules for interpreting the semantics of the policy ontology. Policies can range from simple to very complex. However, even the simplest policy carries its semantics by reference because it uses a namespace for the policy ontology, which can be looked up, and the rules (reasoner) associated with ontology can be used. Rein relies on N3 semantics and cwm functionality to integrate the policy networks. Rein supports N3 rules [6, 7] for representing interconnections between policies and resources. N3 rules allow policies to access the web and objectively check the contents and inferences of documents, without having to load them and believe everything they say. Rein uses the cwm reasoning engine [8] to provide distributed reasoning capability over policy networks. Though Rein consists of N3 and cwm, we believe that other rule languages such as RuleML [9, 10] and SWRL [11] and their reasoners could also provide similar functionality.

An important contribution of this work is that the creation of socially-aware systems usually involves a division of responsibilities between different parties with different roles and skills and Rein supports this. Designing ontologies, writing reasoners for policy ontologies, developing policies, and enforcing policies are all modular tasks. This allows policy developers to make frequent changes at their high level of understanding without requiring any other changes to the system.

Some of the main contributions of Rein include :

- It is a open and distributed approach to representing and reasoning over policies. Policies in Rein policy networks can be represented in any policy ontology.
- It allows flexibility in how sophisticated or expressive the policies can be. For example, a policy can use a simple ontology in RDF to list users and the resources they can/cannot access whereas another policy can be a set of N3 rules that defines access rights in terms of attributes of users, resources, and the environment and can use information and inferences from other policies and resources.
- It promotes extensibility and reusability as it allows every policy to have its own policy ontology and reasoner and still be used within policy networks by other policies.
- It provides a unified way for policies to access policy networks.
- It supports a compartmentalized approach to policy development.
- All objects in the system, including resources and policies, are self-describing.

The rest of the paper is as follows : Section 2 provides a high level description of N3 and cwm. Section 3 describes the Rein framework including the top level Rein ontology for describing policy networks, the mechanisms used by policies for accessing the web, and the process for reasoning over policy networks. Section 4 is an example of a Rein policy network that brings out some of the interesting capabilities of Rein. It also discusses how policy ontologies and their reasoners can be developed. Section 5 describes other policy efforts in the Semantic Web space and discusses their relationship to Rein. Section 6 summarizes the features of Rein and discusses some of our future research directions.

2 N3 AND CWM OVERVIEW

Notation3 or N3 provides a human-readable syntax for RDF and is a language that uses conventional unix-style punctuation, which is both more easily writable and readable than the RDF/XML syntax.

The N3 example below uses *prefix* for describing namespaces and defines *c1* an instance of the *Car* class as defined in *ex* namespace. Please refer to the primer [12] for more details about the N3 syntax.

```
ex:c1 rdf:type ex:Car;
      ex:licensedYear  2002, 2003, 2004;
      ex:color  "green".
```

N3 also provides functions for describing rules. The fact that the rule language and the data language are the same gives a certain simplicity (there is only one syntax) and completeness (rules can operate on themselves, anything written in N3 can be queried in N3). This would be broken if a special syntax were added for built-in functions and operators. Instead, these are simply represented as RDF properties. The N3 rule engine, cwm, when analyzing a rule prior to running it, treats specially those properties it knows as calculable functions which occur in the antecedent. Cwm is a forward chaining reasoner operating with such rules. Rules may have full N3, even with nested graphs, on both sides of the implication. This gives a form of completeness as rules can generate rules. When used as a rule language on RDF alone, N3 can of course be constrained so that there is no nesting of graphs.

Although initially developed as a serialization of RDF, full N3 extends RDF with variables and nested graphs. Some important features of N3 when combined with built-in functions of cwm include rules, accessing web resources, graph functions, inferencing within rules, and builtin functions such as math and string functions.

