Domain Specific Formal Languages

– The FACPL language –

Francesco Tiezzi

School of Science and Technology

Computer Science Division

University of Camerino

A.A. 2016/2017

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
- 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

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

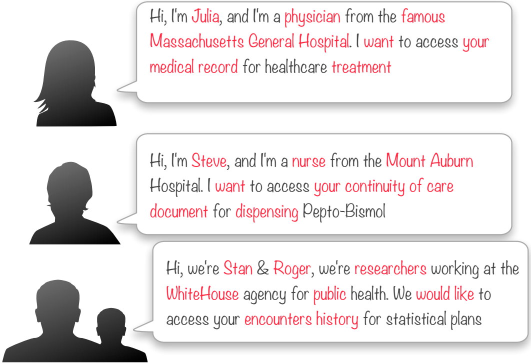
Hi, I'm Julia, and I'm a physician from the famous Massachusetts General Hospital. I want to access your medical record for healthcare treatment

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? ?!?!#*&ˆ%&$ (*&ˆ????

RBAC vs ABAC: an e-Health Scenario



- Different hospitals, different actions and different roles

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

ABAC

- Requester credentials are rendered as a collection of attributes, i.e. pairs (*name*, *value*)
  - Values from the context like, e.g., requester location and current time
- 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
- Last normative specification XACML 3.0: January 2013

An European eHealth Platform: the EU pilot epSOS

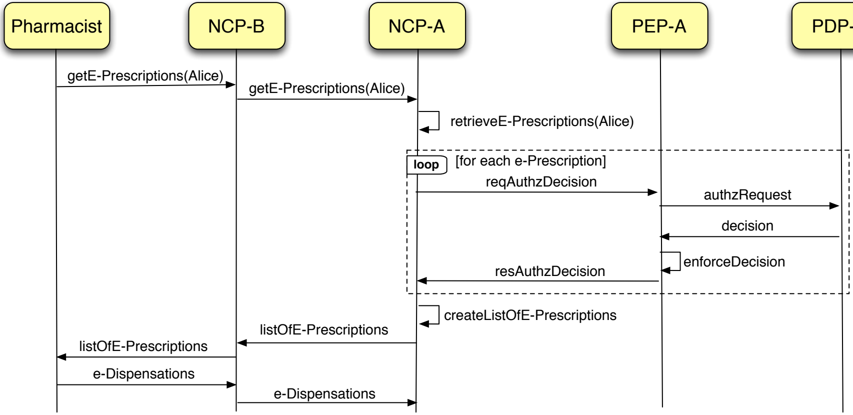
Objectives

- 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
- NCP-A enforces the patient 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 ≈ 240 lines, the all epSOS policies are ≈ 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 correct 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

Attributes

- Attributes are pairs ($name, value$) and form access requests
- Attribute values, which can be literals or sets, are accessed via names

E.g., given the attribute (subject/id, "*Andrea*"), 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 request
- deny: a policy forbids the access request
- not-applicable: no policy applies to the access request
- indeterminate: a policy is unable to evaluate the access request

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

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

FACPL Syntax (1 of 2)

Policy Auth. Systems	PAS	$::=$	$(\,\mathsf{pep} : \mathit{EnfAlg}\quad \mathsf{pdp} : \mathit{PDP}\,)$
Enf. algorithms	EnfAlg	$::=$	$\mathsf{base} \quad\mid\quad \mathsf{deny\text{-}biased} \quad\mid\quad \mathsf{permit\text{-}biased}$
Policy Decision Points	PDP	$::=$	$\{\mathit{Alg}\quad \mathsf{policies} : \mathit{Policy}^{+}\}$
Combining algorithms	Alg	$::=$	$\mathsf{permit\text{-}overrides}_\delta \quad\mid\quad \mathsf{deny\text{-}overrides}_\delta$
		$\mid$	$\mathsf{deny\text{-}unless\text{-}permit}_\delta \quad\mid\quad \mathsf{permit\text{-}unless\text{-}deny}_\delta$
		$\mid$	$\mathsf{first\text{-}app}_\delta \quad\mid\quad \mathsf{one\text{-}app}_\delta$
		$\mid$	$\mathsf{weak\text{-}consensus}_\delta \quad\mid\quad \mathsf{strong\text{-}consensus}_\delta$
Fulfilment strategies	δ	$::=$	$\mathsf{greedy} \quad\mid\quad \mathsf{all}$
Policies	Policy	$::=$	$(\mathit{Effect}\quad \mathsf{target} : \mathit{Expr}\quad \mathsf{obl} : \mathit{Obligation}^{*}\,)$
		$\mid$	$\{\mathit{Alg}\ \mathsf{target} : \mathit{Expr}\ \mathsf{policies} : \mathit{Policy}^{+}\ \mathsf{obl} : \mathit{Obligation}^{*}\}$
Effects	Effect	$::=$	$\mathsf{permit} \quad\mid\quad \mathsf{deny}$
Obligations	$\mathit{Obligation}$	$::=$	$[\,\mathit{Effect}\quad \mathit{ObType}\quad \mathit{Action}(\mathit{Expr}^{*})\,]$
Obligation types	ObType	$::=$	$\mathsf{M} \quad\mid\quad \mathsf{O}$

