Policy Assurance for PIR Queries

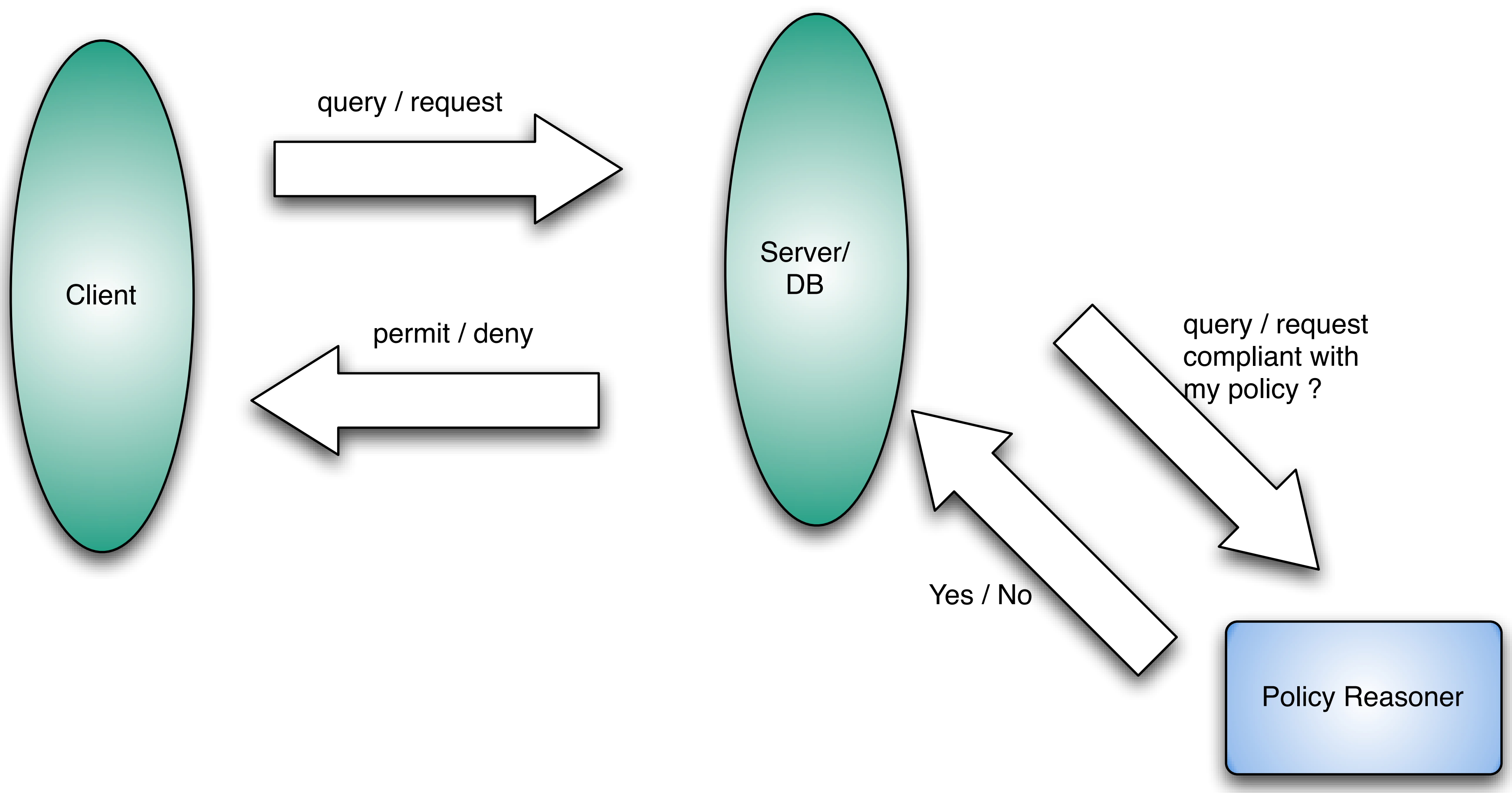
Lalana Kagal
MIT CSAIL
Decentralized Information Group

Overview

- Introduction
- Motivation and Problem Statement
- Challenges
- Technical Approach
- Next steps
- Summary

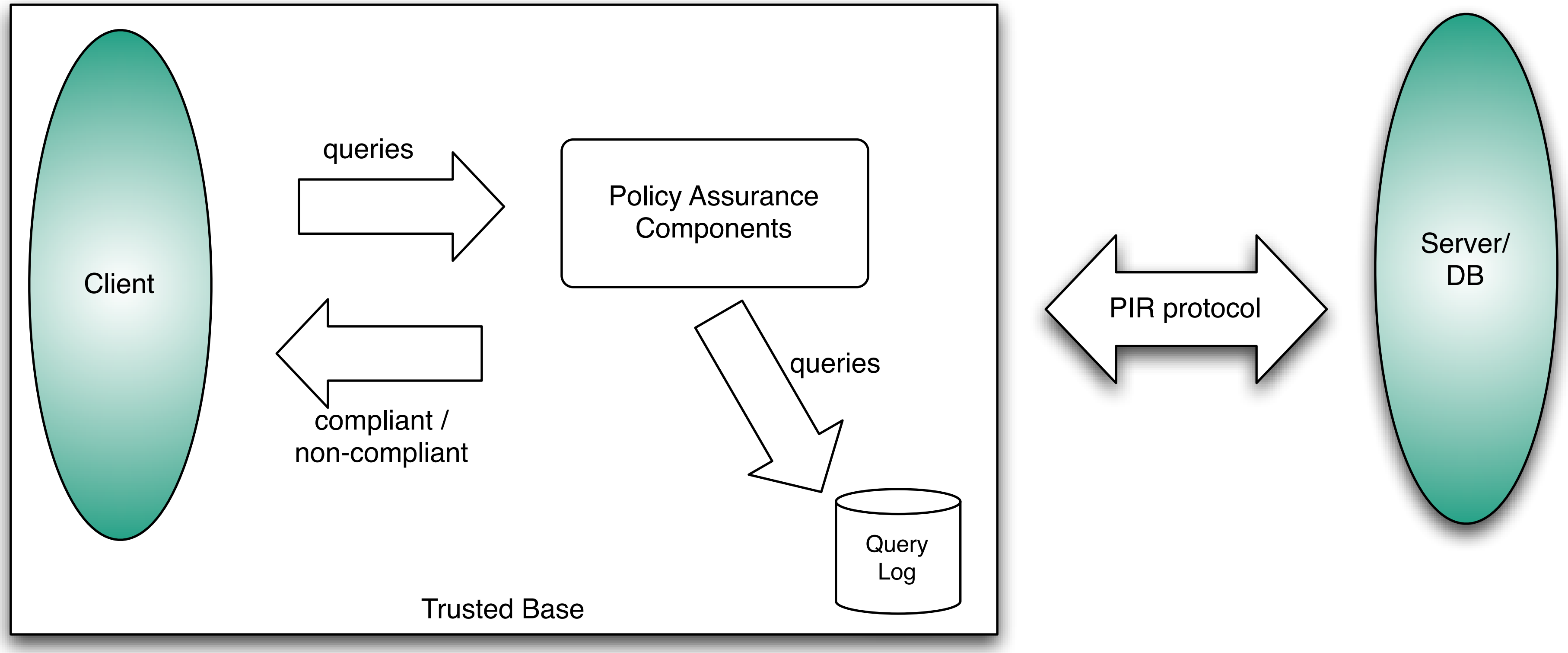
Introduction

- What does **policy compliance** mean ?
  - Proving that **requests** made by the client **conform to policies**
  - Usually for upfront authorization