Quoting :

Various forms of literal value are allowed in RDF graphs, however the RDF standard does not provide for a RDF graph to be a data value. Remedying this allows one to express relationships between graphs, for example that a given graph is the RDF content of a particular document. The importance of agents on the semantic web being aware of where data has come from and where it is allowed to go to raises a need to be able to explicitly talk about graphs.

```
ex:Joe  ex:said   {  ex:Peter policy:permitted abc:acbService }.
```

Existential and universal variables :
In its blank nodes (items in the graph not directly identifies by a URI) an RDF graph has a form of existential variable. Extending the language to allow variables existentially or universally quantified over a graph allows N3 to be used for a form of logic. The reason for this initially was so that, given variables, a rule is just a relation between two graphs.

Variables are defined such that when substitution occurs in a graph, it also occurs in any nested graph.

Rules :
The *log:implies* property expresses a rule, its subject being the antecedent graph, and the object being the consequent graph. The shorthand $\Rightarrow$ may be used for log:implies. N3 allows blank nodes in the conclusion of a rule, hence allowing the creation of new objects.

The following example states that all students who have registered for some class are allowed to use the library.

```
@forAll :s.
{ @forSome :c.
  :s a school:GradStudent. :s school:registered :c. :c a school:GradCourse.
} => { :s policy:permitted school:UseLibrary }.
```

The use of N3 rules permits the description of policies that are not supported by either RDF or OWL such as policies that require chaining of variables (e.g. the uncle of relationship, same group as) and those that require calculations (e.g. the minutes must be available 3 days after the meeting, managers are permitted to sign on purchase orders $\leq$ 1000, for modeling safety conditions of a nuclear plant - pressure $\leq$ P, temperature $\leq$ T, radiation $\star$ flow $\leq$ RF).

Accessing web resources :
The built-in function *log:semantics* accesses a web resource, retrieves a representation of it, parses that, and returns the graph. Currently, cwm will retrieve and parse RDF/XML and N3. We have also developed a new builtin called *log:semanticsWithImportsClosure* that accesses a resource and all those resources that are part of its *owl:imports* closure.

Graph functions :
Another function, *log:includes*, checks whether one graph is a subset of the other. Together, *log:semantics* and *log:includes* allow rules to access the web, and to objectively check the contents of documents, without having to load them and believe everything they say.

Whereas some datasets (such as a list of members of a club) are definitively complete, others (such as a set of temperature measurements) are not. This aspect of the semantic web makes negation as failure meaningless unless it is

associated to a specific dataset. The effect of a default with an explicit domain is achieved with *log:notIncludes*, the negation of *log:includes*.

This rule states that if the specification for a car does not say what color it is then it is assumed to be black by default.

```
{ <carOrder.rdf> log:semantics :F.
  :F log:notIncludes { :car auto:color [] }
} => { :car auto:color auto:black }.
```

Inferencing within rules :

In order to do inference over a set of RDF statement within rules, the builtin function *log:conclusion* can be used.

Other builtin functions :

N3 also provides other built-in functions that aid in the development of expressive policies such as the *cryptographic* functions, *math* functions - cos, notGreaterThan, difference, and memberCount, *os* functions for retrieving environment information, *string* functions - concatenation, matches, and startsWith, and *time* functions.

We can define a rule that states that checks of more than $10,000 are only valid if signed by managers from the Finance department.

```
@forAll :check, :amt, :manager.
{  :check a biz:Check.
   :check biz:amount :amt.
   :amt math:greaterThan "10000".
   :check biz:signedBy :manager.
   :manager biz:department biz:FinanceDept.
} => { :check a biz:ValidCheck }
```

3 REIN FRAMEWORK

Rein is a distributed policy framework for resources (web resources, services, actions, etc. [1]) on the Semantic Web. It uses N3 rules to allow policies to access each other and other web resources and uses cwm for deducing policy decisions. Though, Rein uses the basic policy concepts defined by the Rei policy specification language [1, 2], it is a high level framework that allows policies to be represented in different ontologies and provides mechanisms for describing and reasoning over networks of policies and web resources. It consists of (i) a high level ontology for describing policy networks, (ii) mechanisms that allows policies to access information and inferences defined in other policies and web resources, and (iii) an engine that reasons over Rein policy networks to infer policy decisions. Though Rein is a general policy framework, it has been tested so far for access control policies only. Whenever a resource is requested, the resource or the entity managing the resource (such as a web server) creates a *rein:Request*