FACPL Syntax (2 of 2)

$$
\begin{array}{rrcl}
\textbf{Expressions} & \mathit{Expr} & ::= & \mathit{Name} \quad | \quad \mathit{Value} \\
 & & | & \mathrm{and}(\mathit{Expr}, \mathit{Expr}) \quad | \quad \mathrm{or}(\mathit{Expr}, \mathit{Expr}) \quad | \quad \mathrm{not}(\mathit{Expr}) \\
 & & | & \mathrm{equal}(\mathit{Expr}, \mathit{Expr}) \quad | \quad \mathrm{in}(\mathit{Expr}, \mathit{Expr}) \\
 & & | & \text{greater-than}(\mathit{Expr}, \mathit{Expr}) \quad | \quad \mathrm{add}(\mathit{Expr}, \mathit{Expr}) \\
 & & | & \mathrm{subtract}(\mathit{Expr}, \mathit{Expr}) \quad | \quad \mathrm{divide}(\mathit{Expr}, \mathit{Expr}) \\
 & & | & \mathrm{multiply}(\mathit{Expr}, \mathit{Expr}) \\
\textbf{Attribute names} & \mathit{Name} & ::= & \mathit{Identifier} / \mathit{Identifier} \\
\textbf{Literal values} & \mathit{Value} & ::= & \mathrm{true} \mid \mathrm{false} \mid \mathit{Double} \mid \mathit{String} \mid \mathit{Date} \\
\textbf{Requests} & \mathit{Request} & ::= & (\mathit{Name}, \mathit{Value})^{+}
\end{array}
$$

FACPL Formal Semantics: given in a denotation style

E.g., the case of policies is

$$\mathcal{P}[\![\{a \;\; \mathsf{target} : expr \;\; \mathsf{policies} : \pi^{+} \; \mathsf{obl} : o^{*} \;\}]\!]r =$$

$$\begin{cases} \langle e \;\; fo_1^* \bullet fo_2^* \rangle & \texttt{if } \mathcal{E}[\![expr]\!]r = \mathsf{true} \;\wedge\; \mathcal{A}[\![a, \pi^{+}]\!]r = \langle e \;\; fo_1^* \rangle \wedge \; \mathcal{O}[\![o^*|_e]\!]r = fo_2^* \\ \mathsf{not\text{-}applicable} & \texttt{if } \mathcal{E}[\![expr]\!]r = \mathsf{false} \;\vee\; \mathcal{E}[\![expr]\!]r = \bot \\ & \qquad \vee\; (\mathcal{E}[\![expr]\!]r = \mathsf{true} \;\wedge\; \mathcal{A}[\![a, \pi^{+}]\!]r = \mathsf{not\text{-}applicable}) \\ \mathsf{indeterminate} & \texttt{otherwise} \end{cases}$$

The function $\mathcal{A}$ defining the semantics of combining algorithms relies on binary operators defined as

$\otimes$permit-overrides	$\langle$permit $FO_2\rangle$	$\langle$deny $FO_2\rangle$	not-applicable	indeterminate
$\langle$permit $FO_1\rangle$	$\langle$permit $FO_1 \bullet FO_2\rangle$	$\langle$permit $FO_1\rangle$	$\langle$permit $FO_1\rangle$	$\langle$permit $FO_1\rangle$
$\langle$deny $FO_1\rangle$	$\langle$permit $FO_2\rangle$	$\langle$deny $FO_1 \bullet FO_2\rangle$	$\langle$deny $FO_1\rangle$	indeterminate
not-applicable	$\langle$permit $FO_2\rangle$	$\langle$deny $FO_2\rangle$	not-applicable	indeterminate
indeterminate	$\langle$permit $FO_2\rangle$	indeterminate	indeterminate	indeterminate

Specification of FACPL policies

Specification of the epSOS Access Control System

Security Requirements

The access control system must ensure the following security requirements:

1. Doctors can write e-Prescriptions
2. Doctors can read e-Prescriptions
3. Pharmacists 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

- Items 1 - 3: *closed-world* requirements stating the allowed accesses
- Items 4 - 6: additional functionalities required for managing accesses

Specification Steps

On the base of the security requirements . . .

