A Rigorous Framework for Specification, Analysis and Enforcement of Access Control Policies

Rosario Pugliese

Università degli Studi di Firenze

Dipartimento di Statistica, Informatica, Applicazioni

Security Engineering A.A. 2017-18

Joint work with A. Margheri, M. Masi and F. Tiezzi
To appear in IEEE Transactions on Software Engineering

Outline

- An introduction to access control
- FACPL: a policy language for attribute-based access control systems
- Specification of FACPL policies
- Analysis of FACPL policies
- FACPL supporting tools
- Cloud IaaS scenario
  - FACPL at work on the scenario: policies
  - FACPL at work on the scenario: implementation
- Concluding remarks

An Introduction to *Access Control*

Access Control Systems

- The first line of defense for the protection of computing systems
- Defined by rules that establish under which conditions a subject's request for accessing a resource has to be permitted or denied
- Since the first applications in operating systems, to the more recent ones in distributed systems, many access control models have been proposed

Some Access Control Models

- Access Control Matrix: controls based on triples (user-action-resource)

	User1	User2
Res1	read	read, write
Res2	write	read, write

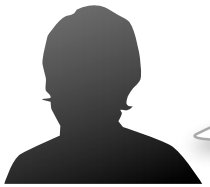
  - *Access Control Lists*
  - *Capability Lists*

- *Role-based* (RBAC): controls defined wrt specific roles
  - Pros Allows high-level design, user groups and hierarchy of groups
  - Cons Suffers from scalability and interoperability problems (it is essential to know in advance the role population)
  - Cons Defining fine-grained rules is tricky (rules cannot easily encompass information representing the evaluation context as e.g. system status or current time)

- *Attribute-based* (ABAC): controls based on attributes, i.e. any security-relevant information of the requester and/or system
  - Pros Differently grained, positive and negative rules
  - Pros Flexible and context-aware access control rules (expressive enough to uniformly represent all the other models)
  - Cons Need to combine possibly contrasting decisions

RBAC vs ABAC: an e-Health Scenario

Hi, I'm Julia, and I'm a physician from the famous Massachusetts General Hospital. I want to access your medical record for healthcare treatment



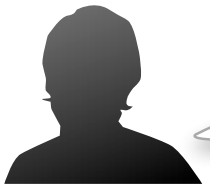
Hi, I'm Steve, and I'm a nurse from the Mount Auburn Hospital. I want to access your continuity of care document for dispensing Pepto-Bismol

Hi, we're Stan & Roger, we're researchers working at the WhiteHouse agency for public health. We would like to access your encounters history for statistical plans

Hi, I'm Homer, I'm the patient. I give access to my medical record to? ?!?!#*&^%&$(*&^????

RBAC vs ABAC: an e-Health Scenario

Hi, I'm Julia, and I'm a physician from the famous Massachusetts General Hospital. I want to access your medical record for healthcare treatment



Hi, I'm Steve, and I'm a nurse from the Mount Auburn Hospital. I want to access your continuity of care document for dispensing Pepto-Bismol

Hi, we're Stan & Roger, we're researchers working at the WhiteHouse agency for public health. We would like to access your encounters history for statistical plans

Hi, I'm Homer, I'm the patient. I give access to my medical record to? ?!?!#*&^%&$(*&^????

The patient *electronic health record* (EHR) must be controlled by the access control system in order to guarantee confidentiality of medical data

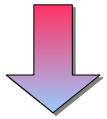
RBAC vs ABAC: an e-Health Scenario

Hi, I'm Julia, and I'm a physician from the famous Massachusetts General Hospital. I want to access your medical record for healthcare treatment

Hi, I'm Steve, and I'm a nurse from the Mount Auburn Hospital. I want to access your continuity of care document for dispensing Pepto-Bismol

Hi, we're Stan & Roger, we're researchers working at the WhiteHouse agency for public health. We would like to access your encounters history for statistical plans

- Different hospitals, different actions and different roles



- RBAC: difficult to define fine-grained rules
- No obvious way to encode such requests in a convenient way for a software actor

ABAC

- Requester credentials are rendered as a collection of attributes, i.e. pairs (*name*, *value*)
- Control carried out by positive/negative rules based on attribute values

An Attribute-based Language: the XACML Standard

The *eXtensible Access Control Markup Language* (XACML) is an OASIS standard

- is the widest-used implementation of the ABAC model
- defines an XML-based language for writing access control *policies*
- defines an XML-based language for representing *access requests*
- defines an authorisation workflow: *decision* and *enforcement* processes
- is currently used in many large scale *projects* (e.g., epSOS, NHIN)

- First normative specification: February 2003
- Current normative specification XACML 3.0: January 2013

An European eHealth Platform: the EU pilot epSOS

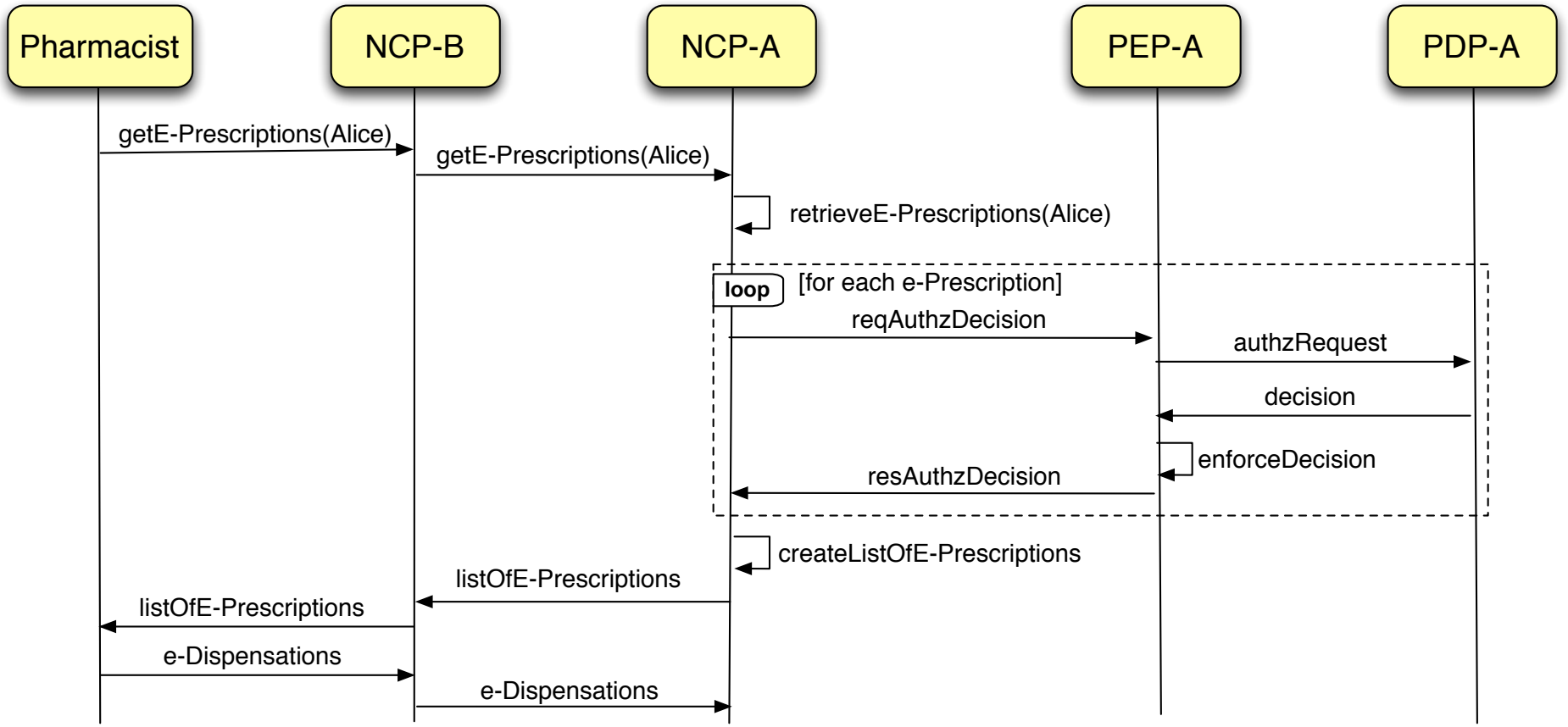
Objectives

- Exchanging patient data among European points of care
  - Facilitating the cross-board interoperability of European countries' healthcare systems
  - Complying with country-specific legislations
- Enforcing the *patient informed consent*
  - Ensuring confidentiality of *high sensitive* medical data

Resources and Services

- *Patient Summary*: the patient's medical data including all the important clinical facts
- *ePrescription*: the electronic prescription of a medicine by a legally authorised health professional
- *eDispensation*: the dispensing of the medicine to the patient as indicated in the corresponding ePrescription

ePrescription Service Protocol



- National Contact Point (NCP):
  - NCP B: from where the request is issued
  - NCP A: the patient's country of origin
- PDP-A takes an AC decision on the base of Alice's informed consent
- PEP-A enforces Alice's informed consent

An XACML Policy: excerpt of the e-Prescription Policy

```
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" ...
 RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-
      algorithm:permit-overrides">
 <Target>
  <AnyOf>
   <AllOf>
    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
      doctor
     </AttributeValue>
     <AttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#anyURI"
      ... />
    </Match>
   </AllOf>
  </AnyOf>
 </Target>
 <Rule RuleId="rule1" Effect="Permit">
  <Target> ... </Target>
  <Condition>
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-subset">
    ...
   </Apply>
  </Condition>
 </Rule>
 <ObligationExpression FulfillOn="Permit"
  ObligationId="urn:oasis:names:tc:xacml:obligation:log">
  ...
 </ObligationExpression>
</Policy>
```

The whole policy is $\approx$ 240 lines, the all epSOS policies are $\approx$ 500 lines

XACML Weaknesses

Designing XACML policies is a difficult and error-prone task

- The language has a verbose syntax
  - it makes writing XACML policies awkward by using common editors (XML is neither easily readable nor writable by human)
  - there exist ad-hoc policy editors, but they are cumbersome and ineffective when dealing with real-world policies
- XACML comes without a formal semantics
  - the standard is written in prose
  - it contains loose points that may lead to different interpretations (e.g., different implementation choices)
  - the portability of XACML policies could be undermined
  - devising analysis techniques is cumbersome

FACPL: a policy language for attribute-based access control systems

FACPL: Formal Access Control Policy Language

- Compact and expressive syntax for *attribute-based* access control policies and requests
- Formal semantics given in *denotational style*
- Formally grounded analysis techniques
- Java-based tools supporting Specification, Analysis and Enforcement of FACPL Policies

FACPL Attributes

- Attributes are pairs (*name*, *value*) and form access requests
- The name is structured in the form `Identifier/Identifier`, where the first identifier stands for a category name and the second for an attribute name
- Attribute values, which can be literals or sets, are accessed via names

E.g., given the attribute (`subject/id`, "*Andrea*"),

- the structured name `subject/id` stands for the value of the attribute `id` within the category `subject`
- the name `subject/id` is resolved to the value "*Andrea*"

A FACPL Specification

- **Policies**
  - a set of rules or policies
  - a combining algorithm to merge access decisions (e.g., `permit-overrides`, `deny-unless-permit`, `one-app`)
  - a target specifying to which requests the policy applies
  - a list of obligations specifying actions to be discharged
- **Rules**
  - an effect specifying a `permit` or `deny` access decision
  - a target
  - a list of obligations

Access Decisions

- `permit`: a policy grants the access
- `deny`: a policy forbids the access
- `not-applicable`: a policy does not apply to the access request
- `indeterminate`: a policy is unable to evaluate the access request

Obligations

- Are additional actions connected to the access control system
- Can correspond to, e.g., *updating a log file*, *sending a message*, *executing a command*
- Are discharged by the PEP through appropriate obligation services
- The PEP determines the enforced decision on the basis of the obligation results
- This decision could differ from the PDP one and is the overall outcome of the evaluation process

A FACPL Policy

```
PolicySet ePre { permit-overrides-all
target: equal("e-Prescription", resource/type)
policies:
Rule write (permit
  target: equal(subject/role, "doctor")
       && equal(action/id, "write")
       && in ("e-Pre-Write", subject/permission)
       && in ("e-Pre-Read", subject/permission))
Rule read (permit
  target: equal(subject/role, "doctor")
       && equal(action/id, "read")
       && in ("e-Pre-Read", subject/permission))
Rule pha (permit
  target: equal(subject/role, "pharmacist")
      && equal(action/id, "read")
      && in ("e-Pre-Read", subject/permission))
obl_p:
  [M log(system/time,resource/type,subject/id,action/id)]
}
```

The FACPL-based access control system of epSOS is defined in $\approx$ 40 lines, rather than $\approx$ 500 lines of the XACML one

FACPL Syntax (1 of 2)

Policy Auth. Systems	$PAS ::= (\texttt{pep:}\ EnfAlg\ \ \texttt{pdp:}\ PDP)$
Enforcment algorithms	$EnfAlg ::= \texttt{base} \mid \texttt{deny-biased} \mid \texttt{permit-biased}$
Policy Decision Points	$PDP ::= Policy \mid \{Alg\ \ \texttt{policies:}\ Policy^{+}\}$
Combining algorithms	$Alg ::= \texttt{p-over}_{\delta} \mid \texttt{d-over}_{\delta} \mid \texttt{d-unless-p}_{\delta} \mid \texttt{p-unless-d}_{\delta} \mid \texttt{first-app}_{\delta} \mid \texttt{one-app}_{\delta} \mid \texttt{weak-con}_{\delta} \mid \texttt{strong-con}_{\delta}$
Instantiation strategies	$\delta ::= \texttt{greedy} \mid \texttt{all}$
Policies	$Policy ::= Rule \mid \{Alg\ \ \texttt{target:}\ Expr\ \ \texttt{policies:}\ Policy^{+}\ \ \texttt{obl-p:}\ Obligation^{*}\ \ \texttt{obl-d:}\ Obligation^{*}\}$
Rules	$Rule ::= (Effect\ \ \texttt{target:}\ Expr\ \ \texttt{obl:}\ Obligation^{*})$
Effects	$Effect ::= \texttt{permit} \mid \texttt{deny}$
Obligations	$Obligation ::= [\,ObType\ \ Action(Expr^{*})\,]$
Obligation types	$ObType ::= \texttt{M} \mid \texttt{O}$