[1] Resources, actions and services, as things to which access may or may not be allowed, are treated identically by the system

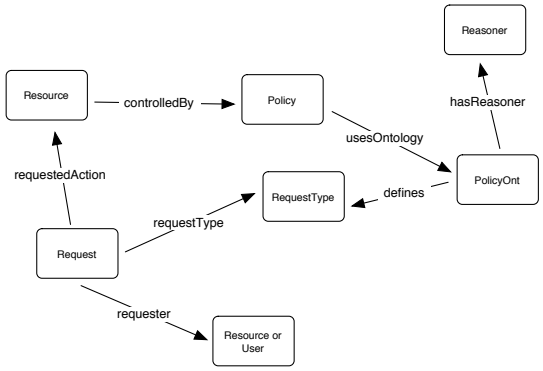

Fig. 1. Rein Policy Network Ontology

instance and uses the Rein reasoning engine to check whether the request is valid. If the request is valid, the request is allowed to go through.

We use the following prefixes in the rest of the examples :

```
@prefix log:  <http://www.w3.org/2000/10/swap/log#>.
@prefix string:  <http://www.w3.org/2000/10/swap/string#>.
@prefix rein: <rein.n3#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix book: <book.n3#>.
@prefix pol: <policy-book.n3#>.
@prefix ont: <pol-ont-book.n3#>.
@prefix mit-ont: <ont-mit.n3#>.
@prefix fil: <filter.n3#>.
@prefix policyontb: <policy-ontb.n3#>
```

3.1 Rein Policy Network Ontology

Though policies can be described in their own policy ontologies, the Rein framework requires the use of its policy network ontology for defining the connections in the policy network. The policy network ontology as illustrated in Figure 1 includes the *rein:Request* class and defines various properties that resources and policies trying to build networks should use.

The *usesOntology* property is used by policies to let cwm know the policy ontology they use. *hasReasoner* is a property used by a policy ontology to describe an external policy reasoner. A policy reasoner is a set of N3 rules that describes the semantics of the policy ontology. Not all policy ontologies will have reason-

ers, but if they do this property is set to inform cwm that an external reasoner must be used to make inferences over policies that use this policy ontology.

```
@forAll :policy, :ontology, :reasoner, :F.
{ :policy.log:semantics log:includes { :policy rein:usesOntology :ontology }.
  :ontology.log:semantics log:includes {:ontology rein:hasReasoner :reasoner }.
} => {  :policy rein:requiresReasoner :reasoner }
```

The *rein:Request* class is used by the Rein reasoning engine to query policy networks. We define three properties - *requester* defines the entity making the request, *requestedAction* is the resource, service, or action being requested, and the *requestType* is a property defined in the domain specific policy ontology for access control. The Rein reasoning engine can be be easily modified to handle additional properties for a request including time, location, and attributes of the requester and resource instead of just their identities.

The *controlledBy* property is used by a resource to inform Rein that a policy (or policies) defines rules about its access.

An example of a resource, policy, and request is described below. This example is a collection of N3 statements from several documents.

```
# from book.n3
:BookService a ws:WebService;
    rein:controlledBy <policy-book.n3>;
    ws:bookTitle :title;
    ws:price :price.

# from policy-book.n3
<> rein:usesOntology <ont-book.n3>.

# from ont-book.n3
<> rein:hasReasoner <reasoner-book.n3>.

# from request.n3
:request1 a rein:Request;
    rein:requester <dig-list.n3#lalana>;
    rein:requestedAction <book:BookService>;
    rein:requestType <ont:permitted>.

:request2 a rein:Request;
    rein:requester <w3-list.n3#tim>;
    rein:requestedAction <book:BookService>;
    rein:requestType <ont:prohibited>.
```

3.2 Mechanisms for Accessing Web Resources and Policies

Rein proposes mechanisms for accessing information defined in and inferences made by Rein policies and web resources. These mechanisms are derived from the functionality of N3 and cwm. Policies can check for the presence or absence of information in or inferences made by web resources and other policies.