Introduction

- What does policy assurance for PIR queries mean ?
  - Assuring that the client's queries are compliant with previously negotiated policies
  - Policy tools are part of trusted client base
  - Queries are logged so after the fact non-compliant queries can also be identified



Introduction

- We view policy assurance and authorization as parts of a broader goal: accountability
- In several application contexts, strictly enforced, before-the-fact authorization of every action is insufficient
- Sometimes it is more appropriate to analyze actions after-the-fact and hold policy violators accountable
  - Unexpected circumstances
  - No single action leads to a violation but a combination of actions does
  - User is authorized to access resource/data but misuses it after getting access
- Accountability framework requirements
  - expressive policy language and reasoner
  - logging and provenance middleware
  - justification generation and interface

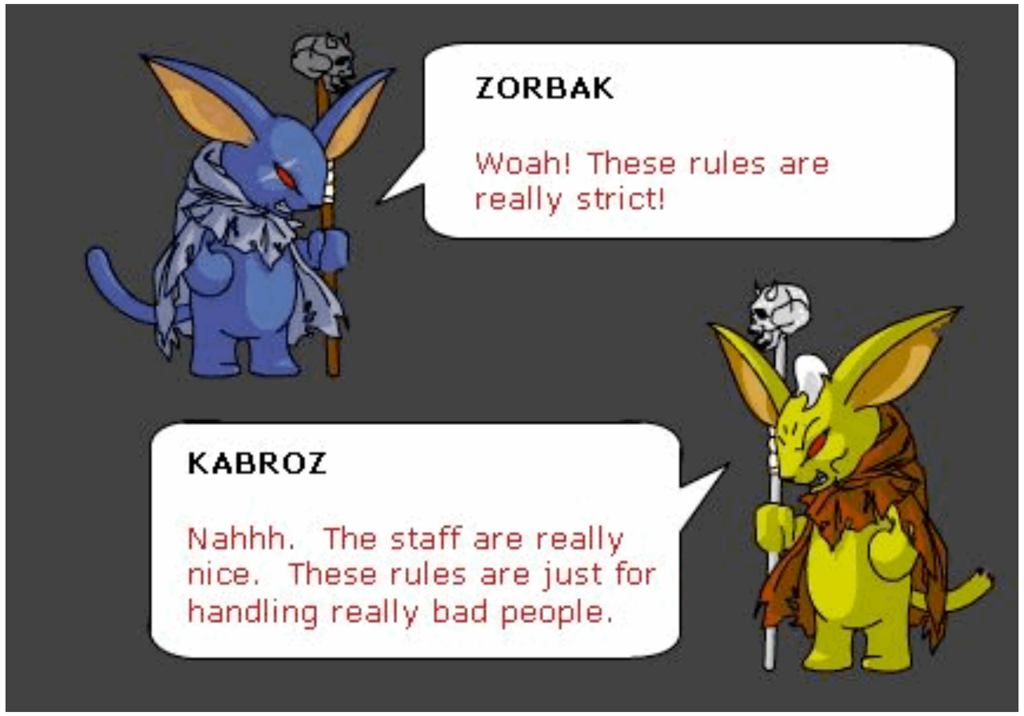


Image courtesy of Adventure Quest http://www.battleon.com/

Motivation

- Why do we need policy assurance for PIR queries ?
  - Queries and results are not revealed to the database administrator/owner
  - Possible that queries violate policy and leak information
  - Client can ensure that he/she meets the policies
    - before-the-fact and only send compliant queries to the PIR database
    - after-the-fact to understand the policies and learn to formulate compliant queries in the future

Example

- Policy
  - The user may not query specifically for people living in New England
- Compliant query
  - SELECT name and age FROM people WHERE zipcode="21244"
  - SELECT * FROM people WHERE last_name="Smith"
  - SELECT * FROM people WHERE birth_city="Cambridge"
- Non-compliant query
  - SELECT openid and ssn FROM people where city=”Boston”
  - SELECT * FROM people WHERE State="MA"

Problem Statement

- What tools and techniques are required to prove that PIR queries are compliant or non-compliant with policies
  - What do these policies look like ?
  - What are the policies based on ?
  - How can these policies be expressed ?
  - How can policy compliance/non-compliance be identified ?
  - Is just identifying non compliance sufficient ?
  - If not, how can the reason for compliance/non-compliance be appropriately explained ?

Challenges in Policy Assurance

- Policy structure
  - dependent on query structure
    - SPARQL Query Language for RDF
    - similar to sql but for Semantic Web data