FACPL Syntax (2 of 2)

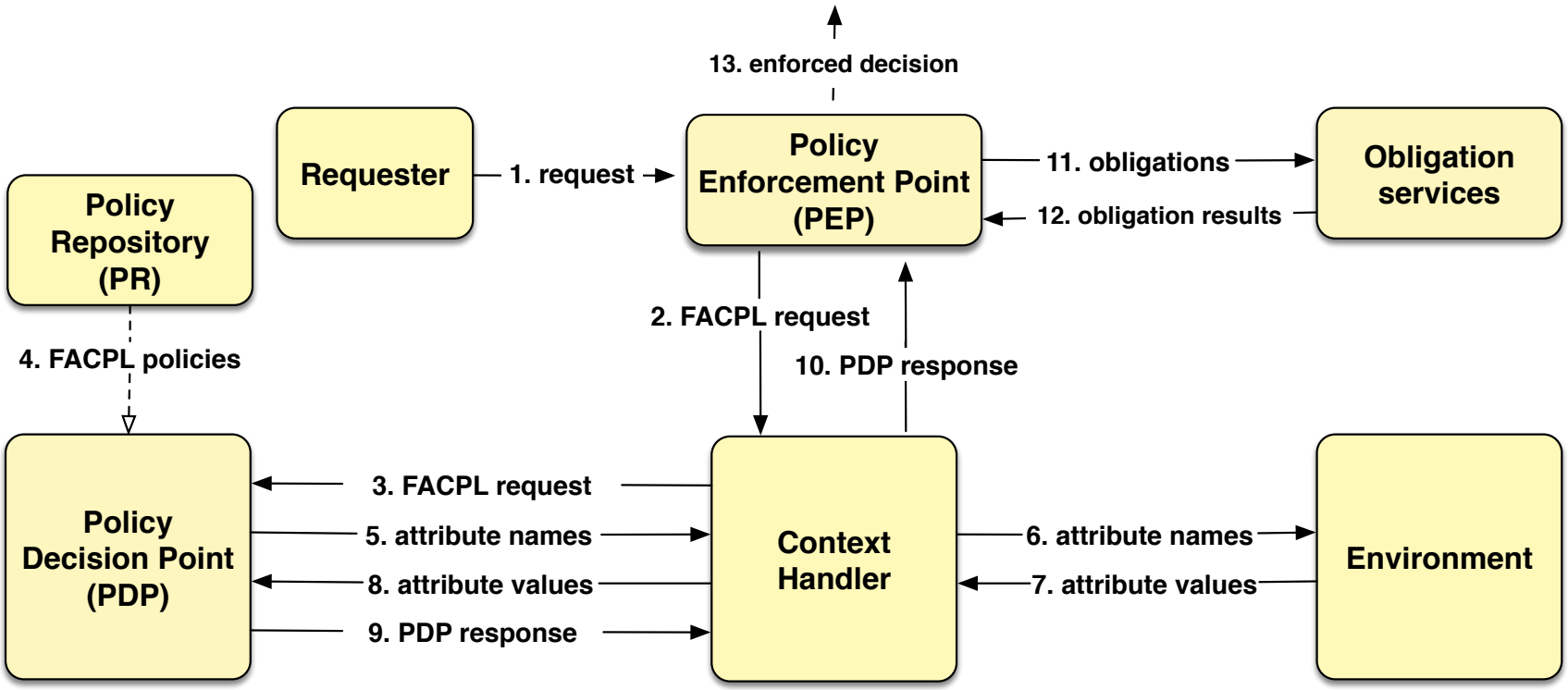
Expressions $Expr ::= Name \mid Value$
$\mid \texttt{and}(Expr, Expr) \mid \texttt{or}(Expr, Expr) \mid \texttt{not}(Expr)$
$\mid \texttt{equal}(Expr, Expr) \mid \texttt{in}(Expr, Expr)$
$\mid \texttt{greater-than}(Expr, Expr) \mid \texttt{add}(Expr, Expr)$
$\mid \texttt{subtract}(Expr, Expr) \mid \texttt{divide}(Expr, Expr)$
$\mid \texttt{multiply}(Expr, Expr)$

Attribute names $Name ::= Identifier / Identifier$

Literal values $Value ::= \texttt{true} \mid \texttt{false} \mid Double \mid String \mid Date$

Requests $Request ::= (Name, Value)^{+}$

FACPL Evaluation Process



- **PDP** decides whether to allow received requests and returns
  - a decision
  - a (possibly empty) list of obligations
- **PEP** enforces the decision taken by the PDP

FACPL Formal Semantics: given in a denotation style

The formal semantics is defined by following a denotational approach

- We introduce some semantic functions mapping each FACPL syntactic construct to an appropriate *denotation*, which is an element of a semantic domain representing the meaning of the construct
- The semantic functions are defined in a *compositional* way, so that the semantics of each construct is formulated in terms of the semantics of its sub-constructs

Notational convention

Application of the semantic functions is left-associative, parenthesis are omitted whenever possible, syntactic objects are surrounded with the brackets $[\![$ and $]\!]$ to increase readability, e.g. $\mathcal{E}[\![n]\!]r$ *stands for* $(\mathcal{E}(n))(r)$

FACPL Formal Semantics: requests

Semantics of a request: an element of the set

$$R \triangleq Name \to (Value \cup 2^{Value} \cup \{\bot\})$$

i.e., a *total function* that maps attribute names to either a literal value, or a set of values (in case of multivalued attributes), or the special value $\bot$ (if the value for an attribute name is missing)

Defined by the semantic function $\mathcal{R} : Request \to R$:

$$\mathcal{R}[\![(n', v')]\!]n = \begin{cases} v' & \text{if } n = n' \\ \bot & \text{otherwise} \end{cases}$$

$$\mathcal{R}[\![(n_i, v_i)^+(n', v')]\!]n = \begin{cases} \mathcal{R}[\![(n_i, v_i)^+]\!]n \Cup v' & \text{if } n = n' \\ \mathcal{R}[\![(n_i, v_i)^+]\!]n & \text{otherwise} \end{cases}$$

The operator $\Cup$ is used to deal with multivalued attributes (we let $V \in 2^{Value}$)

$$v \Cup v' = \{v, v'\} \qquad V \Cup v' = V \cup \{v'\} \qquad \bot \Cup v' = v'$$

FACPL Formal Semantics: expressions

The semantic function for expressions is

$$\mathcal{E} : Expr \rightarrow (R \rightarrow Value \ \cup \ 2^{Value} \cup \{\texttt{error}, \bot\})$$

thus the semantics of an expression is a function that, given a request, returns a literal value, or a set of values, or the special value $\bot$, or an error (e.g., when an argument of an operator has unexpected type)

Some clauses for $\mathcal{E}$ (r is a *semantic* request):

$$\mathcal{E}[\![v]\!]r = v$$

$$\mathcal{E}[\![n]\!]r = r(n)$$

$$\mathcal{E}[\![\texttt{or}(expr_1, expr_2)]\!]r =$$

$$\begin{cases} \texttt{true} & \text{if } \mathcal{E}[\![expr_1]\!]r = \texttt{true} \vee \mathcal{E}[\![expr_2]\!]r = \texttt{true} \\ \texttt{false} & \text{if } \mathcal{E}[\![expr_1]\!]r = \mathcal{E}[\![expr_2]\!]r = \texttt{false} \\ \bot & \text{if } \mathcal{E}[\![expr_i]\!]r = \bot \wedge \mathcal{E}[\![expr_j]\!]r \in \{\texttt{false}, \bot\} \\ \texttt{error} & \text{otherwise} \end{cases}$$

FACPL Formal Semantics: policies

The semantic function for policies is

$$\mathcal{P} : \mathit{Policy} \to (R \to \mathit{PDPReponse})$$

thus the semantics of a policy is a function that, given a request, returns an authorisation decision paired with a (possibly empty) sequence of instantiated obligations

The clause for *rules* is (r is a *semantic* request):

$$\mathcal{P}[\![(e \ \texttt{target:}\ \mathit{expr} \ \ \texttt{obl:}\ o^*)]\!]r = \begin{cases} \langle e \ \ io^* \rangle & \mathit{if}\ \mathcal{E}[\![\mathit{expr}]\!]r = \texttt{true} \ \wedge \ \mathcal{O}[\![o^*]\!]r = io^* \\ \texttt{not-applicable} & \mathit{if}\ \mathcal{E}[\![\mathit{expr}]\!]r = \texttt{false} \ \vee \ \mathcal{E}[\![\mathit{expr}]\!]r = \bot \\ \texttt{indeterminate} & \mathit{otherwise} \end{cases}$$

- Rule effect and *instantiated obligations* are returned when the target evaluates to `true` and all obligations are successfully instantiated
- Otherwise, it could be the case that
  - the rule does not apply to the request
  - an error has occurred while evaluating the target or instantiating the obligations

FACPL Formal Semantics: policies

The clause for *policy sets* is then

$$\mathcal{P}[\![\{a\ \texttt{target:}\ expr\ \ \texttt{policies:}\ \pi^+\ \texttt{obl-p:}\ o_p^*\ \ \texttt{obl-d:}\ o_d^*\ \}]\!]r =$$

$$\begin{cases}
\langle \texttt{permit}\ \ io_1^* \bullet io_2^* \rangle & \text{if } \mathcal{E}[\![expr]\!]r = \texttt{true} \\
& \quad \wedge\, \mathcal{A}[\![a, \pi^+]\!]r = \langle \texttt{permit}\ io_1^* \rangle \\
& \quad \wedge\, \mathcal{O}[\![o_p^*]\!]r = io_2^* \\
\langle \texttt{deny}\ \ io_1^* \bullet io_2^* \rangle & \text{if } \mathcal{E}[\![expr]\!]r = \texttt{true} \\
& \quad \wedge\, \mathcal{A}[\![a, \pi^+]\!]r = \langle \texttt{deny}\ io_1^* \rangle \\
& \quad \wedge\, \mathcal{O}[\![o_d^*]\!]r = io_2^* \\
\texttt{not-applicable} & \text{if } \mathcal{E}[\![expr]\!]r = \texttt{false} \\
& \quad \vee\, \mathcal{E}[\![expr]\!]r = \bot \\
& \quad \vee\, (\mathcal{E}[\![expr]\!]r = \texttt{true} \\
& \qquad \wedge\, \mathcal{A}[\![a, \pi^+]\!]r = \texttt{not-applicable}) \\
\texttt{indeterminate} & \text{otherwise}
\end{cases}$$

FACPL Formal Semantics: binary operators

- The function $\mathcal{A}$ defining the semantics of combining algorithms is defined in terms of a family of binary operators
- Let alg denote the name of a combining algorithm, e.g. `permit-overrides`; the corresponding semantic operator is identified as $\otimes$alg and is defined by means of a two-dimensional matrix that, given two PDP responses, calculates the resulting combined response
- For instance, the combination matrix for the $\otimes$`permit-overrides` operator is

$res_1 \backslash^{res_2}$	$\langle$`permit` $io_2^*\rangle$	$\langle$`deny` $io_2^*\rangle$	`not-applicable`	`indeterminate`
$\langle$`permit` $io_1^*\rangle$	$\langle$`permit` $io_1^* \bullet io_2^*\rangle$	$\langle$`permit` $io_1^*\rangle$	$\langle$`permit` $io_1^*\rangle$	$\langle$`permit` $io_1^*\rangle$
$\langle$`deny` $io_1^*\rangle$	$\langle$`permit` $io_2^*\rangle$	$\langle$`deny` $io_1^* \bullet io_2^*\rangle$	$\langle$`deny` $io_1^*\rangle$	`indeterminate`
`not-applicable`	$\langle$`permit` $io_2^*\rangle$	$\langle$`deny` $io_2^*\rangle$	`not-applicable`	`indeterminate`
`indeterminate`	$\langle$`permit` $io_2^*\rangle$	`indeterminate`	`indeterminate`	`indeterminate`

- Notably, combining algorithms, as well as expression operators and PEP enforcement algorithms, enable forms of error management

FACPL Formal Semantics: combining algorithms

Semantics of combining algorithms

$$\mathcal{A} : Alg \times Policy^{+} \rightarrow (R \rightarrow \ PDPReponse)$$

If the `all` strategy is adopted, the definition clauses are

$$\mathcal{A}[\![\mathsf{alg}_{\texttt{all}}, \pi]\!]r = \otimes\mathsf{alg}(\mathcal{P}[\![\pi]\!]r)$$

$$\mathcal{A}[\![\mathsf{alg}_{\texttt{all}}, \pi^{+}\pi']\!]r = \otimes\mathsf{alg}(\mathcal{A}[\![\mathsf{alg}_{\texttt{all}}, \pi^{+}]\!]r, \mathcal{P}[\![\pi']\!]r)$$

The definition clauses for the `greedy` strategy are

$$\mathcal{A}[\![\mathsf{alg}_{\texttt{greedy}}, \pi]\!]r = \otimes\mathsf{alg}(\mathcal{P}[\![\pi]\!]r)$$

$$\mathcal{A}[\![\mathsf{alg}_{\texttt{greedy}}, \pi^{+}\pi']\!]r =$$

$$\begin{cases} \mathcal{A}[\![\mathsf{alg}_{\texttt{greedy}}, \pi^{+}]\!]r & \textit{if} \quad \textit{isFinal}_{\mathsf{alg}}(\mathcal{A}[\![\mathsf{alg}_{\texttt{greedy}}, \pi^{+}]\!]r) \\ \otimes\mathsf{alg}(\mathcal{A}[\![\mathsf{alg}_{\texttt{greedy}}, \pi^{+}]\!]r, \mathcal{P}[\![\pi']\!]r) & \textit{otherwise} \end{cases}$$

Differently from the `all` strategy, the `greedy` one halts the evaluation of the input policy sequence as soon as a final decision is determined, without necessarily evaluating all the policies in the sequence

Specification of FACPL policies

Specification of the epSOS Access Control System

Starting from the epSOS specifications, we deduced a set of *business requirements* concerning the e-Prescription service

Requirements for the e-Prescription service

The access control system must ensure the following security requirements:

1. Doctors with `e-Pre-Read` and `e-Pre-Write` permissions can write e-Prescriptions
2. Doctors with `e-Pre-Read` permission can read e-Prescriptions
3. Pharmacists with `e-Pre-Read` permission can read e-Prescriptions
4. Authorised user accesses must be recorded by the system
5. Patients must be informed of unauthorised access attempts
6. Data exchanged should be compressed

Specification of the epSOS Access Control System

- Notably, the first three *requirements* deal with access restrictions, while the other ones deal with *additional functionalities* that sophisticated access control systems, like ours, can provide
  - Items 1 - 3: *closed-world requirements* stating the allowed accesses
  - Items 4 - 6: *additional functionalities* required for managing accesses
- We explicitly report all and only those requirements authorising some actions. Hence, every action not explicitly authorised is *forbidden*
- For instance, it is *not allowed* to pharmacists to write e-Prescriptions, which is instead allowed to doctors exhibiting specific permissions

Specification Steps

On the base of the security requirements . . .