Presence of information in a web resource :

In order to check for the presence of certain information in a web resource, Rein suggests the use of *log:semantics* and *log:includes* builtins provided by N3.

In the following example, the file describing book:BookService is checked to see who it says its deployer is. The deployer information is then read in and all people that the deployer knows are permitted to use the book:BookService. At no time is either file trusted for any other information.

```
@forAll :n, :d, :F, :G.
{  book:BookService log:semantics :F.
   :F log:includes { book:BookService abc:deployed :d }.
   :d  log:semantics :G.
   :G log:includes  { [] a foaf:Person; foaf:knows
                [ a foaf:Person; foaf:homepage :n] }.
} => { :n ont:permitted book:BookService }.
```

Sometimes along with accessing a file, the closure of all files imported by the file also need to be accessed and checked for the presence of certain information. This is possible using the *log:semanticsWithImportsClosure* builtin.

Absence of information from a web resource :

It is not possible to enumerate all instances of a certain dataset so we need to model defaults. This is done by using the negation of *log:includes*, which is *log:notIncludes*.

In the example below, if the requesting entity does not belong to the DIG list i.e. if the entity is not defined in dig-list.n3, then the entity is prohibited from using the book:BookService.

```
@forAll :request, :actor, :F.
{  :request a rein:Request.
   :request rein:requester :actor.
   <dig-list.n3> log:semantics :F.
   :F log:notIncludes { [] a foaf:Person; foaf:homepage :actor }.
} => { :actor ont:prohibited book:BookService }.
```

It is also possible to use *log:notIncludes* with the imports closure provided by *log:semanticsWithImportsClosure*.

Presence of inference from a web resource :

Instead of only checking the presence or absence of certain data, it is also possible to check inferences made by web resources including policies. For example, an RDF file may specify that Mark is a GradStudent, where GradStudent is a subclass of Student and Student is a subclass of MemberOfMIT. In this case, if *log:semantics* and *log:includes* are used, Mark is only a GradStudent and we cannot infer that Mark is also a Student and a MemberOfMIT. In order to draw all conclusions, Rein proposes the use of *log:conclusion* in addition to *log:semantics* and *log:includes*.

Absence of inference from a web resource :
Similar to checking the absence of a certain information or graph from a resource, in addition to *log:conclusion*, *log:notIncludes* is used to check the absence of an inference.

Presence of inference from a policy :
Though, the presence of information from a web resource can be easily checked using *log:semantics*, *log:conclusion*, followed by *log:includes*, the presence of inference from a policy is different. This is because in order to access inferences of another policy, a policy must query the Rein policy network and Rein provides a unified interface to policy networks. To query the network, a *rein:Request* instance is created and queried against the Rein engine (engine.n3). This is done using *log:conjunction* to add *rein:Request* to engine.n3, followed by *log:conclusion* for drawing inferences and *log:includes* to check whether the required inference was made. This implies that the access rights of entities on resources can be checked using the policy network without knowing which policies are acting on the resources because resources themselves identify the policies that govern them.

In this example, if a grad student of MIT is permitted to access book:BookService, then he/she is also permitted to perform mit-ont:abcAction.

```
@forAll :x, :F, :G.
{  :x a ont:MemberOfMIT.
   ({[] a rein:Request; rein:requester :x;
     rein:requestedAction book:BookService; rein:requestType ont:permitted}
    <engine.n3>.log:semantics) log:conjunction :F.
   :F log:conclusion :G.
   :G log:includes {:x ont:permitted book:BookService}
} => {:x mit-ont:ispermitted mit-ont:abcAction}.
```

Absence of inference from a policy :
Checking the absence of inference from a policy can be done in two ways - (i) by the reasoner and is especially useful for defining policy defaults, for example prohibited if not explicitly permitted and permitted if not explicitly prohibited, and (ii) by another policy.