1. Assume each relevant requester credential is represented by a pre-defined attribute (n, v). E.g.

   - Requester role:
     - n = subject/role
     - v ∈ {"*doctor*", "*pharmacist*"}
   - Requested action:
     - n = action/id
     - v ∈ {"*read*", "*write*"}

2. Write basic access rules by defining controls on attributes

3. Combine basic access rules into policies

4. Possibly combine policies hierarchically

Step1: Writing Rules

- Doctors 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 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 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 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 can read e-Prescriptions

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

- Pharmacists 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:
   [permit 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. the logging of the allowed accesses

Step3: Validating the Policy (1/2)

Let us consider the requirement: "*Doctors can write e-Prescriptions*"

The following request must be 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)

Let us consider the requirement: "*Doctors can write e-Prescriptions*"

The following request must be 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))
```

Step3: Validating the Policy (1/2)

Let us consider the requirement: "*Doctors can write e-Prescriptions*"

The following request must be 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))
```

Step3: Validating the Policy (1/2)

Let us consider the requirement: "*Doctors can write e-Prescriptions*"

The following request must be 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))
```

Step3: Validating the Policy (1/2)

Let us consider the requirement: "*Doctors can write e-Prescriptions*"

The following request must be 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")
}
```

- As expected, the application of the combining algorithm `permit-overrides-all` to the decisions permit, not-applicable, not-applicable returns permit

Step3: Validating the Policy (2/2)

Let us consider the requirement: “Pharmacists can read e-Prescriptions”

Due to the *closed-world* nature of the requirements, the following request, representing a pharmacist willing to write an e-Prescription, must 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)

Let us consider the requirement: "Pharmacists can read e-Prescriptions"

Due to the *closed-world* nature of the requirements, the following request, representing a pharmacist willing to write an e-Prescription, must 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))
```

Step3: Validating the Policy (2/2)

Let us consider the requirement: "Pharmacists can read e-Prescriptions"

Due to the *closed-world* nature of the requirements, the following request, representing a pharmacist willing to write an e-Prescription, must 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))
```

Step3: Validating the Policy (2/2)

Let us consider the requirement: “Pharmacists can read e-Prescriptions”

Due to the *closed-world* nature of the requirements, the following request, representing a pharmacist willing to write an e-Prescription, must 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))
```

Step3: Validating the Policy (2/2)

Let us consider the requirement: "Pharmacists can read e-Prescriptions"

Due to the *closed-world* nature of the requirements, the following request, representing a pharmacist willing to write an e-Prescription, must 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")
}
```

- Now, 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: the patient-informed consent policy

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

 policies:

   PolicySet ePre { ... }

   Rule ruleDeny (deny)

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

The policy can be amended by introducing an additional layer comprising

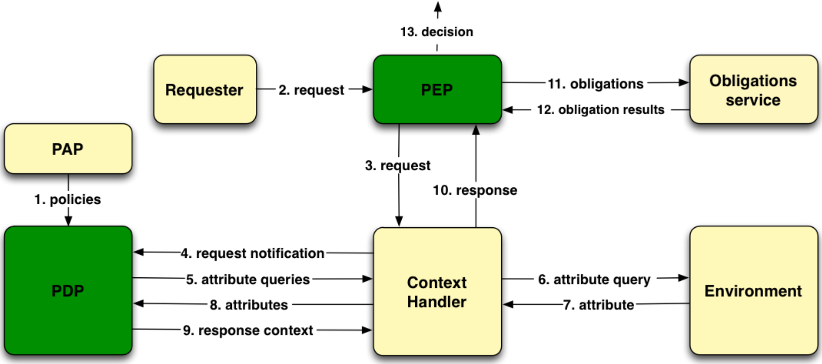
- a target matching any request
- the policy managing the e-Prescription
- the always applicable rule *deny*
- two obligations enforcing the Requirements 5 & 6

Step5: Alice patient-informed consent policy

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

 policies:

   PolicySet ePre { ... }

   Rule ruleDeny (deny)

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

The target is tailored thus to only apply to requests regarding Alice

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

Analysis of FACPL 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 authorisations of a single request and, possibly, of its extensions

- Structural Properties

  characterisations of the relationships among policy rules with respect to the authorisations they enforce

Difficulties to tackle:

- *Hierarchical policies* featuring *combining algorithms*
- Role of *missing* and *erroneous* attributes
- Various expressions and controls on attribute values, e.g. arithmetic and comparison operators

Authorisation Properties

Conditions on the authorisations of a single request and, possibly, of its extensions

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

Additional attributes

Extending the request with additional attributes might change the authorisation of a request in a possibly unexpected way

Authorisation Properties (cont.)