- Assume each relevant requester credential is represented by a pre-defined attribute (n, v). E.g.
  - Requester role:
    - ★ n = `subject/role`
    - ★ v $\in$ {"*doctor*", "*pharmacist*"}
  - Requested action:
    - ★ n = `action/id`
    - ★ v $\in$ {"*read*", "*write*"}
- Write basic access rules by defining controls on attributes
- Combine basic access rules into policies
- Possibly combine policies hierarchically

Step1: Writing Rules

- Doctors with e-Pre-Read and e-Pre-Write permissions can write e-Prescriptions

```
Rule write (permit
  target: equal(subject/role, "doctor")
       && equal(action/id, "write")
       && in ("e-Pre-Write", subject/permission)
       && in ("e-Pre-Read", subject/permission))
```

Step1: Writing Rules

- Doctors with e-Pre-Read and e-Pre-Write permissions can write e-Prescriptions

```
Rule write (permit
  target: equal(subject/role, "doctor")
      && equal(action/id, "write")
      && in ("e-Pre-Write", subject/permission)
      && in ("e-Pre-Read", subject/permission))
```

- Doctors with e-Pre-Read permission can read e-Prescriptions

```
Rule read (permit
  target: equal(subject/role, "doctor")
      && equal(action/id, "read")
      && in ("e-Pre-Read", subject/permission))
```

Step1: Writing Rules

- Doctors with e-Pre-Read and e-Pre-Write permissions can write e-Prescriptions

```
Rule write (permit
  target: equal(subject/role, "doctor")
      && equal(action/id, "write")
      && in ("e-Pre-Write", subject/permission)
      && in ("e-Pre-Read", subject/permission))
```

- Doctors with e-Pre-Read permission can read e-Prescriptions

```
Rule read (permit
  target: equal(subject/role, "doctor")
      && equal(action/id, "read")
      && in ("e-Pre-Read", subject/permission))
```

- Pharmacists with e-Pre-Read permission can read e-Prescriptions

```
Rule pha (permit
  target: equal(subject/role, "pharmacist")
     && equal(action/id, "read")
     && in ("e-Pre-Read", subject/permission))
```

Step2: Combining Rules

```
PolicySet ePre { permit-overrides-all
 target: equal("e-Prescription", resource/type)

 policies:

   Rule write (permit   target: ... )
   Rule read (permit   target: ... )
   Rule pha (permit   target: ... )

 obl_p:
   [M log(system/time, resource/type,subject/id, action/id)]
}
```

- The permit-overrides-all algorithm ensures that decision permit takes precedence over the others
- The obligation of the policy *ePre* enforces the Requirement 4, i.e. authorised user accesses must be recorded by the system

Step3: Validating the Policy (1/2)

On the base of the first requirement:

"Doctors with e-Pre-Read and e-Pre-Write permissions can write e-Prescriptions"

we expect that the following request is allowed, i.e. evaluated to permit

```
Request:{ Request1
  (subject/id,"Dr House")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"doctor")
  (subject/permission,"e-Pre-Read","e-Pre-Write")
  (action/id,"write")
}
```

Step3: Validating the Policy (1/2)

On the base of the first requirement:

"Doctors with e-Pre-Read and e-Pre-Write permissions can write e-Prescriptions"

we expect that the following request is allowed, i.e. evaluated to permit

```
Request:{ Request1
  (subject/id,"Dr House")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"doctor")
  (subject/permission,"e-Pre-Read","e-Pre-Write")
  (action/id,"write")
}
```

- the policy target

  ```
  target: equal("e-Prescription", resource/type)
  ```

  evaluates to true, thus the policy is applicable to the request

Step3: Validating the Policy (1/2)

On the base of the first requirement:

"Doctors with e-Pre-Read and e-Pre-Write permissions can write e-Prescriptions"

we expect that the following request is allowed, i.e. evaluated to permit

```
Request:{ Request1
  (subject/id,"Dr House")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"doctor")
  (subject/permission,"e-Pre-Read","e-Pre-Write")
  (action/id,"write")
}
```

- the first rule evaluates to permit

```
Rule write (permit
  target: equal(subject/role, "doctor")
       && equal(action/id, "write")
       && in ("e-Pre-Write", subject/permission)
       && in ("e-Pre-Read", subject/permission))
```

since its target evaluates to true

Step3: Validating the Policy (1/2)

On the base of the first requirement:

"Doctors with e-Pre-Read and e-Pre-Write permissions can write e-Prescriptions"

we expect that the following request is allowed, i.e. evaluated to permit

```
Request:{ Request1
  (subject/id,"Dr House")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"doctor")
  (subject/permission,"e-Pre-Read","e-Pre-Write")
  (action/id,"write")
}
```

- the second rule evaluates to not-applicable

```
Rule read (permit
  target: equal(subject/role, "doctor")
       && equal(action/id, "read")
       && in ("e-Pre-Read", subject/permission))
```

since its target evaluates to false

Step3: Validating the Policy (1/2)

On the base of the first requirement:

"Doctors with e-Pre-Read and e-Pre-Write permissions can write e-Prescriptions"

we expect that the following request is allowed, i.e. evaluated to permit

```
Request:{ Request1
  (subject/id,"Dr House")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"doctor")
  (subject/permission,"e-Pre-Read","e-Pre-Write")
  (action/id,"write")
}
```

- the third rule evaluates to not-applicable

```
Rule pha (permit
  target: equal(subject/role, "pharmacist")
      && equal(action/id, "read")
      && in ("e-Pre-Read", subject/permission))
```

since its target evaluates to false

Step3: Validating the Policy (1/2)

On the base of the first requirement:

"Doctors with e-Pre-Read and e-Pre-Write permissions can write e-Prescriptions"

we expect that the following request is allowed, i.e. evaluated to permit

```
Request:{ Request1
  (subject/id,"Dr House")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"doctor")
  (subject/permission,"e-Pre-Read","e-Pre-Write")
  (action/id,"write")
}
```

- The application of the combining algorithm permit-overrides-all to the decisions permit, not-applicable, not-applicable returns permit ... as expected!

Step3: Validating the Policy (2/2)

Due to the *closed-world* nature of the requirements, the following request, representing a pharmacist willing to write an e-Prescription, should be forbidden, i.e. evaluated to deny

```
Request:{ Request2
  (subject/id,"Dr Alex")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"pharmacist")
  (action/id,"write")
}
```

Step3: Validating the Policy (2/2)

Due to the *closed-world* nature of the requirements, the following request, representing a pharmacist willing to write an e-Prescription, should be forbidden, i.e. evaluated to deny

```
Request:{ Request2
  (subject/id,"Dr Alex")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"pharmacist")
  (action/id,"write")
}
```

- the policy target

  ```
  target: equal("e-Prescription", resource/type)
  ```

  evaluates to true, thus the policy is applicable to the request

Step3: Validating the Policy (2/2)

Due to the *closed-world* nature of the requirements, the following request, representing a pharmacist willing to write an e-Prescription, should be forbidden, i.e. evaluated to deny

```
Request:{ Request2
  (subject/id,"Dr Alex")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"pharmacist")
  (action/id,"write")
}
```

- the first rule evaluates to not-applicable

```
Rule write (permit
  target: equal(subject/role, "doctor")
       && equal(action/id, "write")
       && in ("e-Pre-Write", subject/permission)
       && in ("e-Pre-Read", subject/permission))
```

since its target evaluates to false

Step3: Validating the Policy (2/2)

Due to the *closed-world* nature of the requirements, the following request, representing a pharmacist willing to write an e-Prescription, should be forbidden, i.e. evaluated to deny

```
Request:{ Request2
  (subject/id,"Dr Alex")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"pharmacist")
  (action/id,"write")
}
```

- the second rule evaluates to not-applicable

```
Rule read (permit
  target: equal(subject/role, "doctor")
       && equal(action/id, "read")
       && in ("e-Pre-Read", subject/permission))
```

since its target evaluates to false

Step3: Validating the Policy (2/2)

Due to the *closed-world* nature of the requirements, the following request, representing a pharmacist willing to write an e-Prescription, should be forbidden, i.e. evaluated to deny

```
Request:{ Request2
  (subject/id,"Dr Alex")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"pharmacist")
  (action/id,"write")
}
```

- the third rule evaluates to not-applicable

```
Rule pha (permit
  target: equal(subject/role, "pharmacist")
      && equal(action/id, "read")
      && in ("e-Pre-Read", subject/permission))
```

since its target evaluates to false

Step3: Validating the Policy (2/2)

Due to the *closed-world* nature of the requirements, the following request, representing a pharmacist willing to write an e-Prescription, should be forbidden, i.e. evaluated to deny

```
Request:{ Request2
  (subject/id,"Dr Alex")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"pharmacist")
  (action/id,"write")
}
```

- The application of the combining algorithm permit-overrides-all to the decisions not-applicable, not-applicable, not-applicable returns not-applicable . . . instead of deny!

Step4: Patient-informed consent policy

```
PolicySet Consent {permit-overrides-all
 target: true

 policies:
   PolicySet ePre { ... }
   Rule ruleDeny (deny)

 obl_p:
   [O compress()]
 obl_d:
   [M mail(resource/patient-mail, "Data request by unauthorised
       subject")]
}
```

The policy can be amended by introducing an additional layer comprising

- a target matching any request
- the (previous) policy managing the e-Prescription
- the always applicable rule `ruleDeny`
- two obligations enforcing the Requirements 5 & 6
  (i.e. patients must be informed of unauthorised access attempts and data exchanged should be compressed)

Step5: Alice's patient-informed consent policy

Alice's policy for the management of her electronic health data could be simply obtained by *tailoring* the target of the previous policy to check the patient identifier so that the policy only applies to requests regarding *Alice*

```
PolicySet Consent {permit-overrides-all
 target: true

 policies:
   PolicySet ePre { ... }
   Rule ruleDeny (deny)

 obl_p:
   [O compress()]
 obl_d:
   [M mail(resource/patient-mail, "Data request by unauthorised
       subject")]
}
```

Step5: Alice's patient-informed consent policy

Alice's policy for the management of her electronic health data could be simply obtained by *tailoring* the target of the previous policy to check the patient identifier so that the policy only applies to requests regarding *Alice*

```
PolicySet AliceConsent {permit-overrides-all
 target: equal("Alice",resource/patient-id)

 policies:
   PolicySet ePre { ... }
   Rule ruleDeny (deny)

 obl_p:
   [O compress()]
 obl_d:
   [M mail(resource/patient-mail, "Data request by unauthorised
       subject")]
}
```

Step5: Alice's patient-informed consent policy

Alice's policy for the management of her electronic health data could be simply obtained by *tailoring* the target of the previous policy to check the patient identifier so that the policy only applies to requests regarding *Alice*

- In this way, *Alice* grants access to her e-Prescription data to the healthcare professionals that satisfy the requirements expressed in her consent policy
- Another patient expressing a more restrictive consent, in which for example writing of e-Prescriptions is disabled, will have a similar policy set where the rule modelling the first Requirement is not included

Step5: Alice's patient-informed consent policy

Alice's policy for the management of her electronic health data could be simply obtained by *tailoring* the target of the previous policy to check the patient identifier so that the policy only applies to requests regarding *Alice*

In a more general perspective, the PDP could have a policy set for each patient, that encloses the policies expressing the consent explicitly signed by the patient

- This is the approach followed, e.g., in the Austrian e-Health platform

Analysis of FACPL policies

Reasonability properties

Proposed to precisely characterise the expressiveness of a policy language

- From the use of combination matrices, it follows that FACPL enjoys

Independent composition of policies

The results of application of combining algorithms depend only on the results of the policies given in input

- Like XACML and other policy languages featuring deny rules and comb. alg. similar to those we have presented, FACPL does not enjoy