If used in a reasoner, then the mechanism is similar to checking the absence of an inference from a resource. The following rule is part of a reasoner for a policy ontology. If book:BookService is not permitted then it is prohibited.

```
@forAll :request, :action, :actor, :F, :G.
{   :request a rein:Request.
    :request rein:requestedAction :action.
    :request rein:requester :actor.
    :F :gives :G.
    :G log:notIncludes {:actor ont:prohibited :action}
} => {:actor ont:permitted :action}.
```

However, if the checking of the absence of an inference is by another policy, then the mechanism is similar to checking the presence of an inference from a policy (i.e. using *rein:Request* instance and engine.n3) except it uses *log:notIncludes*.

For example,if a grad student of MIT is not permitted to access book:BookService, then he/she is prohibited from performing mit-ont:xyzAction.

```
@forAll :x, :F, :G.
{  :x a ont:MemberOfMIT.
   ({[] a rein:Request; rein:requester :x;
     rein:requestedAction book:BookService; rein:requestType ont:permitted}
    <engine.n3>.log:semantics) log:conjunction :F.
   :F log:conclusion :G.
   :G log:notIncludes {:x ont:permitted book:BookService}
} => {:x mit-ont:isprohibited mit-ont:xyzAction}.
```

Use of other builtin functions on a web resource :
A Rein policy can use other builtin functions of N3 for greater expressivity. For example, it can use *math* functions to define safety conditions of a nuclear plant, or the stopping distance of a car, *date* functions to describe policies such as the minutes of a meeting must be available 3 days after the meeting and only upto one month after the meeting, and *string* and *list* functions for policies about filtering data such as if a file contains one of the words in the list, it is unsuitable for children under thirteen.

```
@forAll :doc, :word, :F.
{  :doc log:includes :F.
   :listOfWords list:in :word.
   :F string:containsIgnoringCase :word.
} => { :doc fil:prohibitedFor fil:under13 }.
```

Include entire web resource :
A Rein policy can also include the entire contents of other policies and web resources by using *owl:imports*. cwm can be run with the *–closure=i* option which causes it to retrieve the closure of all the files being imported and treat them as a single RDF graph. *log:semanticsWithImportsClosure* can also be used to access all the files imported by a resource or policy.

3.3 Reasoning Engine

The Rein reasoning engine is an N3 rules file, engine.n3 It accepts as input a file containing a list of *rein:Request* instances. It processes every *rein:Request* by finding the policy associated with the resource and reasoning over its network. The engine finds the policy that controls the requested resource by looking for its *rein:controlledBy* property. It then reads in the policy using *log:semantics* and gets it *rein:usesOntology* property. The engine then checks if the ontology has a reasoner. If it does not, then the engine accesses the policy file and the resource file. If the policy ontology does have a reasoner, it reads in the reasoner file and the *rein:Request* class. The Rein engine then infers over the files accessed

using *log:conclusion*. It then tests for the presence of a triple formed by [*requester requestType requestedAction*] using *log:includes*. If the triple exists, the *requester* has the *requestType* of deontic on the *requestedAction* and the *rein:Request* is said to be true.

In order for the Rein reasoning engine to be executed, the file containing the *rein:Requests* must be added at the commandline as input to cwm. The correct cwm usage for querying is

```
cwm --n3 myRequest.n3 engine.n3 --think --filter="engine.n3"
```

It is also possible to query Rein policy networks from within other resources using N3 rules as described in Section 3.2. The policy or reasoner cannot be arguments to engine.n3 implying that a request cannot be executed against wrong policies or reasoners. All the information required to infer the validity of requests is defined within the resource being asked for access, the policy that controls it, and the ontology used for describing the policy making our approach completely self-describing.