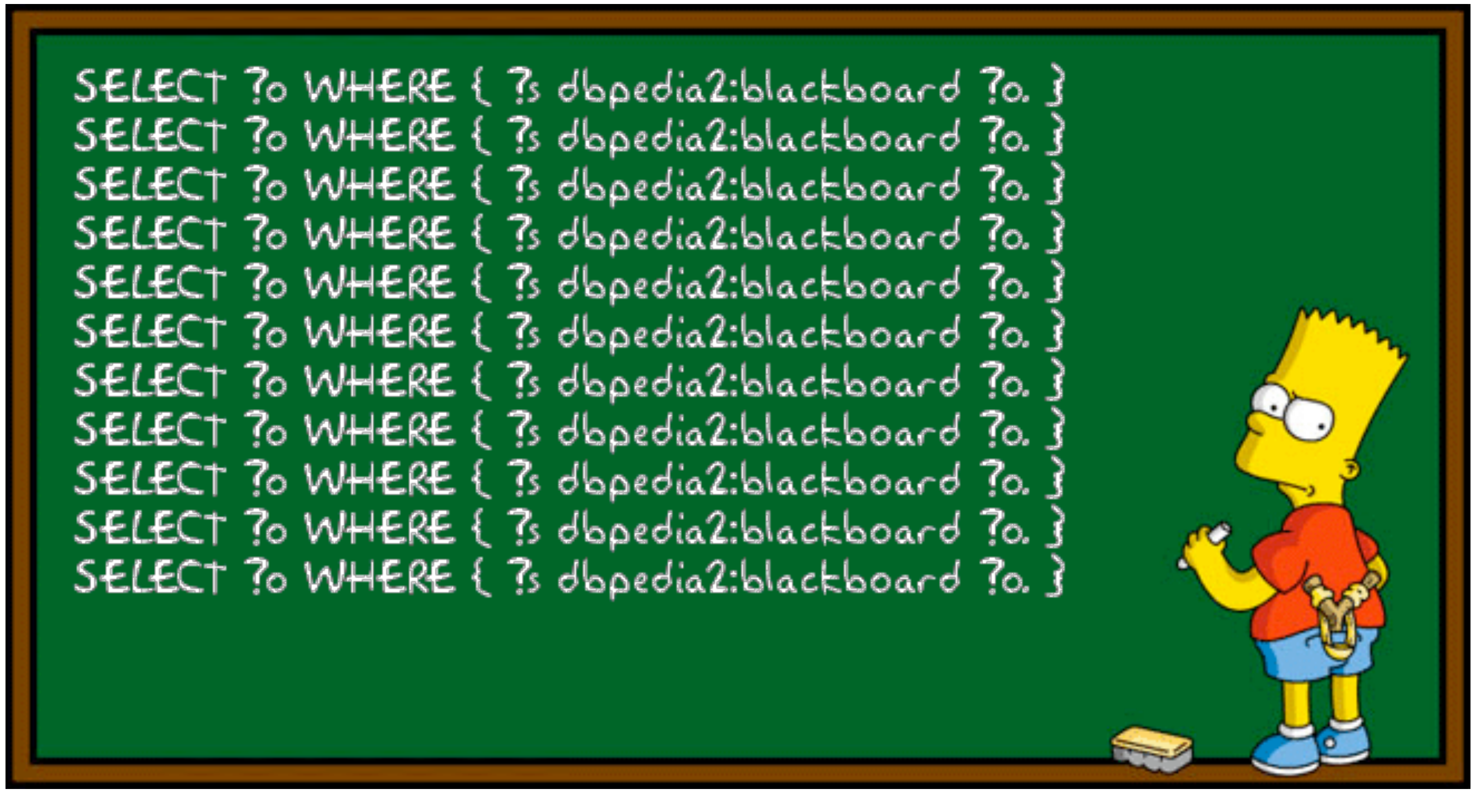


Image courtesy http://www.snee.com/bobdc.blog/

Challenges in Policy Assurance

- Policy Structure
  - Customizable by database
    - columns,values of the columns
    - range, domain, instance, or subclass of column or column value
  - Integrate data external to the current domain
  - Span query log/history

Challenges in Policy Assurance

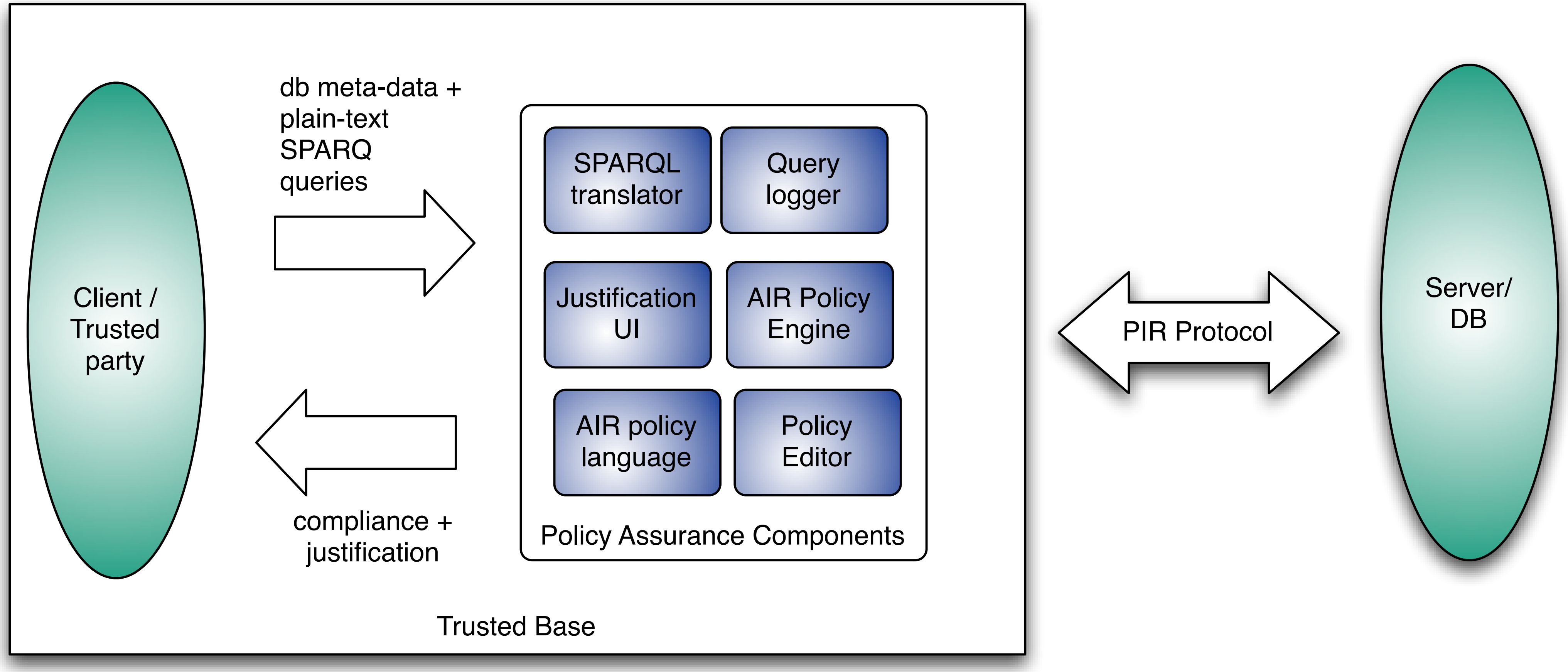
- Policy language
  - Policies are based on different kinds of data and conditions
    - For example, “Access to marital status, gender, and religion for US citizens is not permitted”
    - Need to understand and capture what it means to be a US citizen
  - Policies tend to deal in abstract terms and talk about kinds of information that should not be accessible or should not be used for certain purposes
    - For example, “Access to contact information for minors is not permitted”, or “my health information cannot be used to contact me regarding experimental drugs”
    - Need to understand that contact information includes email, mailing address, telephone num, fax num.

Challenges in Policy Assurance

- User Interfaces
  - Policy authoring
    - What input needs to be provided for automated policy generation ?
  - Justification UI
    - How to display meaningful portions of the justification ?