Safety

A granted request remains granted also if it is extended with new attributes

Monotonicity

The introduction of a new policy in a combination of policies does not change a `permit` decision to a different one

Motivations for the analisys

- It could be challenging to *identify unexpected authorisations* and to determine whether policy fixes *affect decisions* that should not be altered
- The combination of a large number of complex policies is indeed an *error-prone task* that has to be supported with effective analysis techniques
- *Difficulties* to tackle regard:
  - hierarchical structure of policies
  - presence of conflict resolution strategies
  - intricacies deriving from the many involved controls (e.g. *missing* and *erroneous* attributes)
- To develop analysis techniques we exploit the FACPL formal semantics and define a *constraint-based analysis* providing effective supporting techniques for the verification of properties on policies

Analysis Objectives

Support policy developers in the validation of FACPL policies, thus to *statically* identify unexpected authorisations that may occur at run-time

Supported Properties:

- Authorisation Properties

  conditions on the expected authorisations of single requests and, possibly, their extensions

- Structural Properties

  conditions on the relationships among policies on the base of the whole set of authorisations they establish

Authorisation Properties

Conditions on the expected authorisations of single requests and, possibly, their extensions

Since FACPL does not enjoy the safety property, extending a request with additional attributes might change the authorisation of the request in a possibly unexpected way

Authorisation Properties: the role of additional attributes

Let us consider the case of a pharmacist willing to perform an action

```
Request:{ Request3
  (subject/id,"Dr Alex")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"pharmacist")
  (subject/permission,"e-Pre-Read")
}
```

The attribute with name `action/id` is missing, hence the original consent policy evaluates to `not-applicable`

Authorisation Properties: the role of additional attributes

Let us consider the case of a pharmacist willing to perform an action

```
Request:{ Request3
  (subject/id,"Dr Alex")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"pharmacist")
  (subject/permission,"e-Pre-Read")
}
```

The attribute with name `action/id` is missing, hence the original consent policy evaluates to `not-applicable`

If the request is extended with the attribute above, we could have

- (`action/id`, "*read*"): the original consent policy evaluates to `permit`
- (`action/id`, "*write*"): the original consent policy evaluates to `deny`

Authorisation Properties: the role of additional attributes

Let us consider the case of a pharmacist willing to perform an action

```
Request:{ Request3
  (subject/id,"Dr Alex")
  (resource/patient-id,"Alice")
  (resource/type,"e-Prescription")
  (subject/role,"pharmacist")
  (subject/permission,"e-Pre-Read")
}
```

The attribute with name `action/id` is missing, hence the original consent policy evaluates to `not-applicable`

If the request is extended with the attribute above, we could have

- (`action/id`, "*read*"): the original consent policy evaluates to `permit`
- (`action/id`, "*write*"): the original consent policy evaluates to `deny`

Thus, it is important for system designers to consider the authorisation decisions not only of specific requests, but also of their extensions since, e.g., a malicious user could try to exploit them to circumvent the access control system

Authorisation Properties

Conditions on the expected authorisations of single requests and, possibly, their extensions

Request extension set of a given request r

$$Ext(r) \triangleq \{r' \in R \mid r(n) \neq \bot \quad \Rightarrow \quad r'(n) = r(n)\}$$

where

$$R \triangleq Name \rightarrow (Value \cup 2^{Value} \cup \{\bot\})$$

- *Eval*: checks if a request is authorised to a certain authorisation
- *May*: checks if *any* of the extensions of a request is authorised to a certain authorisation
- *Must*: checks if *all* the extensions of a request are authorised to a certain authorisation

Authorisation Properties: the patient consent policies

Verify whether the patient consent policies disallow the access to a pharmacist who wants to write an e-Prescription

First, we can check that the request evaluates to `deny`

(`sub/role`, "`pharmacist`")(`act/id`, "`write`")(`res/type`, "`e-Prescr.`")
Eval `deny`

Second, by exploiting request extensions, we can check if there exists a request by a pharmacist acting on e-Prescriptions which can be evaluated to `not-applicable`

(`sub/role`, "`pharmacist`")(`res/type`, "`e-Prescr.`")
May `not-applicable`

Authorisation Properties: the patient consent policies

(sub/role, "pharmacist")(act/id, "write")(res/type, "e-Prescr.")
Eval deny

(sub/role, "pharmacist")(res/type, "e-Prescr.")
May not-applicable

The verification of these properties with respect to the first policy we have seen results in unsat (*unsatisfiable*) and sat (*satisfiable*), respectively

Indeed, each request by a pharmacist assigning to act/id a value different from read evaluates to not-applicable, hence the first property is not satisfied while the second one holds

Authorisation Properties: the patient consent policies

(sub/role, "pharmacist")(act/id, "write")(res/type, "e-Prescr.")
Eval deny

(sub/role, "pharmacist")(res/type, "e-Prescr.")
May not-applicable

On the contrary, the verification with respect to the amended policy results in sat and unsat, respectively

Both results are due to the internal policy (deny) which, together with the algorithm permit-overrides, prevents not-applicable to be returned and establishes deny as a default decision

Structural Properties

Conditions on the relationships among policies on the base of the whole set of authorisations they establish

- *Completeness*: checks if there is no access request for which there is an absence of decision (i.e. `not-applicable` is never returned)
- *Coverage*: checks if the set of authorisations enforced by a policy is covered by that of another policy (i.e. when the former returns `permit` or `deny`, the latter does the same)
- *Disjointness*: checks if two or more policies enforce disjoint sets of authorisations (i.e. there is no request for which both return any of `permit` or `deny`)

Support system designers in developing and maintaining policies:
for instance, they enable change-impact analysis, which examines policy modifications for discovering unintended consequences of such changes

Structural Properties: the patient consent policies

Specifically, by verifying completeness, we can check if there is a request that evaluates to `not-applicable`

As expected, the original patient consent policy does not satisfy completeness, because there is at least one request that evaluates to `not-applicable`, whereas the amended policy is complete

Instead, we can check if the amended policy correctly refines the original policy by simply verifying coverage

This is true and follows from the fact that amended policy evaluates to `permit` the same set of requests as the original one and that the latter never returns `deny`; clearly, the opposite coverage property does not hold

The two policies are not disjoint as they share the same set of permitted requests

Towards automated verification

- Property verification requires extensive checks on large (possibly infinite) amounts of requests, hence, in order to be practically effective, tool support is essential
- For enabling the analysis of FACPL policies through well-established and efficient software tools,
  - we first introduce a constraint formalism and use it to uniformly represent FACPL policies and the properties to be verified
  - we then translate these constraints into the SMT-LIB language (a standardised constraint language accepted by most SMT solvers)
- Satisfiability Modulo Theory (SMT)
  - First-order formulae containing operations from various theories
  - Main theories used: *Record, Linear Arithmetic, Uninterpreted Functions, Array*
  - SMT solvers are "extensions" of SAT solvers

A constraint formalism for FACPL policies

- A constraint is a relation defined through some conditions on a set of attribute names
  - An assignment of values to attribute names satisfies a constraint if all constraint conditions are matched
- Each policy is represented by a *4-tuple of mutually exclusive constraints*, one for each possible decision
- Constraint tuples representing policies are combined according to the semantics of the combining algorithms
  - Policy hierarchies are thus *flattened* according to the (binary operator) semantics of combing algorithms
- Obligations are ignored since constraint-based analysis is static and cannot thus take into account the result of obligation discharge

Representing FACPL Policies

The first epSOS rule

```
Rule write (permit
  target: equal(subject/role, "doctor")
      && equal(action/id, "write")
      && in ("e-Pre-Write", subject/permission)
      && in ("e-Pre-Read", subject/permission))
```

corresponds to the following tuple of constraints

$$\langle\ \texttt{permit} : \chi_{trg1}$$
$$\texttt{deny} : \texttt{false}$$
$$\texttt{not-applicable} : \neg\chi_{trg1}$$
$$\texttt{indeterminate} : \neg\texttt{isBool}(\chi_{trg1}) \wedge \neg\texttt{isMiss}(\chi_{trg1})\ \rangle$$

Representing FACPL Policies

The first epSOS rule

```
Rule write (permit
  target: equal(subject/role, "doctor")
      && equal(action/id, "write")
      && in ("e-Pre-Write", subject/permission)
      && in ("e-Pre-Read", subject/permission))
```

corresponds to the following tuple of constraints

$$\langle\ \texttt{permit} : \chi_{trg1}$$
$$\texttt{deny} : \texttt{false}$$
$$\texttt{not-applicable} : \neg\chi_{trg1}$$
$$\texttt{indeterminate} : \neg\texttt{isBool}(\chi_{trg1}) \wedge \neg\texttt{isMiss}(\chi_{trg1})\ \rangle$$

where the constraint χ_{trg1} is

$$\begin{aligned}\chi_{trg1} \triangleq\ & \texttt{subject/role} = \text{“doctor”} \\ & \dot{\wedge}\ \texttt{action/id} = \text{“write”} \\ & \dot{\wedge}\ \text{“e-Pre-Write”} \in \texttt{sub/permission} \\ & \dot{\wedge}\ \text{“e-Pre-Read”} \in \texttt{sub/permission}\end{aligned}$$

Representing FACPL Policies

The first epSOS rule

```
Rule write (permit
  target: equal(subject/role, "doctor")
      && equal(action/id, "write")
      && in ("e-Pre-Write", subject/permission)
      && in ("e-Pre-Read", subject/permission))
```

corresponds to the following tuple of constraints

$$\langle\ \texttt{permit}: \chi_{trg1}$$
$$\texttt{deny}: \texttt{false}$$
$$\texttt{not-applicable}: \neg\chi_{trg1}$$
$$\texttt{indeterminate}: \neg\texttt{isBool}(\chi_{trg1}) \wedge \neg\texttt{isMiss}(\chi_{trg1})\ \rangle$$

where the semantic clause for *rules* is (r is a *semantic* request):

$$\mathcal{P}[\![(e\ \ \texttt{target:}\ expr\ \ \texttt{obl:}\ o^*\ )]\!]r = \begin{cases} \langle e\ \ io^*\rangle & \texttt{if}\ \mathcal{E}[\![expr]\!]r = \texttt{true} \wedge \mathcal{O}[\![o^*]\!]r = io^* \\ \texttt{not-applicable} & \texttt{if}\ \mathcal{E}[\![expr]\!]r = \texttt{false} \vee \mathcal{E}[\![expr]\!]r = \bot \\ \texttt{indeterminate} & \texttt{otherwise} \end{cases}$$

Semantic Correspondence and Property Verification

Key result: *the semantics of the constraint-based representation of a policy and the semantics of the policy itself do agree*

Policy Semantic Correspondence

For all $\pi \in \mathit{Policy}$ and for all $r \in R$, it holds that

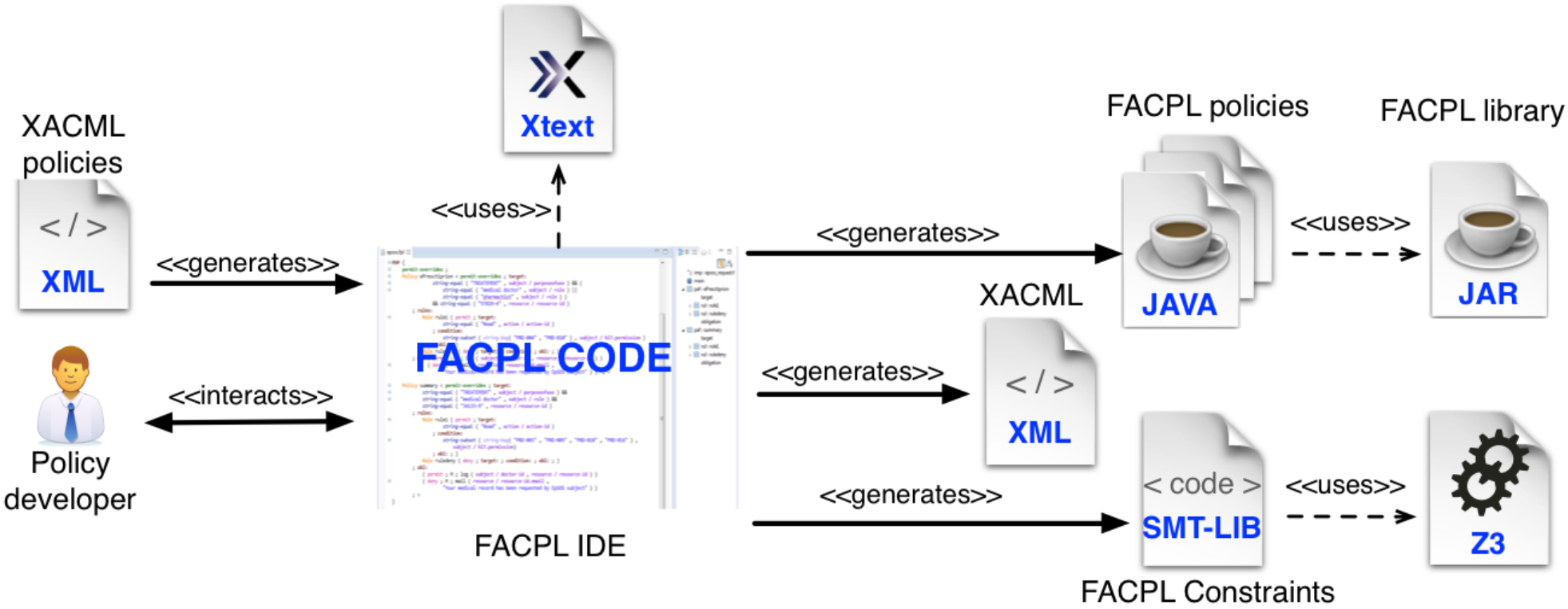
$$\mathcal{P}[\![\pi]\!]r = \langle \mathit{dec}\ \mathit{io}^* \rangle \quad \Leftrightarrow \quad \mathcal{C}[\![\mathcal{T}_P\{\!|\pi|\!\} \downarrow_{\mathit{dec}}]\!]r = \texttt{true}$$

