

**IEEE POLICY 2005  
Stockholm, Sweden**



# **Driving and Monitoring Provisional Trust Negotiation with Metapolicies**

Piero A. Bonatti, Università di Napoli Federico II  
Daniel Olmedilla, L3S Research Center and Hanover University  
June 4<sup>th</sup>, 2005

# Outline

- Motivation
- The rule language
- Metapolicies
- More applications of metapolicies
- Conclusions & Further work

# Motivation (I)

The term **policy** refers to ...

- **Security Policies: pose constraints on the behavior of a system**
- **Trust Management Policy Languages: typically used to collect user properties in open environments**
- **Business Rules: statements about how a business is done**

**In addition, associated to policies one needs to execute actions. Therefore also relevant:**

- **Action Languages: used in reactive policy specification to execute actions**

# Motivation (& II)

## Integration of policies

Although many approaches have been described to address the above points, there is no common solution, integrating them all in a single framework.

# The rule language (I)

## Specification

Based on normal logic program  $A \leftarrow L_1, \dots, L_n$

Categories of predicates are

- **Decision Predicates:**

- **Allow():** queried by the negotiation for access control decisions
- **Sign():** used to issue statements signed by the principal owning the policy

- **Abbreviation/Abstraction Predicates**

- **Constraint Predicates:** comprise usual equality and disequality predicates

- **State Predicates:** decisions according the state

- **State Query Predicates:** read the state without modifying it
- **Provisional Predicates:** may be made true by means of associated actions that may modify the current state
  - E.g. `credential(C,K)`, `declaration()`, `logged(X,logfile_name)`

# The rule Language (& II)

## Design Assumptions ...

Provisional actions are orthogonal

- The action associated to any ground atom A cannot change the truth value of any other ground provisional atom.

Exchange of filtered set of policies between parties

- in order to avoid combinatorial explosion of requests

Negation is not applied neither to provisional predicates nor to any predicate occurring in a rule head

# Metapolicies (I)

## Current valid attributes

Attribute	Domain	Range
action	provisional predicates	commands
actor	provisional predicates	self, peer
aggregation_method	cost and sensitivity attributes	max, min, sum, adopt(Predicate)
cost	provisional predicates	number
evaluation	state predicates	immediate, delayed, concurrent
expected_outcome	provisional predicates	success, failure, undefined, unknown
explanation	literals and rules	string expression
ontology	abbreviation predicates, credentials, declarations, actions	URI
predicate	literals	predicate names
selection_method	negotiator	certain_first, order(attribute_list), adopt(Predicate)
sensitivity	predicates, literals, rules	public, private, not_applicable
type	predicates, literals	abbreviation, constraint, decision, state_predicate, provisional, state_query

# Metapolicies

## Examples

```
table(Key,Data).evaluation:immediate ←  
    ground(Key).
```

```
logged(Msg,File).action:'echo'+Msg+'>'+File.
```

```
credential(_).ontology:URI.
```

```
abbrev(_).explanation:"this condition checks..."
```



# Policy filtering

## Semantics-preserving ...

### Removing irrelevant rules

- only the relevant subset of the policies is selected

### Evaluating State Predicates

- Partial evaluation

### Compiling Private Policies

- Internal structure of the rules is lost

### Abbreviate Predicate Renaming

- avoiding that meaningful predicate names disclose confidential information about the policy

# Policy filtering

## Win information loss ...

### Blurring

- some rules may have to be hidden and blocked until the client is trusted enough
- sensitive state predicates may have to be blocked until their evaluation does not disclose confidential information.
- replaced with a reserved propositional symbol

allow(enter site()) ←  
declaration( usr = U; passwd = P), blurred.