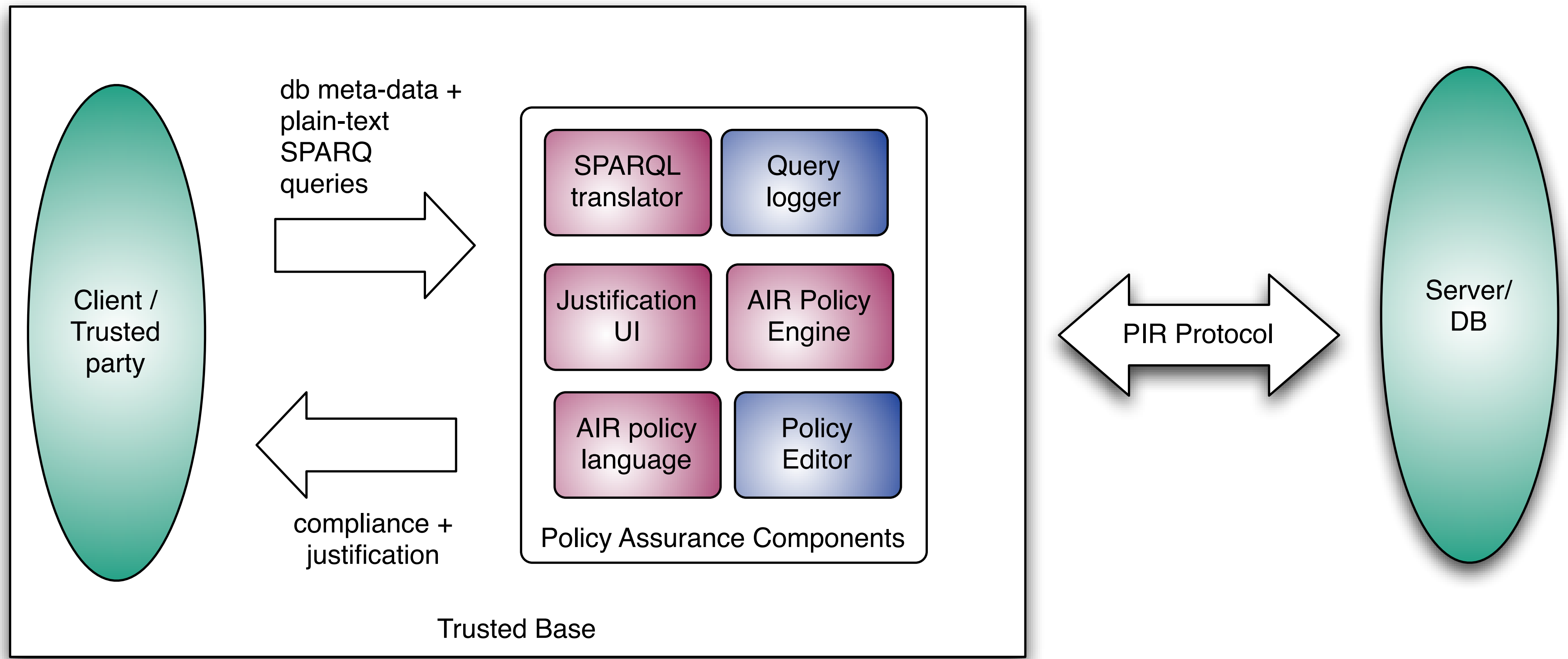
Technical Approach

- Policy Assurance Components
  - AIR Policy Language
  - AIR Reasoner
  - SPARQL Translator
  - Query Logger
  - Policy Editor
  - Justification UI



Technical Approach

- Policy Assurance Components
  - AIR Policy Language
  - AIR Reasoner
  - SPARQL Translator
  - Query Logger
  - Policy Editor
  - Justification UI



AIR Policy Language

- a machine-understandable policy language
- Semantic Web technologies for shared model of queries and policies
- Why Semantic Web ?
  - Need to ground terms on common models of data and knowledge so that data can be exchanged and used between different systems with some assurance of its meaning
  - Semantic Web technologies offer some good advantages
    - shared model of discourse
    - global unique identifiers
    - open & dynamic
    - interoperability - mapping between concepts and instances possible

Image courtesy of http://www.cartoonbank.com/

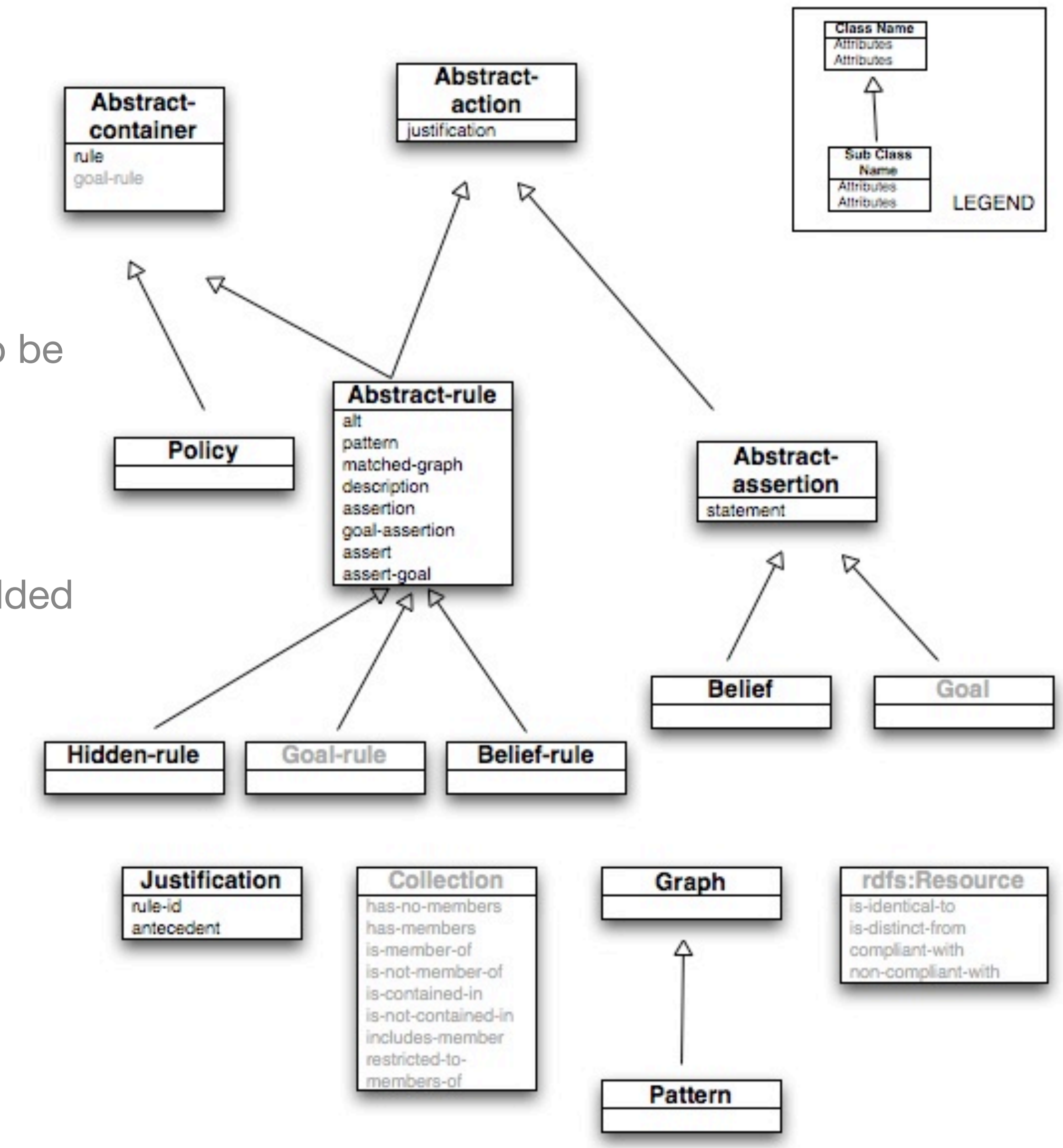
AIR Policy Language

- AIR policies are written in Terse RDF Triple Language (Turtle)
- Each AIR policy consists of one or more rules
  - policy = { rule }
- A rule is made up of a pattern that when matched causes an action to be fired. Optional: variable, description
  - rule = { pattern, action [variable description]}
- An action can either be an assertion, which is a set of facts that are added to the knowledge base or a nested rule
  - action = { [assert | assertion] | rule }

```
:MyFirstPolicy a air:Policy;
    air:rule [
        air:pattern { ... };
        air:assertion { ... };
        air:rule [ ... ]
    ].
```

- Third version of AIR to be released in Fall with simpler syntax

```
:MySecondPolicy a air:Policy;
    air:rule [
        air:if { ... };
        air:then { ... };
        air:else { ... }
```



Policy Format

- PIR policies written in terms of retrieving and filtering
  - Queries retrieve certain values
    - Example: The user may / may not retrieve attribute X
  - Queries use certain values as conditions or filters
    - Example: The user may filter based only on X or Y
- AIR policies for PIR queries use properties from the SPARQL translation ontology
  - retrieve property deals with values that are output
  - clause property deals with filter conditions

```
:SSN_Rule1 a air:BeliefRule;
	air:label "SSN Retrieval Rule 1";
	air:if {
		:Q a s:SPARQLQuery;
			s:retrieve :VL;
			s:clause :C.
		:VL s:var :V1.
		:C s:triplePattern [ log:includes { [] db:ssn :V1 } ].
	};
	air:description (:Q " is a SPARQL Query and retrieves SSN values");
	air:then { air:assert {:Q air:compliant-with :SSN_Policy} }.
```