Property Verification

- FACPL policies are automatically translated into *SMT-LIB*, i.e. a constraint language widely accepted by SMT solvers
- The SMT solver Z3 is exploited to verify properties, i.e. to check if an SMT-LIB code is satisfiable or, when it is the case, valid

FACPL *supporting tools*

The FACPL ToolChain



- Eclipse Integrated Development Environment (Xtext-based Eclipse plug-in)
  - Web Application for experimenting FACPL directly online
- Java library for policy Design and Evaluation
- Integration with Z3 via SMT-LIB code
- Partial interoperability with XACML

The FACPL IDE

- Supporting features for writing FACPL policies
  (code suggestion and completion, cross-references, highlighting of code, etc.)
- Evaluation of FACPL policies by using the dedicated Java library
- Automatic generation of SMT-LIB, XACML and Java code

An excerpt of the Java code of the e-Prescription policy

```
public class PolicySet_e-Prescription extends PolicySet{
  public PolicySet_e-Prescription(){
    addCombiningAlg(PermitOverrides.class);
    addTarget(new ExpressionFunction(Equal.class, "e-Prescription",
      new AttributeName("resource","type")));
    addRule(new rule1());
    addRule(new rule2());
    addRule(new rule3());
    addObligation(new Obligation("log",Effect.PERMIT,ObligationType
        .M,
      new AttributeName("system","time"),new AttributeName("
          resource","type"),
      new AttributeName("subject","id"),new AttributeName("action"
          ,"id")));}
  private class rule1 extends Rule{
    rule1 (){
     addEffect(Effect.PERMIT);
     addTarget(...new ExpressionFunction(In.class,
    new AttributeName("subject","permission"),"e-Pre-Write"),...)
        ;}}
  private class rule2 extends Rule{ rule2 (){...} }
  private class rule3 extends Rule{ rule3 (){...} }}
```

Cloud IaaS scenario

Cloud IaaS scenario

- A IaaS (Infrastructure as a Service) provider offers customers a range of pre-configured VMs
- Each type of VM uses specific amounts of dedicated computing resources and features a Service Level Agreement (SLA) that the provider commits to guarantee
- The provider offers strongly defined types of VMs, like most of popular IaaS providers

Cloud IaaS scenario

- A IaaS (Infrastructure as a Service) provider offers customers a range of pre-configured VMs
- Each type of VM uses specific amounts of dedicated computing resources and features a Service Level Agreement (SLA) that the provider commits to guarantee
- The provider offers strongly defined types of VMs, like most of popular IaaS providers

E.g. Amazon Elastic Compute Cloud (Amazon EC2) è un servizio Web che fornisce capacità di elaborazione sicura e scalabile nel cloud

Instance Type	vCPU	Memory (GiB)	Storage (GB)	Networking Performance	Physical Processor	Clock Speed (GHz)
t3.nano	2	0.5	EBS-only	Low	Intel Scalable Proc.	2.5
t3.micro	2	1	EBS-only	Low-Moderate	Intel Scalable Proc.	2.5
t3.small	2	2	EBS-only	Low-Moderate	Intel Scalable Proc.	2.5
...	...	...	...	...	...	...
t3.2xlarge	8	32	EBS-only	Low-Moderate	Intel Scalable Proc.	2.5
...	...	...	...	...	...	...

Updates at https://aws.amazon.com/it/ec2/instance-types/

Cloud IaaS scenario

- A IaaS (Infrastructure as a Service) provider offers customers a range of pre-configured VMs
- Each type of VM uses specific amounts of dedicated computing resources and features a Service Level Agreement (SLA) that the provider commits to guarantee
- The provider offers strongly defined types of VMs, like most of popular IaaS providers
- The provider's service portfolio contains two VMs:
  - `TYPE_1` requires the allocation of **one** unit of resources
  - `TYPE_2` requires the allocation of **two** units of resources

For simplicity sake, we use an aggregated measure to describe the HW resources needed to instantiate the VMs (CPU, memory, storage, ...)

Cloud IaaS scenario

- A IaaS (Infrastructure as a Service) provider offers customers a range of pre-configured VMs
- Each type of VM uses specific amounts of dedicated computing resources and features a Service Level Agreement (SLA) that the provider commits to guarantee
- The provider offers strongly defined types of VMs, like most of popular IaaS providers
- The provider's service portfolio contains two VMs:
  - `TYPE_1` requires the allocation of **one** unit of resources
  - `TYPE_2` requires the allocation of **two** units of resources

For simplicity sake, we use an aggregated measure to describe the HW resources needed to instantiate the VMs (CPU, memory, storage, . . .)

Key aspect → Resource Usage

Allocation of the proper amount of resources needed to instantiate new VMs

SLA guarantees

`TYPE_1` and `TYPE_2` have different SLA guarantees if the system is highly loaded

SLA guarantees

`TYPE_1` and `TYPE_2` have different SLA guarantees if the system is highly loaded

IF the system cannot allocate a new `TYPE_2` VM

THEN an appropriate number of `TYPE_1` VMs already instantiated will be **frozen** and moved to a *queue* of suspended VMs

- The *queue* is periodically checked for reactivating suspended VMs
- Owners of suspended VMs receive a credit

SLA guarantees

`TYPE_1` and `TYPE_2` have different SLA guarantees if the system is highly loaded

IF the system cannot allocate a new `TYPE_2` VM

THEN an appropriate number of `TYPE_1` VMs already instantiated will be **frozen** and moved to a *queue* of suspended VMs

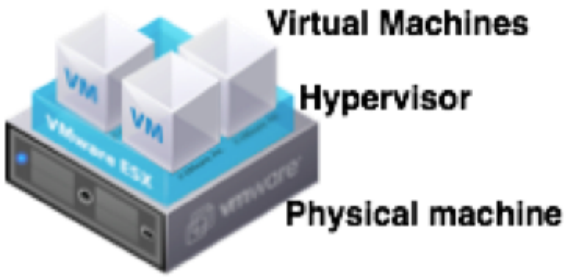
- The *queue* is periodically checked for reactivating suspended VMs
- Owners of suspended VMs receive a credit

Key Aspect $\rightarrow$ **Adaptation**

Freeze the proper amount of `TYPE_1` VMs to re-configure the system and then instantiate a `TYPE_2` VM

Virtualisation

Virtualisation is accomplished using *hypervisors*

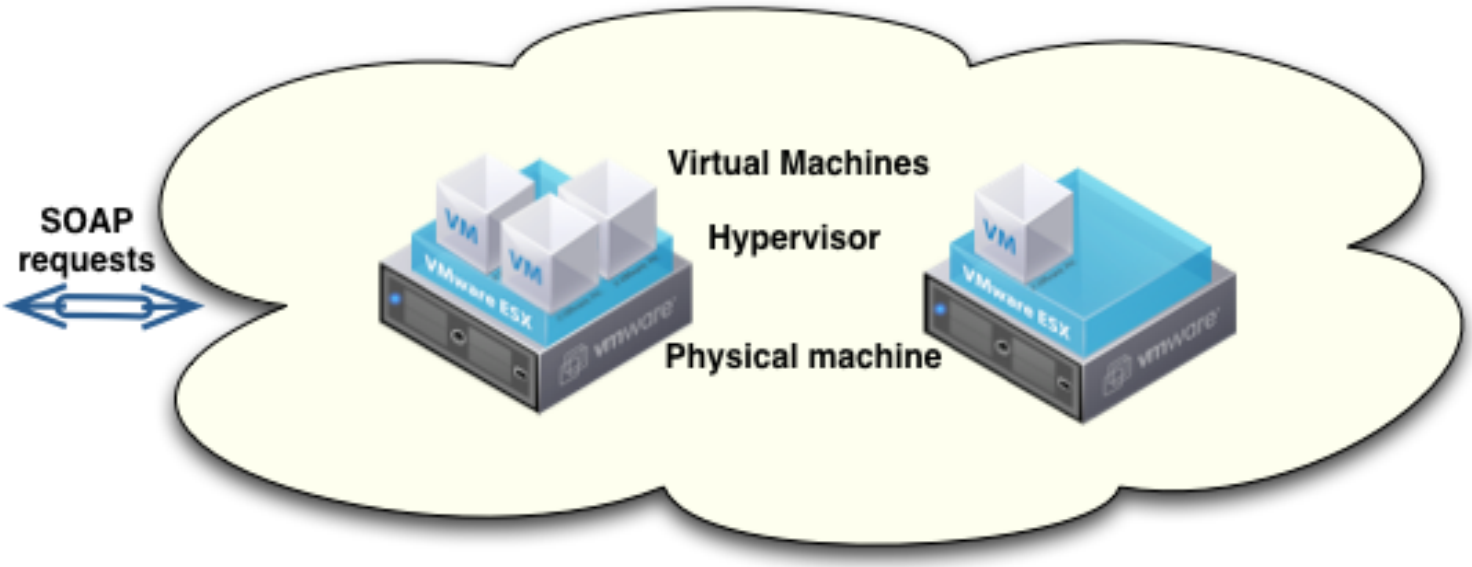


...

Virtualisation

Virtualisation is accomplished using *hypervisors*

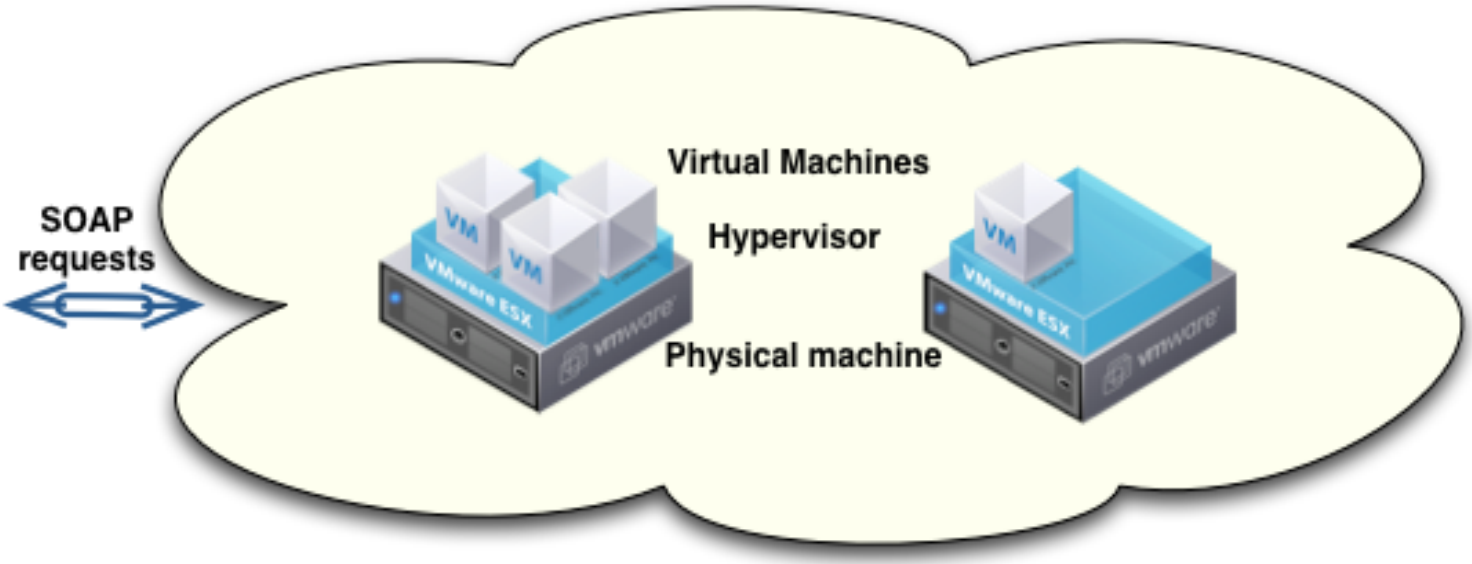
- The provider relies on two hypervisors running on top of two physical machines
- The IaaS platform has a remote access by SOAP messages