4 EXAMPLE

In order to verify the utility of the Rein framework, we developed three policy ontologies - two of which require reasoners and one that is simple enough to be described without a reasoner. We used these policy ontologies to develop six policies. We also defined several resources that were controlled by these six policies. We used the Rein network ontology to define a policy network. Figure 2 is a pictorial representation of part of this Rein policy network. The Rein network ontology, engine, and examples are available at http://dig.csail.mit.edu/Rein

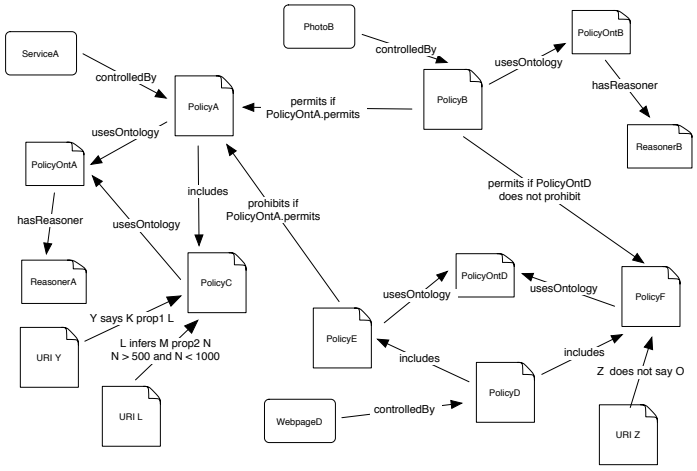

Fig. 2. Rein Policy Network Example

There are three resources being protected by the six policies - ServiceA, PhotoB, and WebpageD. ServiceA is controlled by PolicyA. PolicyA is described in the PolicyOntA ontology, which has a reasoner, ReasonerA. PolicyA also includes PolicyC, which is described in PolicyOntA. This implies that both PolicyA and PolicyC are used in conjunction to control access to ServiceA. PolicyC uses information from URI Y and an inference from URI L to make decisions about access to ServiceA. PolicyB is the policy that controls PhotoB. It uses the PolicyOntB ontology, which has a reasoner, ReasonerB. It also uses inferences about access to ServiceA and WebpageD. For example, if X is permitted (described by the PolicyOntA) to access ServiceA, PolicyB allows John to access PhotoB. WebpageD is controlled by PolicyD, which includes PolicyE and PolicyF. Both PolicyE and PolicyF use the same ontology, PolictOntE. PolicyF uses information from URI Z within its rules to decide access rights to WebpageD.

4.1 Simple Policy Language in N3

In order to show how policy ontologies and reasoners can be developed, we describe one we developed, PolicyOntB.

PolicyOntB consists of four kinds of deontic concepts - pospermitted, posprohibited, permitted, and prohibited. permitted is a subclass of pospermitted, and prohibited is a subclass of posprohibited. A policy uses pospermitted and posprohibited to define rules for possible access. The reasoner uses these rules and the meta policies to infer actual permissions and prohibitions. The reasoner accesses the policy that uses PolicyOntB, whose semantics it interprets, using the following rule. Rein suggests that all reasoners use the same mechanism.

```
@forAll :request, :action, :policy, :ontology, :F, :G.
{   :request a rein:Request.
    :request rein:requestedAction :action.
    :action.log:semantics log:includes { :action rein:controlledBy :policy }.
    :policy.log:semantics log:includes { :policy rein:usesOntology :ontology }.
    :ontology.log:semantics log:includes {:ontology rein:hasReasoner <> }.
    ( :policy.log:semantics
      :ontology.log:semantics
      :action.log:semantics) log:conjunction :F.
    :F log:conclusion :G.
} => { :F :policyInfo :G }.
```

The meta policies include defaults for both actions and policies - permittedByDef (if there is there is no prohibition, it is permitted) or prohibitedByDef (if there is no permission, then it is prohibited) [2]. It is possible to define defaults for an entire policy as a policy can be used by several actions. For conflict resolution, the meta policy ontology includes modality setting [2]. This allows the reasoner to choose either a permission (positive modality) or a prohibition (negative modality) if both exist.