Supporting Database Metadata

- Database meta-data provided in Semantic Web format
- Metadata not restricted to any structure or ontology
  - Example: Person a Class; with name, ssn, email, address, telephone as properties. address has several properties - street, house number, state, city, and zipcode, etc.
- Abstract data support
  - Example policy: “Access to contact information of minors is not permitted”
  - Example: “My health information cannot be used to contact me regarding experimental drugs”
  - contact information and health information are not individual attributes but a collection of several values or instances

Supporting Database Metadata

- Abstract policy support
  - Contact info and health info can be described in multiple ways
  - Grouping column names
    - Example: Contact details is a group containing email, address, telephone, fax, office add
  - Ontological relationship between column names
    - Example 1: HealthInfo is a Class with currentSymptoms, currentPrescriptions, pastPrescriptions properties
    - Example 2: HealthInfoData is a class and CurrentSymptoms, CurrentPerscriptions etc are instances

Integration with Semantic Web Data

- AIR policy language allows referring to and using any SW data
  - Example policy: The user may not query specifically for people living in New England
  - Input: SW data to allow reasoner to infer meaning of New England
  - Input: SW data to allow reasoner to infer lives-in is abstract data type that maps to database attributes city, state, and zipcode

```
:NewEngland a :Region.
:MA a :State; :in :NewEngland.
:NY a :State.
:CT a :State.
:Boston a :City; :in :MA.
:Cambridge a :City; :in :MA.
:02139 a :zipcode; :in :Cambridge.
@forAll :A, :B, :C.
{ :A :in :B.
  :B :in :C
} => { :A :in :C }.
```

Simple rules defining NE region

```
db:LivesIn a rdf:List;
      rdf:first db:city;
      rdf:rest
           ( db:state
            db:zipcode
            ).
```

Grouping of database meta-data

Integration with Semantic Web Data

- Authentication information can also be provided in SW format
  - Example: CSAIL members may not query specifically for people living in New England
  - Along with providing SW data about what it means to be "living in" and how "New England" can be inferred, authentication and/or user credential information can also be provided

```
:ABC a s:Requester;
     foaf:openid <http://people.csail.mit.edu/lkagal/foaf#me>.

@forAll :U.
{ <http://csail.mit.edu/members.rdf>.log:semantics log:includes
          { :U  foaf:openid <http://people.csail.mit.edu/lkagal/foaf#me> }
} => { :U db:group db:CSAIL }.
```

Inferring group membership of requester

Integration with Semantic Web Data

- Example policy: CSAIL group members may not query specifically for people living in New England

```
:NE_Rule1 a air:BeliefRule;
	air:label "New England Rule 1";
	air:pattern {
		:Q a s:SPARQLQuery;
			s:retrieve :VL;
			s:clause :C;
			s:user :U.
		:U dg:group db:CSAIL.
	};
	air:description (:Q " is a SPARQLQuery and the requester belongs to CSAIL");
	air:rule :NE_Rule2.

:NE_Rule2 a air:BeliefRule;
	air:label "New England Rule 2";
	air:pattern {
		:VL s:var :V1.
		:C s:triplePattern [log:includes {[] :L :V1}].
		:L list:in db:LivesIn.
	};
	air:description (:Q " contains a lives-in attribute " :L);
	air:rule :NE_Rule3;
	air:alt [ air:assert {:Q air:compliant-with :NE_NewPolicy}].

:NE_Rule3 a air:BeliefRule;
	air:label "New England Rule 3";
	air:pattern {
		:V1 db:in db:NewEngland
	};
	air:description ("The user is filtering on " :L " with value set to " :Y ", which is in New
England. The user belongs to CSAIL and may not query specifically for people living in
New England");
	air:assert {:Q air:non-compliant-with :NE_NewPolicy};
	air:alt[ air:assert {:Q air:compliant-with :NE_NewPolicy}].
```

General Types of Policies

- To enable thinking about and expressing policies, we've defined some broad types of policies
- Restriction / Black List
  - The user may not retrieve/filter X, Y or Z
  - Example policy:The user may not retrieve ssn, dob, or address
- Conditional Restriction / Black List
  - The user may not retrieve/filter X, Y or Z if condition
  - Example policy:The user may not retrieve ssn, dob, or address if age < 18
- Permit / White List
  - The user may only retrieve/filter X,Y and Z
  - Example policy: The user may only filter on first_name, last_name, workplace, work add

General Types of Policies

- Conditional White List / Permit
  - The user may retrieve/use X,Y and Z if (condition)
  - Example policy: The user may retrieve photos if age > 18
- Inclusion
  - if you retrieve/filter A you should/should not retrieve/filter B
  - Example policy: The user may retrieve first_name, last_name if he does not filter on ssn

AIR Reasoner

- Production-rule system in python
- Uses dependency tracking to generate justifications for compliant and non-compliant queries

```
@prefix : <http://dig.csail.mit.edu/data#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#> .
@prefix tms: <http://dig.csail.mit.edu/TAMI/2007/amord/tms#> .
@prefix yosi: <http://dig.csail.mit.edu/People/yosi#> .

  :DAP_1     tms:justification tms:premise .

  :DAP_3     tms:description  (
    :Req2
    " is a request made by a requester with openid, "
    <http://auth.mit.edu/syosi>
    ", for DIG resource "
    <http://dig.csail.mit.edu/proposals/nsf.tex/> );
    tms:justification  [
      tms:antecedent-expr  [
        a tms:And-justification;
        tms:sub-expr :DAP_1,
            {:DIG     :owns
              <http://dig.csail.mit.edu/proposals/nsf.tex/> .
             :Req2     a air:Request;
                  air:resource
                  <http://dig.csail.mit.edu/proposals/nsf.tex/>
;
             foaf:openid <http://auth.mit.edu/syosi> .
            } ];
      tms:rule-name :DAP_1 ] .

  :Req2    air:compliant-with :DIGPolicy .
```

1

```
  {
    :Req2     air:compliant-with :DIGPolicy .
  }
  tms:description  (
    "The requester with openid, " <http://auth.mit.edu/syosi>
    ", is known to a DIG member, "
    <http://dig.csail.mit.edu/People/RRS> );
  tms:justification  [
      tms:antecedent-expr  [
        a tms:And-justification;
         tms:sub-expr :DAP_3,
        {<http://dig.csail.mit.edu/People/RRS>    air:in
                :MemberList;
                foaf:knows yosi:YES .
          yosi:YES foaf:openid  <http://auth.mit.edu/syosi> .
        } ];
      tms:rule-name :DAP_3 ] .

  {
    <http://dig.csail.mit.edu/People/RRS>     air:in
:MemberList;
    foaf:knows yosi:YES .
    yosi:YES    foaf:openid  <http://auth.mit.edu/syosi> .
    :DIG     :owns <http://dig.csail.mit.edu/proposals/nsf.tex/> .
    :Req2     a air:Request;
      air:resource <http://dig.csail.mit.edu/proposals/nsf.tex/>;
      foaf:openid <http://auth.mit.edu/syosi> .
  }  tms:justification tms:premise .
```

2

Part of justification generated by reasoner

Justification User Interface

- AIR reasoner generates proofs of compliance and non-compliance
- Proofs are not easy to understand
- Graphical justification interface that provides an explorable structured natural language explanation for policy compliance and non-compliance
- Part of Tabulator, a Semantic Web browser
- Available as a Firefox extension