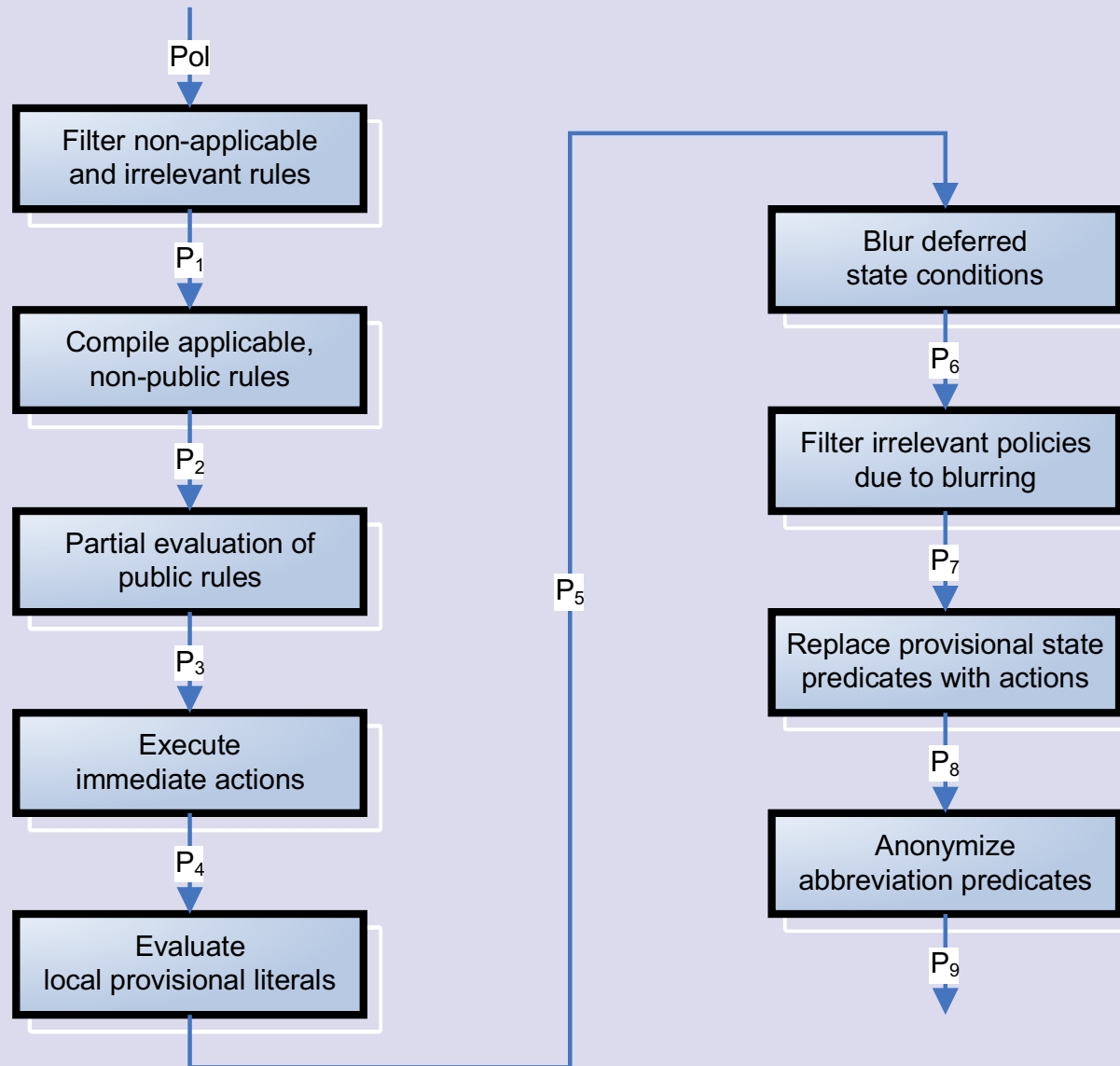
### Expectation

- what-if queries require the server to evaluate a request without executing immediate actions during such an evaluation

$\hat{R}.expected\_outcome : Val \mid Val \in \{success, failure, undefined, unknown\}$

# Policy Filtering (& II)

## Driving filtering with metapolicies



# More applications of metapolicies

## Credential and action selection

**Candidate set:** a set of credentials and actions occurring in the proof of a goal  $G$  given a set of (filtered) policies  $P$ .

A user may have different candidate sets and therefore a selection mechanism. Typical measures are:

- Number of action executions
- Distributed credential collection

But metapolicies can help on this issue according to

- sensibility of credentials disclosed
- cost of each action executed

`action.cost.aggregation_method:sum.`

`logged.cost:Number.`

# More applications of metapolicies

## Metalevel Constraints

Like metapolicy rules without head

$\leftarrow L_1, \dots, L_n.$

At design time

- E.g. Protecting specific combinations of credentials.

$\leftarrow \text{credential}(c_1, \_), \dots, \text{credential}(c_n, \_).$

At runtime

- Monitor policies and metapolicies at runtime

$\leftarrow X.\text{action}:A, A.\text{actor}:Y, A.\text{actor}:Z, Y \neq Z.$

# More applications of metapolicies

## Distributed Credentials

Credential gathering distinguishes between:

- Issuer
- Credential repository
- Credential owner
- Actor responsible for fetching the credential

Issuer is encoded in the credential and ownership can be checked via challenges.

`Credential.location:URI` and `Credential.actor:X`

allow encoding the repository and fetcher respectively.

# More applications of metapolicies

## Libraries and Language Extensions

Abbreviations and credentials can be linked to the ontologies that specify their meaning by means of a suitable metaattribute:

`Obj:ontology:URI`

This attribute may have multiple values because the contents of `Obj` may use symbols defined in different ontologies.

Metapolicy and abbreviation libraries can be exported and stored in standard formats, using RuleML and RDF/OWL.

# Conclusions & Further Work

Our main contributions are ...

A **trust management language** supporting general provisional-style actions

An **extendible declarative metalanguage** for driving decisions about

- Request formulation
- Information disclosure
- Distributed credential collection

A **parameterized negotiation procedure**

**Integrity constraints** for

- negotiation monitoring
- disclosure control

General **ontology-based techniques** for importing and exporting metapolicies and for smoothly integrating language extensions



# Conclusions & Further Work

## What we plan to do ...

Integrate event-condition-action (ECA) rules as

- some policies would be more naturally described under this paradigm
- It would extend the set of business rules directly supported

Study completeness issues, that in this context sound like: “Is negotiation always successful when the policies of the parties allow it?”

Natural language front-end to the policy domain

- Natural Language Processing (NLP)
- automatic generation of natural language explanations from proofs and filtered policies

# References

- REVERSE WG I2  
Policy Language, enforcement, composition  
<http://www.reverse.net/I2/>
- Policy Language Specification  
Project deliverable D2, Working Group I2, EU NoE  
REVERSE, Mar. 2005
- Security Agent in an Applet  
<http://www.l3s.de/~olmedilla/projects/trust/applet/instructions.html>
- PeerTrust project  
<http://sourceforge.net/projects/peertrust/>

**Thanks**

**Questions?**