A policy defined in PolicyOntB consists of policyontb:pospermitted and policyontb:posprohibited rules for access and meta policies for defaults and modality setting. All requests for a resource controlled by policies described in PolicyOntB

ontology will include policyontb:permitted and policyontb:prohibited as requestTypes. The reasoner for PolicyOntB starts out by figuring out the default and modality preference for every request. If there is a default associated with the action, it is used, otherwise the default defined by the policy is used. The following cases apply - (i) if permittedByDef and no prohibition rule, then the action is permitted, else it is prohibited (ii) if prohibitedByDef and no permission rule, then the action is prohibited, else it is permitted (iii) if there is no default and there is either a permission or a prohibition, the action is permitted or prohibited, and (iv) if there is no default and there is both a prohibition and permission rule, there is a conflict and the modality preference is used. The reasoner uses the following rules for conflict resolution - (i) if there is both a prohibition and permission rule and modality is positive, then the action is permitted, (ii) if there is both a prohibition and permission rule and modality is negative, then the action is prohibited, and (iii) if there is both a prohibition and permission rule and modality is not specified, then there is a conflict and the request is invalid.

The reasoner uses these rules when the default is permittedByDef.

```
@forAll :request, :actor, :action, :F, G.
{  :F :policyInfo :G.
   :request a rein:Request.
   :request rein:requester :actor.
   :request rein:requestedAction :action.
   :default :is policyontb:permittedByDef.
   :G log:notIncludes { :actor policyontb:posprohibited :action }.
} => { :actor policyontb:permitted :action }.

{  :F :policyInfo :G.
   :request a rein:Request.
   :request rein:requester :actor.
   :request rein:requestedAction :action.
   :default :is policyontb:permittedByDef.
   :G log:includes { :actor policyontb:posprohibited :action }.
} => { :actor policyontb:prohibited :action }.
```

5 EXISTING POLICY EFFORTS

Extensible Access Control Markup Language (XACML) [13] is a policy language in XML for expressing policies. It can be used to describe policies and rules in terms of boolean combinations of attribute-value pairs based on the subject, resource, and environment. It also provides conflict resolution through its combining algorithms. The Platform for Privacy Preferences (P3P) is a standard developed by the World Wide Web Consortium (W3C) that enables websites to describe their privacy policies and allows browsers to reason over these policies to decide whether they match the user's preferences [14]. KAoS is a policy language developed in OWL [15]. It is similar to Rei [2] and can be used to develop positive and negative authorization and obligation policies over actions.

KAoS policies are OWL descriptions of actions that are permitted (or not) or obligated (or not). This limits the expressive power, so that there are policies that can described in N3 rules that KAoS cannot. Using OWL, however, allows the classification of policy statements, enabling conflicts to be discovered from the rules themselves.

We consider XACML, P3P, and KAoS to be specific policy ontologies (such as PolicyOntA) that can be used within Rein policy networks to describe policies. If their semantics can be represented in N3 rules, it will be possible to integrate them seamlessly into the Rein framework.

RuleML [9, 16] is a proposed standard for a rule language, based on declarative logic programs. SWRL is a closely related rule language standard for describing Horn like rules in first order logic, intended to be combined with OWL-DL [11]. SWRL's syntax for Horn rules is essentially a subset of RuleML's. Unlike SWRL which is based on classical logic, full RuleML can represent nonmonotonicity and conflict handling because it is based on logic programs that have negation as failure and, in the courteous extension, priorities. As part of our future work, we will look into representing policies in RuleML and SWRL and using these rule languages to provide functionality similar to Rein.