Virtualisation

Virtualisation is accomplished using *hypervisors*

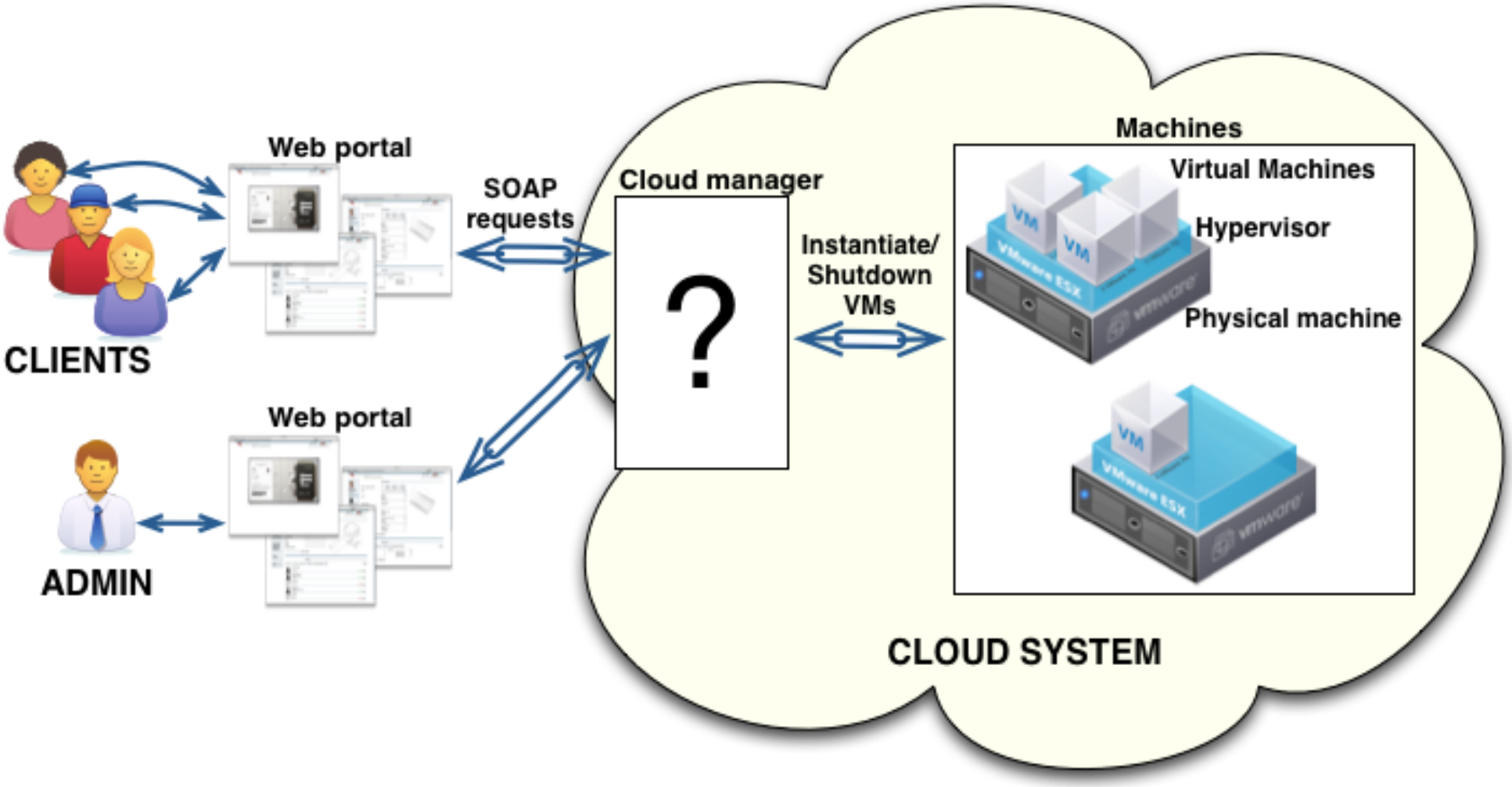
- The provider relies on two hypervisors running on top of two physical machines
- The IaaS platform has a remote access by SOAP messages



Key Aspect → **Access Control**

Allow a client to access the Cloud platform services only if the client's *profile* has the needed credentials

Overall Architecture and data-flow



Role of the Cloud Manager

The **cloud manager** evaluates the received requests wrt a set of policies defining the logic of the system:

- access control policies specify the credentials the clients have to provide in order to access the service
- resource-usage policies specify the resources allocation strategy
- adaptation policies specify the system re-configuration actions in case of high load

Role of the Cloud Manager

The **cloud manager** evaluates the received requests wrt a set of policies defining the logic of the system:

- access control policies specify the credentials the clients have to provide in order to access the service
- resource-usage policies specify the resources allocation strategy
- adaptation policies specify the system re-configuration actions in case of high load

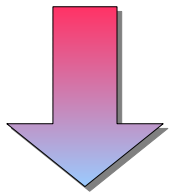
How can we implement the **Cloud manager**?

Role of the Cloud Manager

The **cloud manager** evaluates the received requests wrt a set of policies defining the logic of the system:

- access control policies specify the credentials the clients have to provide in order to access the service
- resource-usage policies specify the resources allocation strategy
- adaptation policies specify the system re-configuration actions in case of high load

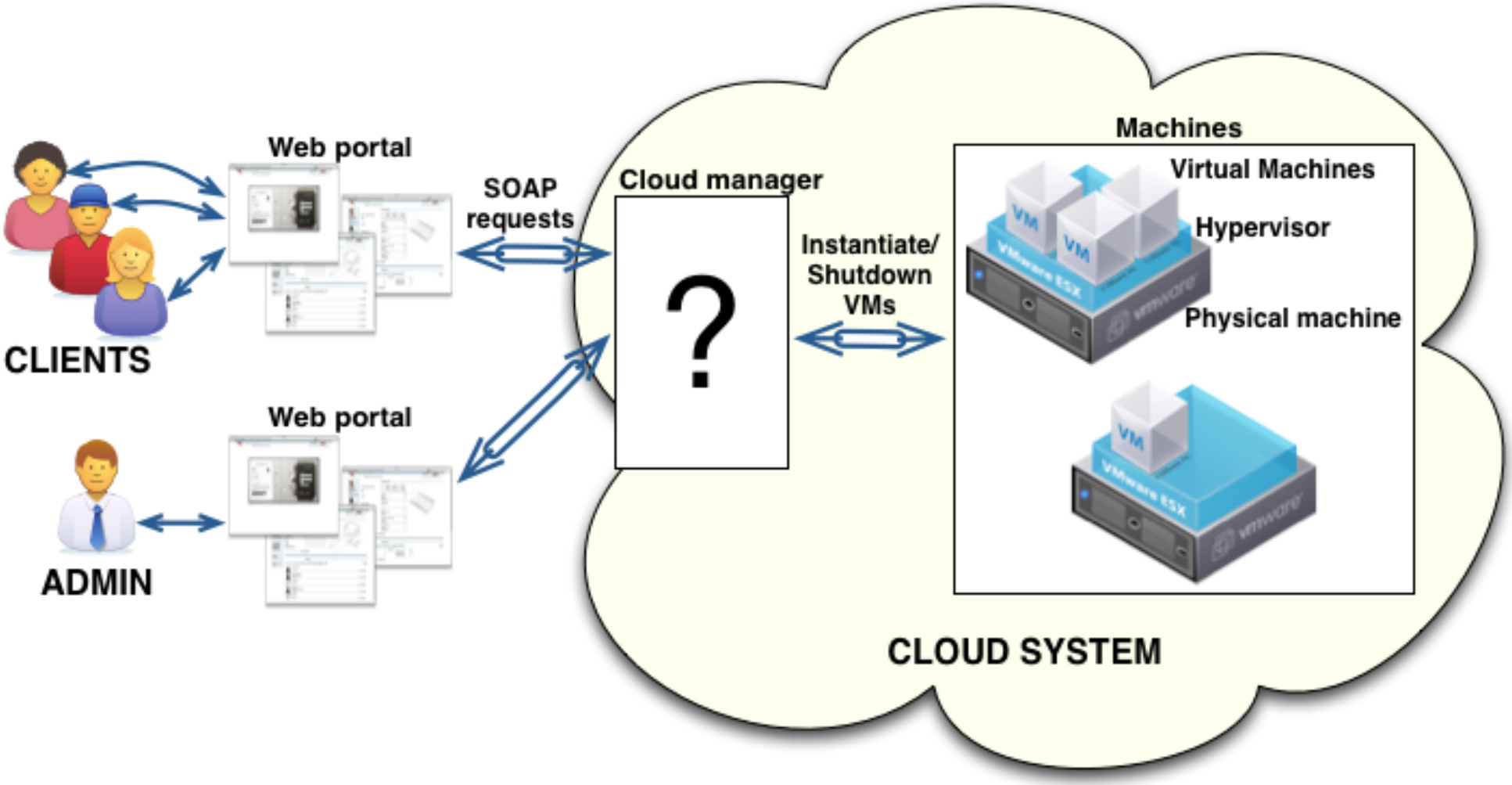
How can we implement the **Cloud manager**?



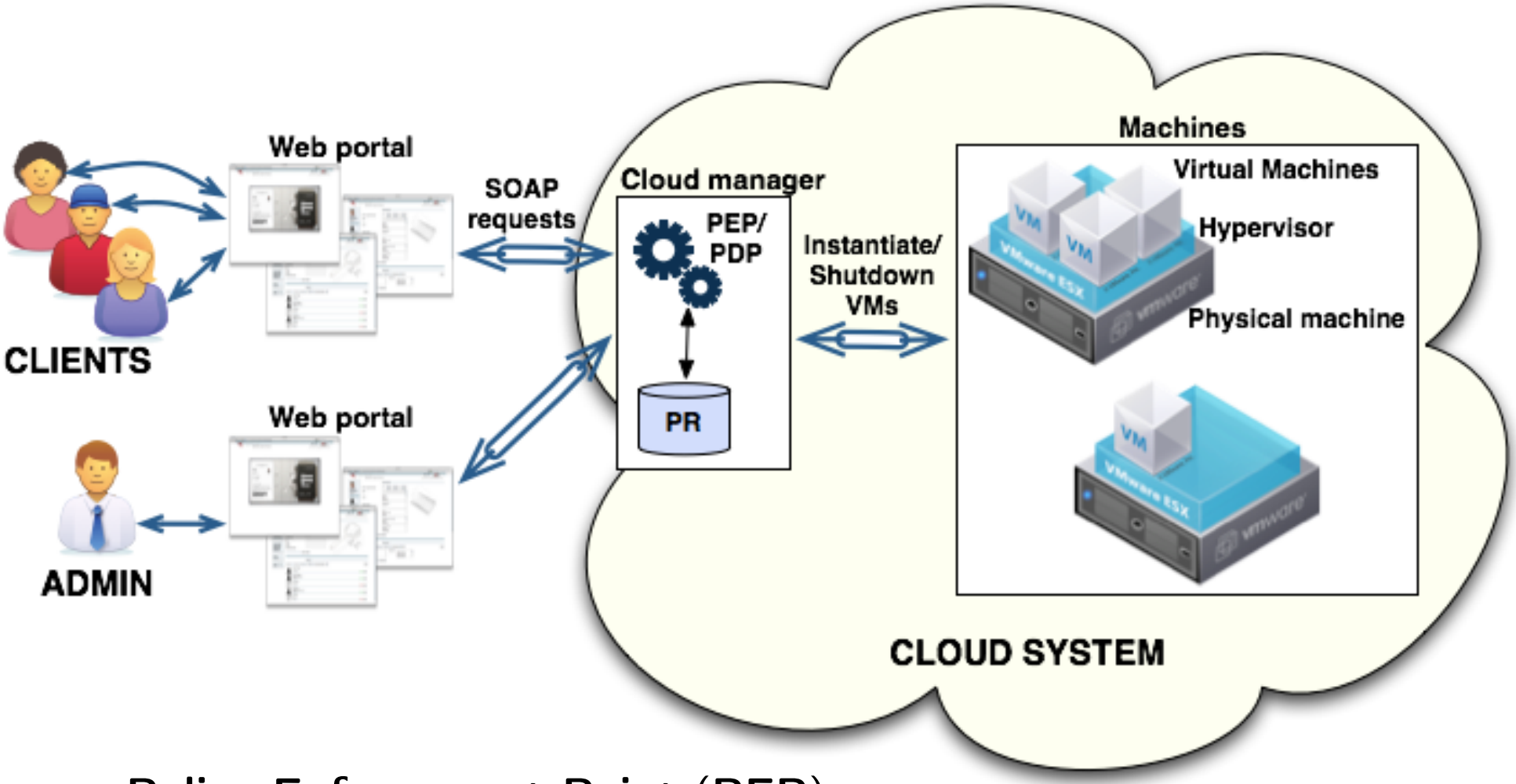
We use FACPL

FACPL at work on the scenario: policies

Policy-based Cloud manager

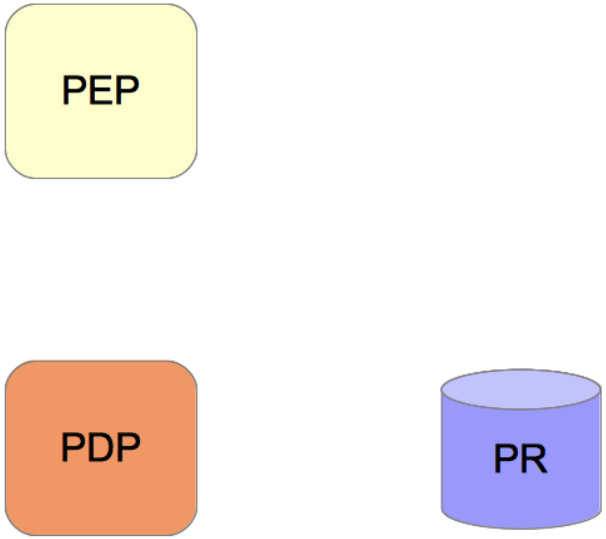


Policy-based Cloud manager

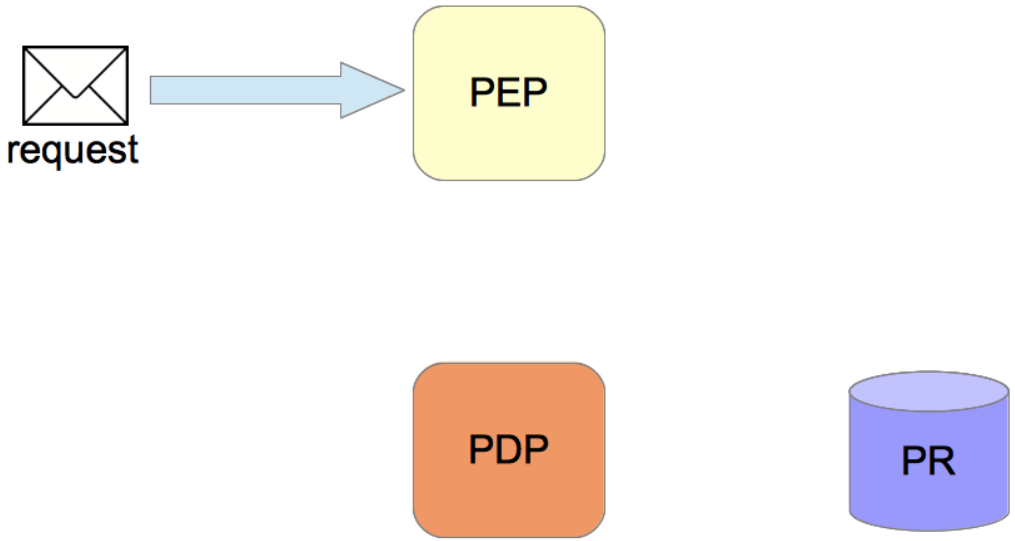


- Policy Enforcement Point (PEP)
- Policy Repository (PR)
- Policy Decision Point (PDP)