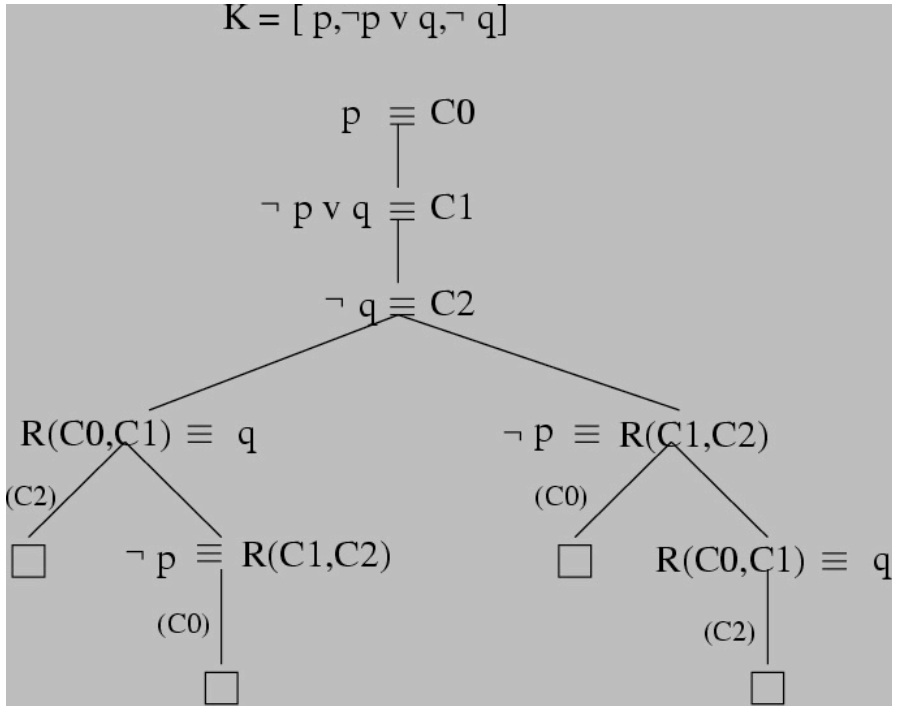
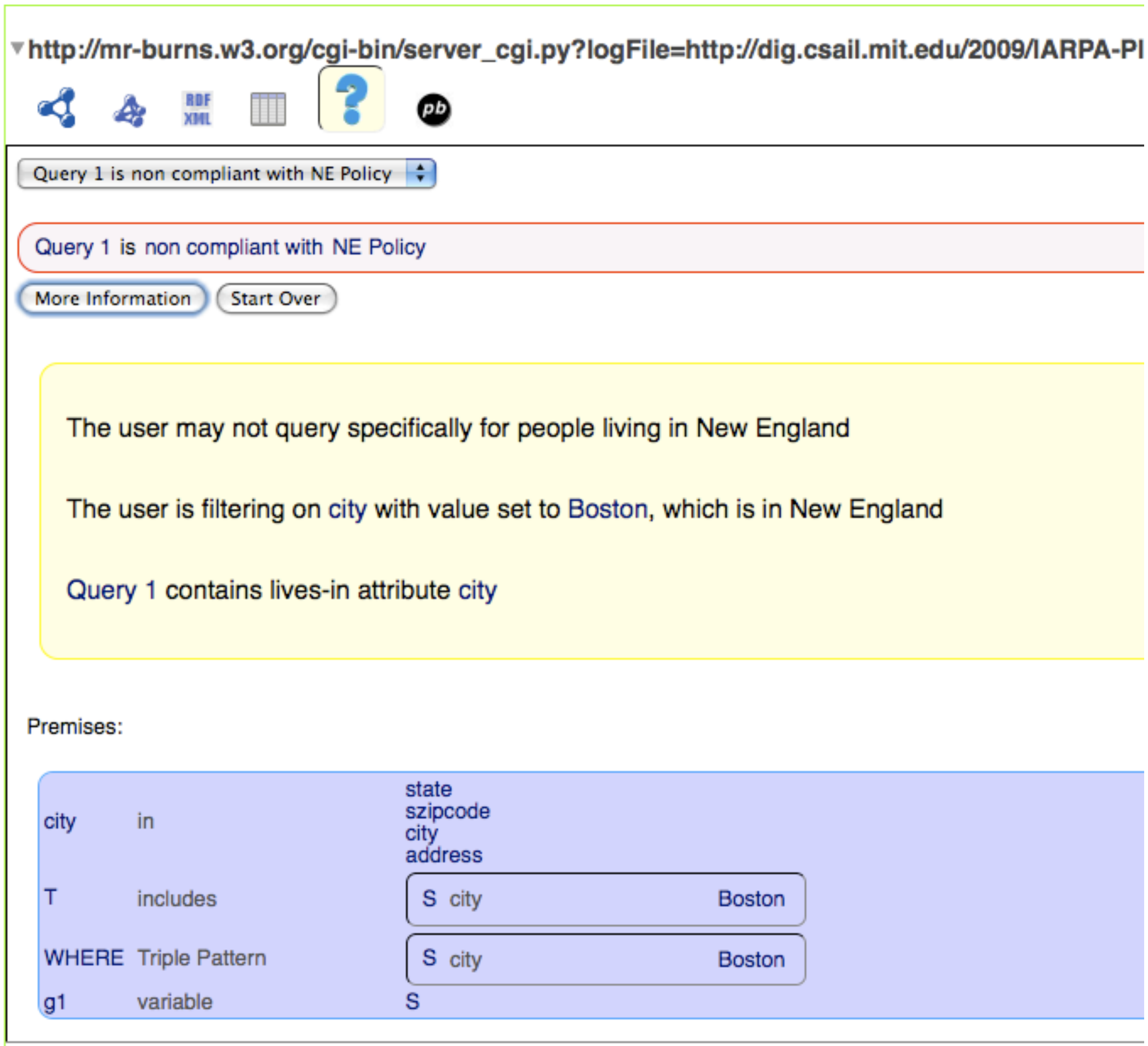


Image courtesy http://clip.dia.fi.upm.es/~logalg/slides/

Justification User Interface



SPARQL Translation

- Why should we translate the query language ?
  - RDF-based tools - AIR reasoner and Justification UI
  - SPARQL is not in RDF
  - Example query: List of the age and openid URIs of all adults living in Boston