6 CONCLUSION AND FUTURE WORK

Rein is an extensible and distributed framework for describing policies that govern the behavior of resources and users on the Semantic Web. Policies use information defined in and inferences made by other policies and web resources forming interconnected policy networks. Rein allows policies to be represented in different policy ontologies and requires the use of N3 rules, a semantic web rule language, for defining these policy networks. Reasoning over these networks to obtain policy decisions is done using cwm. Rein includes a high level ontology for describing policy networks, mechanisms for using information and inferences from both policies and web resources, and an engine that reasons over Rein policy networks to infer policy decisions. To show the usefulness of Rein, we have developed a Rein policy network of several resources, three policy ontologies (two of which require policy reasoners), and six policies that access each other and other resources in the network.

We have not taken into consideration the complexity caused by long chains of access in policy networks. For example, if PolicyA accesses ResourceB which accesses ResourceC which accesses ResourceD etc. Some possible solutions include either preventing more files from being accessed after n have been loaded into cwm or by allowing upto n links in an access chain. There is no problem if there are loops in access chains (PolicyA accesses PolicyB which accesses PolicyA) as we find the closure of the files.

Another problem is that Rein does not take possible attacks such as denial of service into consideration and imports all required rules. Cwm also does not do any sandboxing. We believe some kind of trust mechanism will be required, so that only policies and reasoners that can be trusted will be loaded into cwm.

The current querying mechanism using *rein:Requests* is very simple and only checks whether a requester has a certain access type on a resource. We would like to extend that to allow different kinds of queries such as who is permitted to perform a printing kind of service, what kind of resources can John access etc.

Though Rein is a general framework for policy specification and reasoning, it has been used so far only for access control. We would like to look into other behavior such as privacy, accountability, and collaboration.

As part of our future work, we would also like to look into using RuleML and SWRL to replicate the functionality of the Rein framework.

REFERENCES

1. Kagal, L., Finin, T., Joshi, A.: A Policy Based Approach to Security for the Semantic Web. In: Second Int. Semantic Web Conference (ISWC2003), Sanibel Island FL, October 2003. (2003)
2. Kagal, L.: A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments. Dissertation (2004)
3. W3C: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, http://www.w3.org/TR/rdf-schema/ (2002)
4. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference, W3C Recommendation 10 February 2004. http://www.w3.org/TR/owl-ref/ (2004)
5. W3C: RDF/XML Syntax Specification (Revised). http://www.w3.org/TR/rdf-syntax-grammar/ (2004)
6. Berners-Lee, T.: Notation 3. http://www.w3.org/DesignIssues/Notation3.html (1998)
7. Berners-Lee, T., Connolly, D., Prud'homeaux, E., Scharf, Y.: Experience with N3 rules. In: W3C Rules language Workshop. (2005)
8. Berners-Lee, T.: Cwm (2000)
9. : The Rule Markup Initiative. (http://www.ruleml.org/)
10. Boley, H., Grosof, B., Kifer, M., Sintek, M., Tabet, S., Wagner, G.: Object-Oriented RuleML. http://www.ruleml.org/indoo/indoo.html (2004)
11. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: Semantic Web Rule Language Combining OWL and RuleML. Version 0.6 of 30 April 2004. http://www.daml.org/rules/proposal/ (2004)
12. Berners-Lee, T.: Primer: Getting into RDF and Semantic Web using N3. http://www.w3.org/2000/10/swap/Primer (2005)
13. Godik, S., Moses, T.: OASIS eXtensible Access Control Markup Language (XACML). (OASIS Committee Secification cs-xacml-specification-1.0, November 2002)
14. Cranor, L., Langheinrich, M., Marchiori, M., Presler-Marshall, M., Reagle, J.: Platform for Privacy Preferences (P3P). http://www.w3.org/P3P (2002)
15. Uszok, A., Bradshaw, J.M., Jeffers, R., Johnson, M., Tate, A., Dalton, J., Aitken, S.: Policy and Contract Management for Semantic Web Services. In: AAAI Spring Symposium, First International Semantic Web Services Symposium. (2004)
16. Grosof, B.: Representing E-Commerce Rules Via Situated Courteous Logic Programs in RuleML. Electronic Commerce Research and Applications (ECRA) (2003)