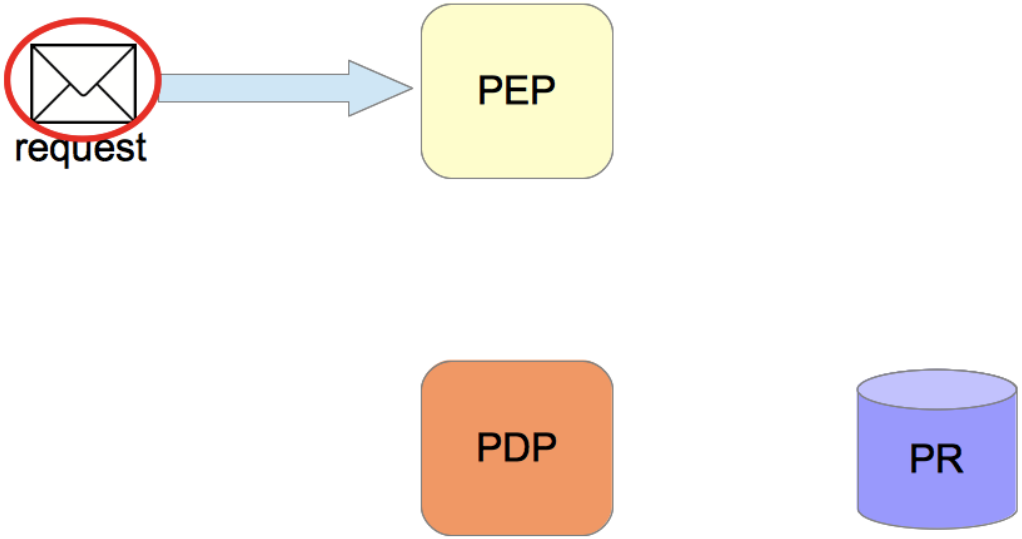
Policy evaluation workflow



Policy evaluation workflow



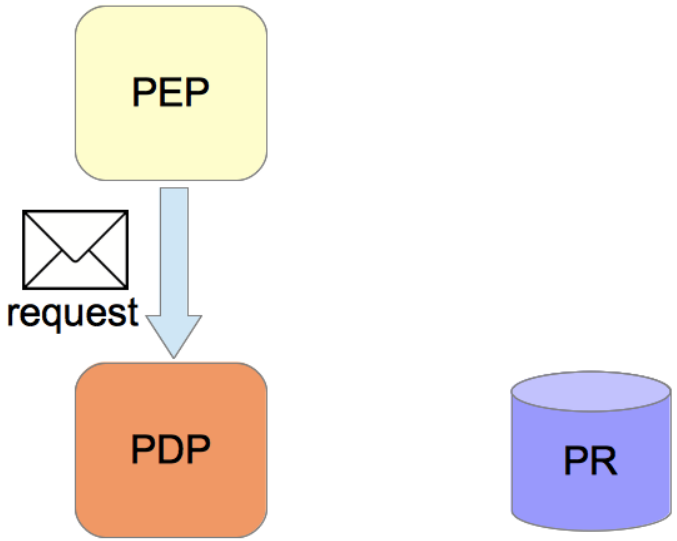
Policy evaluation workflow



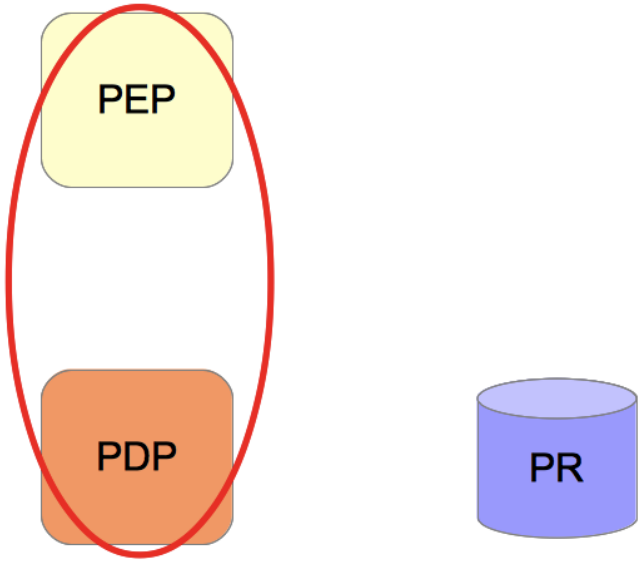
Listing 1: Client Request for creating a new TYPE_2 VM

```
Request:{ Create_Type2_VM
  (subject/profile-id, "P_2")
  (resource/vm-type, "TYPE_2")
  (action/action-id, "CREATE")
}
```

Policy evaluation workflow



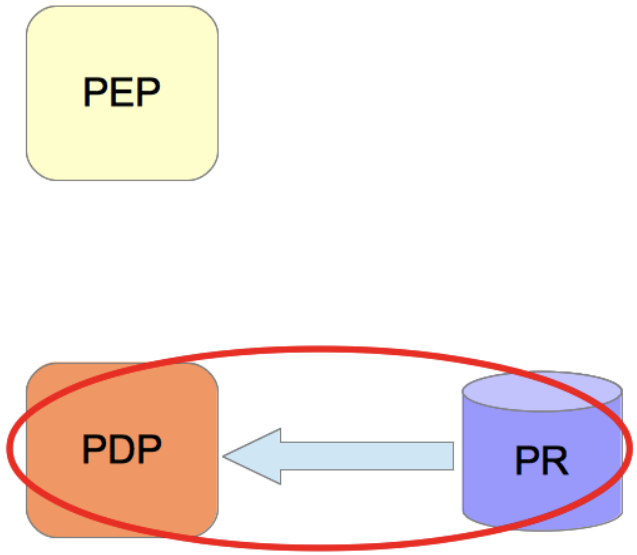
Policy evaluation workflow



Listing 2: FACPL specification of PEP and PDP

```
{
 pep: deny-biased;
 pdp: permit-overrides
      include Create_Policies
      include Release_Policies
}
```

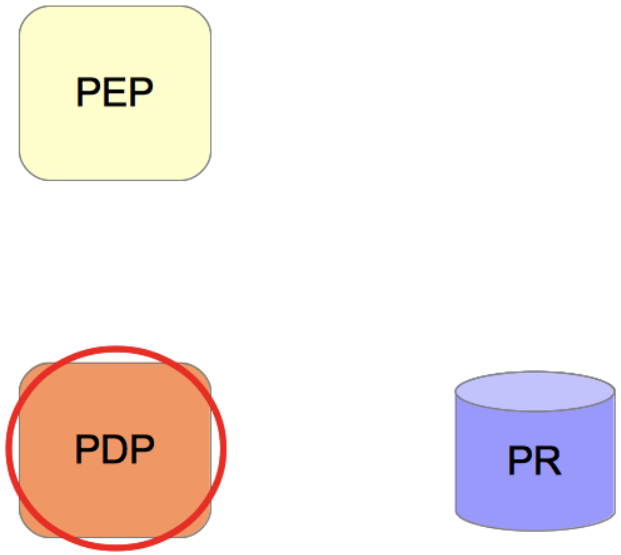
Policy evaluation workflow



Listing 3: Stored Policies

```
{
 pep: deny-biased;
 pdp: permit-overrides
      include Create_Policies
      include Release_Policies
}
```

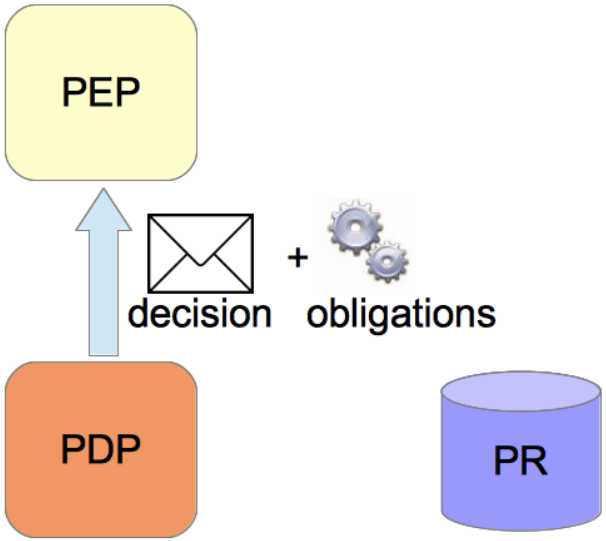
Policy evaluation workflow



PDP performs the evaluation of the policy set consisting of

- the policies `Create_Policies` and `Release_Policies`
- the combining algorithm `permit-overrides`

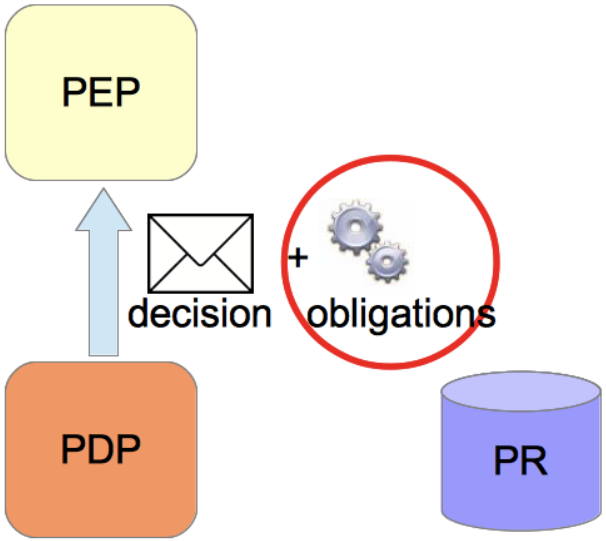
Policy evaluation workflow



PDP returns

- a decision value `permit/deny/not-applicable/indeterminate`
- possibly, some obligations (i.e., actions) to be performed for enforcing the decision, e.g. creation, freezing and releasing of VMs

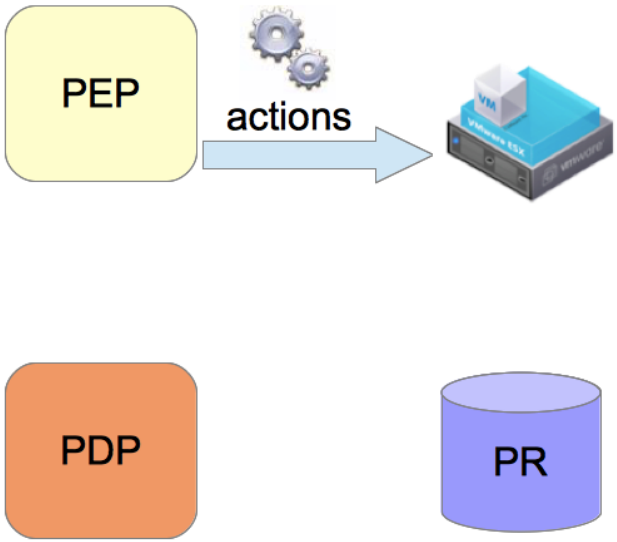
Policy evaluation workflow



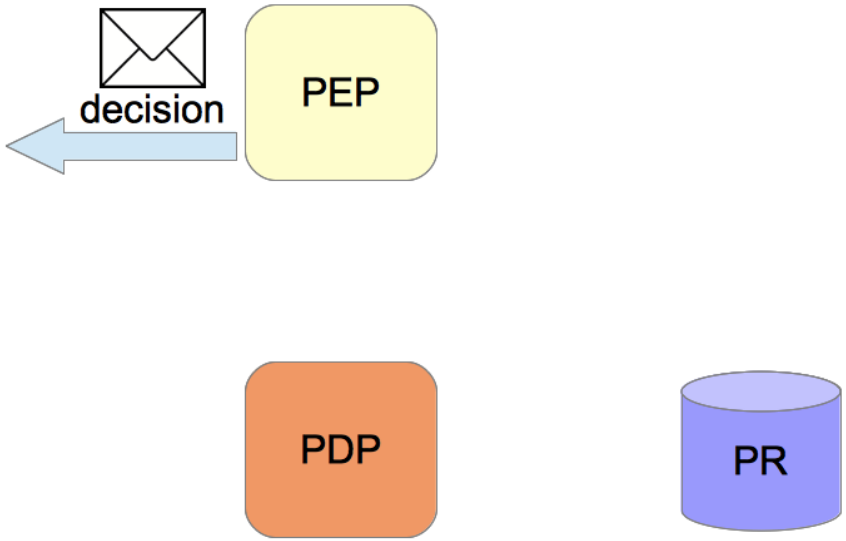
Listing 4: Obligation for creating a new TYPE_2 VM

```
[ M create("HYPER_1", "67cf8383", "TYPE_2") ]
```