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX db: <http//dig.csail.mit.edu/db#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?s ?id ?n WHERE {
  ?s db:city db:Boston.
  ?s db:age ?a.
  ?s foaf:openid ?id.
  FILTER (?a > 18)
}
```

Example SPARQL query

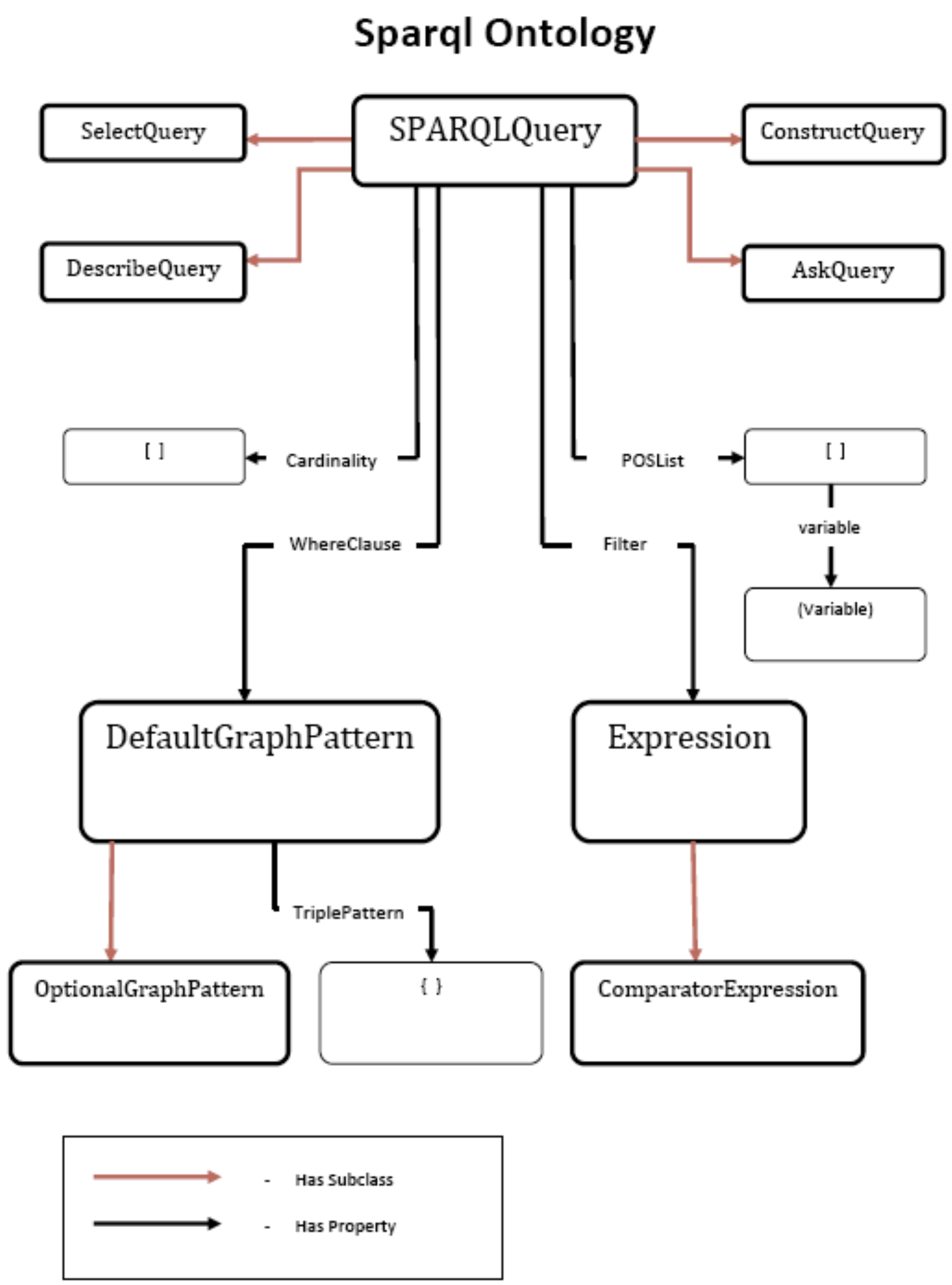
SPARQL Translation

- SPARQL translation ver 1
  - Detailed SPARQL ontology in RDF
  - Captured SPARQL semantics
  - Lead to lengthy and complex policies

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix db: <http//dig.csail.mit.edu/db#> .
@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .
@prefix : <http://dig.csail.mit.edu/2009/IARPA-PIR/query1#> .

:Query-1 a s:SPARQLQuery;
   s:cardinality :ALL;
   s:POSList [
     s:variable :S;
     s:variable :ID;
     s:variable :N;
    ];
   s:WhereClause  :WHERE.

   :WHERE a s:DefaultGraphPattern;
 s:TriplePattern  { :S db:city db:Boston };
       s:TriplePattern  { :S db:age :A };
       s:TriplePattern  { :S foaf:openid :ID };
       s:Filter [
             a s:ComparatorExpression;
          s:TriplePattern { :A s:BooleanGT "18"^^xsd:integer }
       ].
```



SPARQL Translation Ontology
Version 1

SPARQL Translation

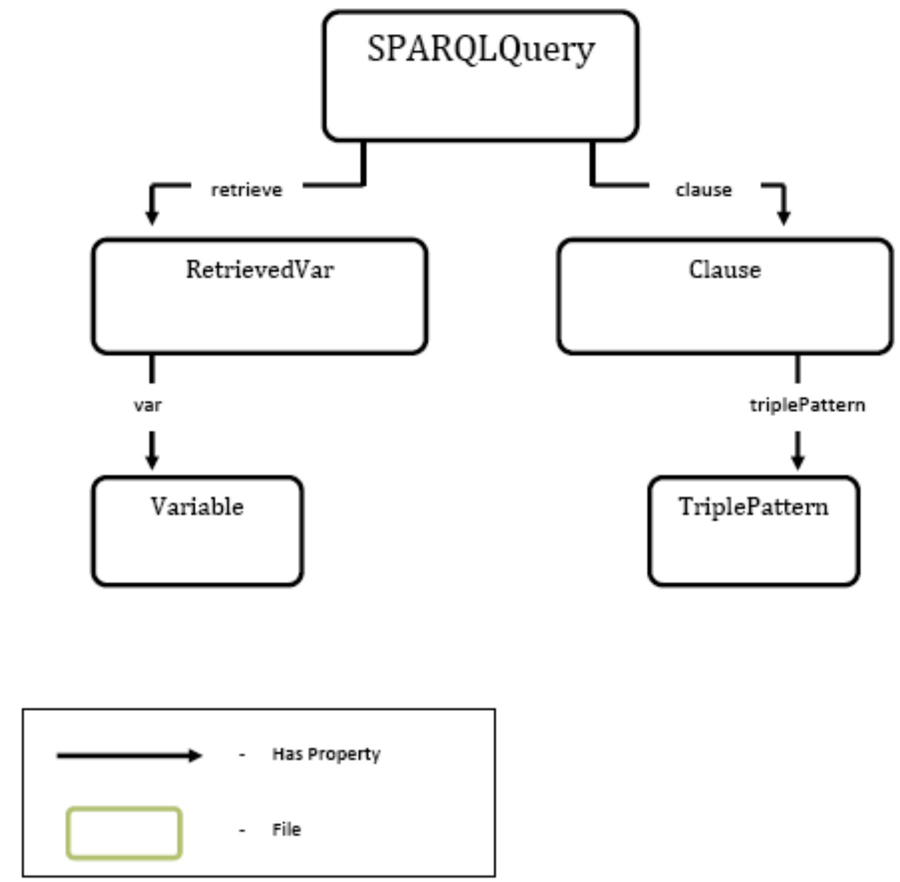
- SPARQL translation ver 2
  - Simple, high level ontology in RDF
  - Does not capture SPARQL semantics
  - Lead to smaller, easier to specify policies

```
@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .

:Query-1996945348 a s:Query;
   s:VarList [
     s:variable :s;
     s:variable :id;
     s:variable :n;
   ];
s:Clause [
  s:TriplePattern  { :s <http//dig.csail.mit.edu/db#city> <http//dig.csail.mit.edu/db#Boston> };
  s:TriplePattern  { :s <http://xmlns.com/foaf/0.1/age> :a };
  s:TriplePattern  { :s <http://xmlns.com/foaf/0.1/openid> :id };
  s:TriplePattern  { :a s:BooleanGT "18 "};

].
```

Translation of SPARQL query



SPARQL Translation Ontology
Version 2

SPARQL Translator

- Converts SPARQL into RDF using Translation Ontology Version 2
- Available as a Web service

SPARQL to N3 Query Conversion

Part of the IARPA-PIR Project

Please enter the SPARQL query you would like to translate. Here's an example, or enter your own. Here are some great test cases.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX db: <http//dig.csail.mit.edu/db#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?s ?id ?n WHERE {
  ?s db:city db:Boston.
  ?s db:age ?a.
  ?s foaf:openid ?id.
  FILTER (?a > 18)
}
```

Translate

Reset

The translated output is:

```
@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .