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")
}
```

The attribute with name action/id is missing. If the request is extended with the following attributes, we have

- (action/id, "*read*"): the previous policy evaluates it to permit
- (action/id, "*write*"): the previous policy evaluates it to deny

Different values assumed by the same attribute may lead to different, possibly unexpected, authorisation decisions

Structural Properties

Characterisations of the relationships among policy rules with respect to the authorisations they enforce

Multiple structural properties of interest, we address

- *Completeness*: if there is no access request for which there is an absence of decision
- *Coverage*: if the set of authorisations enforced by a policy is covered by that of another policy
- *Disjointness*: two or more policies enforce disjoint sets of authorisations

Representing FACPL Policies with SMT

- 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
- Each policy is represented by a *4-tuple of constraints*, one for each possible decision
- Each attribute is modelled by a *3-valued record* representing
  - its (typed) value
  - if it is missing
  - if it is of an unexpected type
- Policy hierarchies are *flattened* according to the (binary operator) semantics of combing algorithms

For all $\pi \in \mathit{Policy}$ enclosing combining algorithms only using all as fulfilment strategy, and for all $r \in R$, it holds that

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

Constraint Generation and Property Verification

The first epSOS rule corresponds to the following tuple of constraints

$$\begin{aligned}\langle\ & \text{permit}: \ \chi_{trg1} \wedge \text{true} \\ & \text{deny}: \ \text{false} \\ & \text{not-applicable}: \ \neg\chi_{trg1} \\ & \text{indeterminate}: \ \neg(\texttt{isBool}(\chi_{trg1}) \vee \texttt{isMiss}(\chi_{trg1})) \vee (\chi_{trg1} \wedge \neg\text{true})\ \rangle\end{aligned}$$

where

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

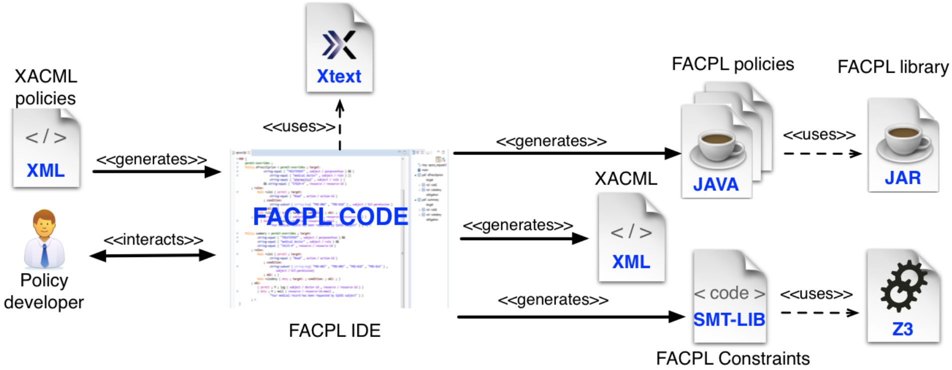
This tuple is then combined with the tuples representing the other rules according to the semantics of the combining algorithms

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 IDE (an Xtext-based plug-in)
  - Web Application for experimenting FACPL directly online
- Java Design and Evaluation library
- 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 and XACML code

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

Additional Application Domains

- *Cloud Computing*: controlling and allocating computing resources
- *Autonomic Computing*: defining adaptation strategies by using a *policy-based* approach

Ongoing and Future Works

Enhancing FACPL to support Usage Control

- *Continuative Access Control*

  checking how assigned access rights are actually used by requesters (e.g., secondary use of data)

- *History-based Access Control*

  evaluating access requests on the base of the previous (allowed) accesses (e.g., dynamic separation of duty and Chinese wall requirements)

High-level design of FACPL policies (or, more in general, of ABAC policies)

Thank you!

For further details about FACPL, visit

http://facpl.sf.net

For experimenting FACPL online, try the web application

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

References

A. Margheri, M. Masi, R. Pugliese, F. Tiezzi
A Rigorous Framework for Specification, Analysis and Enforcement of Access Control Policies
Technical Report, 2016 - 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