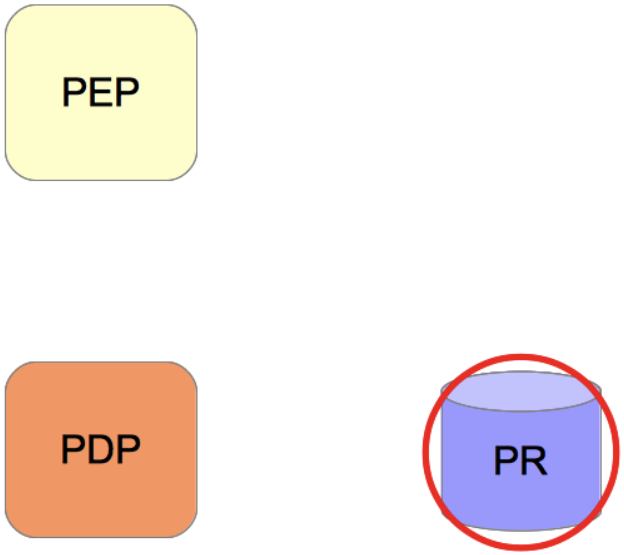
Policy evaluation workflow



Policy evaluation workflow



Policy evaluation workflow



Let us have a glimpse at the Cloud system policies

FACPL specification of PEP and PDP

```
{
 pep: deny-biased;
 pdp: permit-overrides
      include Create_Policies
      include Release_Policies
}
```

FACPL specification of PEP and PDP

```
{
 pep: deny-biased;
 pdp: permit-overrides
      include Create_Policies
      include Release_Policies
}
```

Listing 5: Create Policy

```
PolicySet Create_Policies { permit-overrides
   target:
     equal("CREATE",action/action-id)
   policies:
     Policy SLA_Type1 < deny-unless-permit
       target: ...
       rules: ...
       obl_*: ...
     >
     Policy SLA_Type2 < deny-unless-permit
       target: ...
       rules: ...
       obl_*: ...
     >
}
```

FACPL specification of PEP and PDP

```
{
 pep: deny-biased;
 pdp: permit-overrides
      include Create_Policies
      include Release_Policies
}
```

Listing 6: Policy for creation of a new TYPE_1 VM

```
PolicySet Create_Policies { permit-overrides
   target:
     equal("CREATE",action/action-id)
   policies:
     Policy SLA_Type1 < deny-unless-permit
       target: ...
       rules: ...
       obl_*: ...
     >
     Policy SLA_Type2 < deny-unless-permit
       target: ...
       rules: ...
       obl_*: ...
     >
}
```

Policy for creation of a new TYPE_1 VM

Energy saving strategy

The workload is concentrated on hypervisor HYPER_1 while hypervisor HYPER_2 is only used when the primary one is fully loaded

```
Policy SLA_Type1 < deny-unless-permit
 target:
 (equal("P_1",subject/profile-id)||equal("P_2",subject/profile-id))
 && equal("TYPE_1", resource/vm-type)

 rules:
  Rule hyper_1 (permit
    target:
      less-than-or-equal(1, system/hyper1.availableResources)
    obl_p:
      [M create("HYPER_1", system/vm-id, "TYPE_1")]
  )
  Rule hyper_2 ( ... )

 obl_d:
  [O warning("Not enough available resources for TYPE_1 VMs")]
>
```

Policy for creation of a new TYPE_2 VM

Policy SLA_Type2 is similar, but also contains rules to freeze TYPE_1 VMs when there are not enough available resources for TYPE_2 VMs

```
...
Rule hyper_1_freeze (permit
  target:
    ( equal(0, system/hyper1.availableResources),
      & & less-than-or-equal(2, system/hyper1.vm1-counter)),
    ||
    ( equal(1, system/hyper1.availableResources),
      & & less-than-or-equal(1, system/hyper1.vm1-counter))
  obl_p:
    [M freeze("HYPER_1",
                    subtract(2,system/hyper1.availableResources),
                    "TYPE_1")]
    [M create("HYPER_1", system/vm-id, "TYPE_2")]
)
...
```

A different Resource Usage Strategy

Load Balancing strategy

If the load of the current hypervisor is less than the load of the other one the VMs is assigned on it, otherwise it's chosen the other hypervisor

Policy implementation

A condition is added (in conjunction) to the target of each hypervisor' rule of `Create_Policies`

```
Rule hyper_1 (permit
  target:
    less-than-or-equal(1, system/hyper1.availableResources)
    && less-than-or-equal(system/hyper2.availableResources,
                          system/hyper1.availableResources)
  obl_p:
    [M create("HYPER_1", system/vm-id, "TYPE_1")]
)
```

FACPL at work on the scenario: implementation

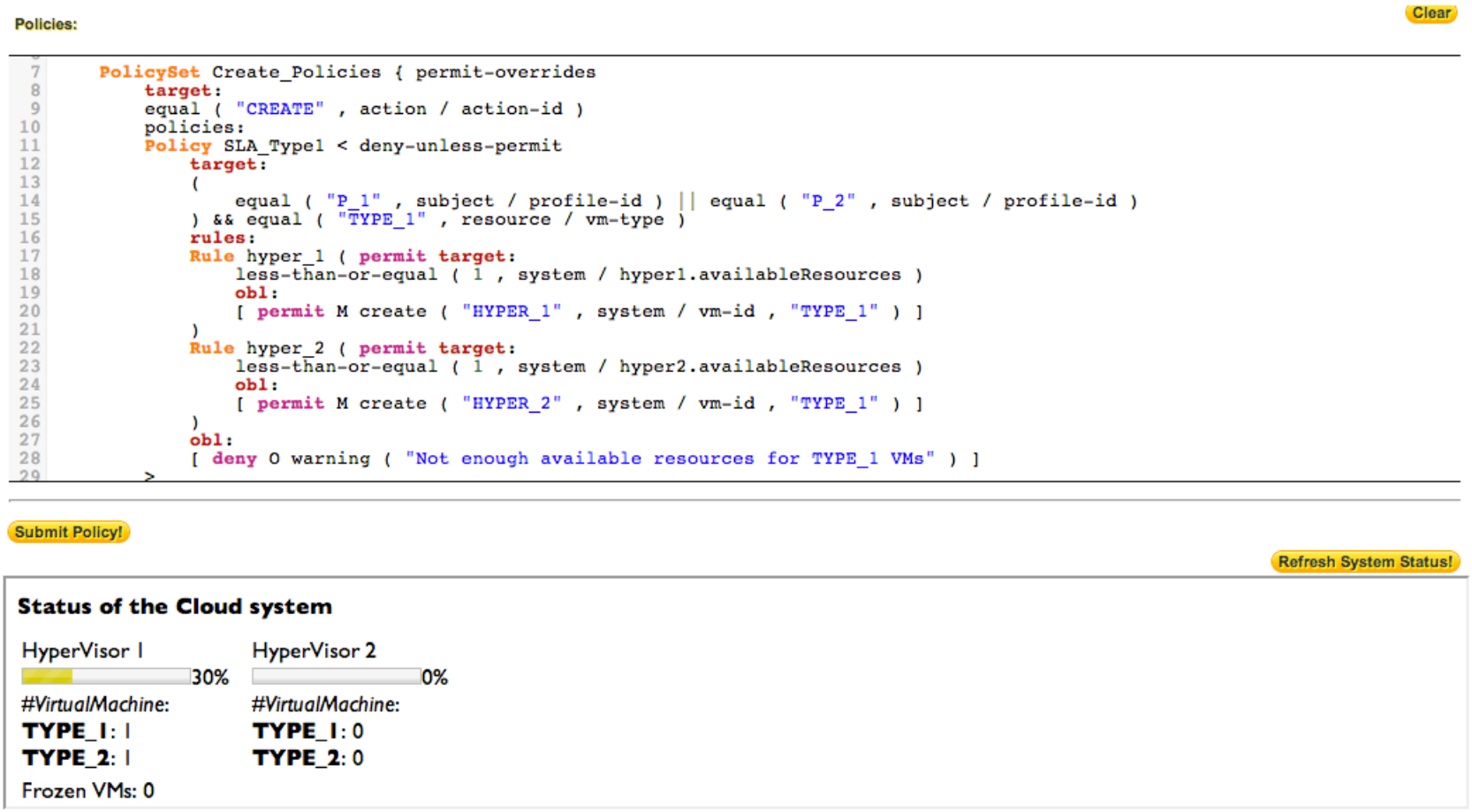
Scenario Implementation

Platform Design

- The platform is built as a web application on the top of a Tomcat server
- The Cloud Manager is the FACPL evaluation framework
- Two different panels (i.e. web pages) for *administrators* and *clients*

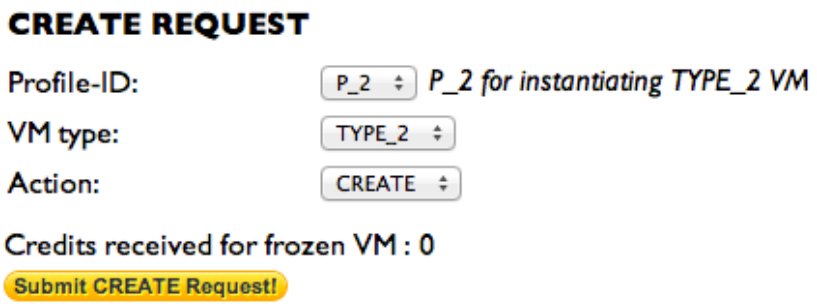
Web application

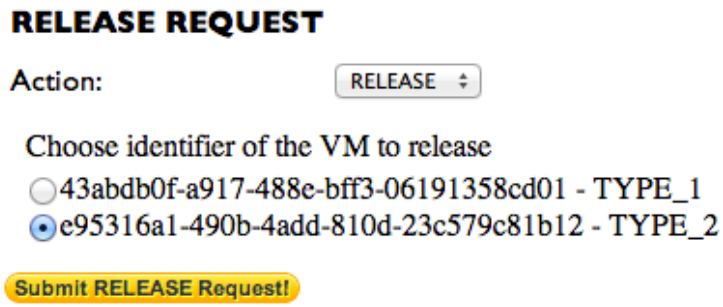
- A front-end for the administrator, from where he can manage the policies governing the hypervisors



Web application

- A front-end for the administrator, from where he can manage the policies governing the hypervisors
- A front-end for the clients, from where they can submit requests for the creation or the shutdown of VMs





Web application

- A front-end for the administrator, from where he can manage the policies governing the hypervisors
- A front-end for the clients, from where they can submit requests for the creation or the shutdown of VMs

The web application can be accessed at
`http://facpl.sf.net/cloud.html`

FACPL policies in Java

Starting from the FACPL code, using the tools, we can auto-generate the corresponding FACPL policy in Java format

```
public class Policy_SLA_Type1 extends Policy {
    public Policy_SLA_Type1() {
        addId("SLA_Type1");
        addCombiningAlg(it.unifi.facpl.lib.algorithm.DenyUnlessPermit.class);
        addTarget(new TargetTree(Connector.AND,
            new TargetTree(...), new TargetTree(...)));
        addRule(new hyper_1());
        addRule(new hyper_2());
        addObligation(new ObligationExpression("warning",Effect.DENY,
            TypeObl.O,"Not enough available resources for TYPE_1 VMs"));
    }
    private class hyper_1 extends Rule {
        hyper_1 (){
          addId("hyper_1");
          addEffect(Effect.PERMIT);
          addTarget( ... );
          addConditionExpression(null);
          addObligation(new ObligationExpression ("create",Effect.PERMIT,
            TypeObl.M,"HYPER_1",new StructName("system","vm-id"), "TYPE_1"));
        }
    }
    private class hyper_2 extends Rule { ... }
}
```

To sum up ...

Strategies enforced by FACPL policies:

- **Access Control**: regulation of user accesses according to user credentials, e.g. user profiles `subject/profile-id`
- **Resource Usage**: allocation of the right amount of resources needed to instantiate new VMs through attribute `resource/vm-type`
- **Adaptation**: re-configuration of the platform, i.e. through obligation `freeze`, to freeze some VMs and guarantee the committed SLAs

Policies 'interact' with the Cloud Platform

- Context Handling: policy evaluation uses contextual information accessed via attributes (e.g. `system/hyper1.availableResources`, `system/hyper1.vm1-counter`)
- Obligations: require dynamic fulfillment of actions that adapt the cloud platform (e.g. `create`, `freeze`)

FACPL vs. XACML

Policy	**Num. of lines**		**Saved**	**Num. of characters**		**Saved**
	XACML	FACPL	**lines**	XACML	FACPL	**chars**
SLA_Type1	162	22	86,42%	5.607	663	88,17%
SLA_Type2	349	36	89,68%	12.715	1.364	89,27%
Release_Policies	113	15	86,72%	4.193	438	89,55%
Overall policies	648	101	84,41%	23.436	3.030	87,07%

Concluding remarks

To sum up ...

FACPL:

- A compact syntax for writing attribute-based access control policies
- A rigorous evaluation process
- A formally grounded analysis technique
- A full-implemented Java-based toolchain

Application Domains

- *e-Health*: controlling access to resources
- *Cloud Computing*: controlling and allocating computing resources
- *Autonomic Computing*: defining adaptation strategies

For further details about FACPL, visit

http://facpl.sf.net

For experimenting FACPL online, try the Web Applications

http://facpl.sf.net/webapp.html

http://facpl.sf.net/cloud.html

References

A. Margheri, M. Masi, R. Pugliese, F. Tiezzi

A Rigorous Framework for Specification, Analysis and Enforcement of Access Control Policies

Transactions on Software Engineering, to appear - Available from the FACPL website

A. Margheri

A Formal Approach to Specification, Analysis and Implementation of Policy-based Systems

PhD Thesis, 2016

A. Margheri, R. Pugliese, F. Tiezzi

On Properties of Policy-based Specification

Automated Specification and Verification of Web Systems (*WWV*) - EPTCS, 2015

A. Margheri, M. Masi, R. Pugliese, F. Tiezzi

Developing and Enforcing Policies for Access Control, Resource Usage, and Adaptation. A Practical Approach

Web Services and Formal Methods (*WS-FM*) - Springer, 2013