:Query-1361427280 a s:Query;
   s:VarList [
     s:variable :s;
     s:variable :id;
     s:variable :n;
   ];
s:Clause [
  s:TriplePattern  { :s <http//dig.csail.mit.edu/db#city> <http//dig.csail.mit.edu/db#Boston> };
  s:TriplePattern  { :s <http//dig.csail.mit.edu/db#age> :a };
  s:TriplePattern  { :s <http://xmlns.com/foaf/0.1/openid> :id };
  s:TriplePattern  { :a s:BooleanGT "18 "};

].
```

Policy Authoring

- Python back-end
- Web front-end
- Process
  - Select type of policy
  - Specify retrieve or filter
  - Specify col names/ abstract names and values

Inclusion Policy Creator

A **Inclusion Policy** does not allow a person to use, retrieve, or both use and retrieve a variable with a certain attribute unless another variable with a certain other attribute is also retrieved or used.

Policy Name (Optional)

Please include a policy name if so desired

Policy Description (Optional)

Please provide a policy description if desired.

Included Attributes

Please list the attributes desired and the condition which you require along with their retrieval.

Variable:
○ Cannot Use ○ Cannot Retrieve ○ Both

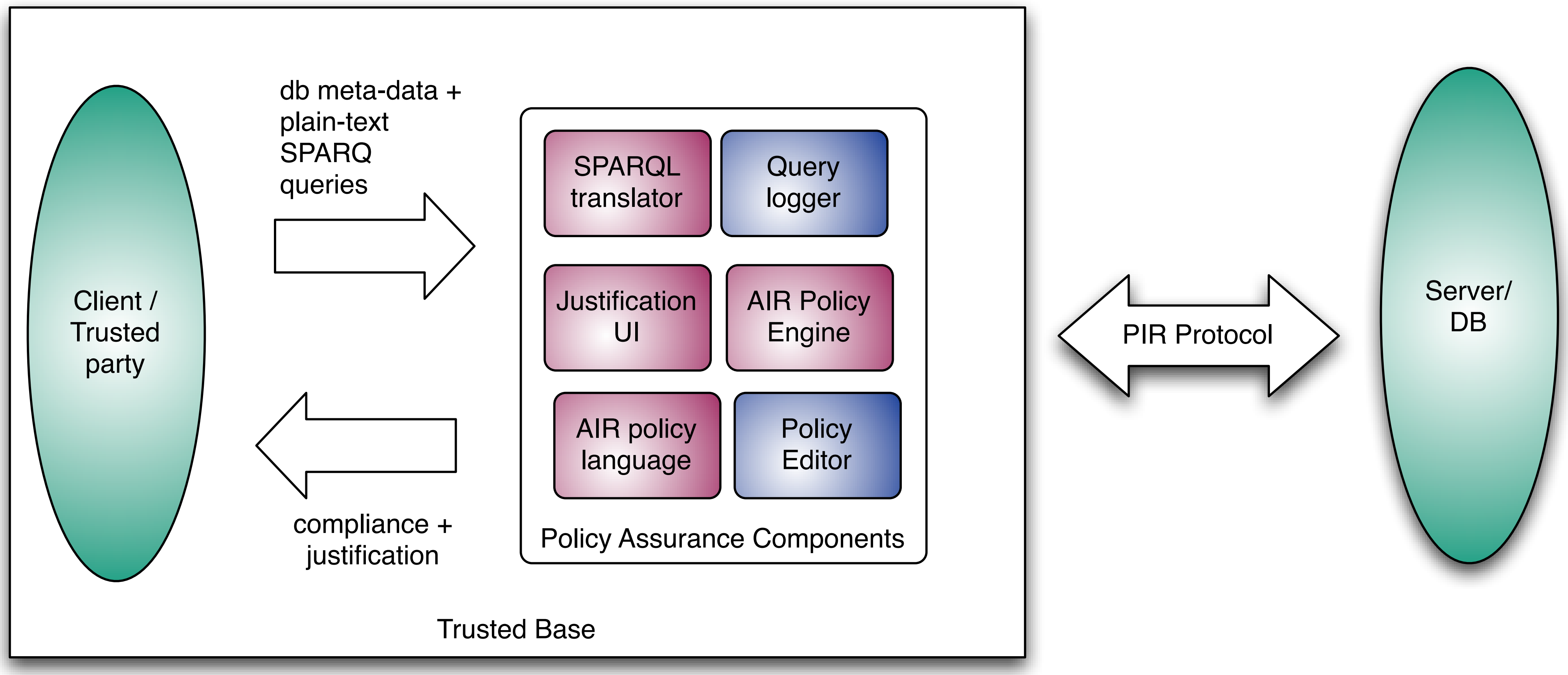
Condition (Optional):
○ Cannot Use ○ Cannot Retrieve ○ Both

Add Another Variable

Submit!

Mockup of Policy Authoring Tool

Next Steps


Client /
Trusted
party
db meta-data +
plain-text
SPARQ
queries
SPARQL
translator
Query
logger
Justification
UI
AIR Policy
Engine
AIR policy
language
Policy
Editor
Policy Assurance Components
compliance +
justification
Trusted Base
PIR Protocol
Server/
DB

Next Steps

- User Interface
  - Python back-end to be completed
  - Policy authoring Web form to be completed
  - Add log based policy generation support to policy authoring UI
  - Import ontologies in UI to define policies
- Provide persistent log for queries
- Support history/log based reasoning

Next Steps

- Policy language
  - default policies - compliant unless proved non-compliant or vice versa
  - conflict resolution
- Wrapper script to accept queries and send them to reasoner and return results
- Convert sets of queries and policies prepared by the test and evaluation team into SPARQL queries and AIR policies

$$\frac{N}{N_{correct} + 1.5*N_{fp} + 2*N_{fn}}$$

where,
N is total number of queries
$N_{correct}$ is the number of queries correctly classified
N_{fp} is the number of queries incorrectly classified as violating policy
N_{fn} is the number of queries incorrectly classified as conforming to policy

Policy Assurance metric

Summary - Research & Contributions

- Framework
  - Helps users conform to policies or learn how to form compliant queries
- Policy language
  - extended to support PIR queries
  - support for database meta-data
  - abstract data types for high level policies
  - integration with external SW data
- Policy UI
  - encourages policy administrators to think clearly about their policies and express them explicitly
  - justification UI helps debug policies and queries

Accountability Projects at DIG

- This project is part of larger effort at DIG aimed at policy-awareness & accountability
  - Some other accountability projects include
    - Policy-aware mash-ups
    - Fusion Center project
    - Social Web Privacy or Respect My Privacy
    - License validator & Semantic Clipboard

References

- Policy Assurance for PIR Queries, http://dig.csail.mit.edu/2009/IARPA-PIR/
- TAMI project, http://dig.csail.mit.edu/TAMI
- Tabulator extension, http://dig.csail.mit.edu/2007/tab/
- AIR specifications, http://dig.csail.mit.edu/TAMI/2008/12/AIR
- Paper on AIR, http://dig.csail.mit.edu/2008/Papers/IEEEPolicy