Software Engineering for Service-Oriented and Autonomic Systems
– The FACPL language –

Francesco Tiezzi

IMT - Institutions, Markets, Technologies
Institute for Advanced Studies Lucca

IMT Lucca - October, 2014

In co-operation with ASCENS members, in particular A. Margheri (UNIFI), M. Masi (TIANI) and R. Pugliese (UNIFI)
Motivations

- Overgrowth of information, computational and network infrastructures
- Distributed and heterogeneous systems
- New technologies, such as Cloud Computing, Web Services, ...

High-level abstractions are needed in order to
- usable means capable to define various aspects of systems’ behaviour (e.g., access control, resource usage, adaptation) in an integrated way
- lead the design and management of applications

Policy-based approach
Declarative set of rules expressing what can be computed in the system
Motivations

High-level abstractions are needed in order to

- usable means capable to define various aspects of systems’ behaviour (e.g., *access control*, *resource usage*, *adaptation*) in an integrated way
- lead the design and management of applications
Access Control

Require access to resources and systems
Access Control
Access Control

Send a request

authorized user

resource (e.g. EHR)
Access Control

authorized user

resource (e.g. EHR)
Access Control

Unauthorized user

Resource (e.g. EHR)
Role-based Access Control (RBAC)

The permissions to perform certain operations are assigned to a specific role. For example, *manager* can access different resources than *employee*

- **Pros:** user’s group and hierarchy of groups
- **Cons:** scalability problems, i.e. it is essential to know (in advance) the role’s population
- **Cons:** defining fine-grained rules is cumbersome
Access Control Models: e-Health domain example

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

- **Different hospitals, different actions and different roles**
- **RBAC: difficult to define fine-grained rules**
- **How to (easily) encode such requests for a software actor?**

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.
Access Control Models: e-Health domain example

- Different hospitals, different actions and different roles
- RBAC: difficult to define fine-grained rules
- How to (easily) encode such requests for a software actor?

Attribute-Based access control - ABAC
Defining an access control model based not only on roles but on the more general concept of attribute, i.e. all the security-relevant information of the requester, the environment, etc.

Policy-Based access control - PBAC
Organising attribute-based rules into policies, i.e. sets of structured rules
The XACML standard

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

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

- First normative specification February 2003
- XACML 3.0 since January 2013
XACML: a PBAC implementation

eXtensible Access Control Markup Language is an OASIS standard

```xml
<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>
          </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>
```
XACML: a PBAC implementation

*eXtensible Access Control Markup Language* is an OASIS standard

### XACML weaknesses

- The language has a verbose syntax
- XACML comes without a formal semantics
  - Lack of official tools
  - Various attempts to formalise XACML semantics

### Our Proposal

We propose a formal policy language called FACPL

- Compact and intuitive syntax
- Endowed with a *formal semantics*
- Takes inspiration from XACML (interoperability is guaranteed)
Policy-based Evaluation Process

- **PDP** decides whether to allow received requests or not returning
  - a decision
  - a (possibly empty) list of obligations
- **PEP** enforces the decision taken by the PDP
FACPL: formal language for access control policies
FACPL: Formal Access Control Policy Language

Formal language to specify and analyse access controls

- BNF-based syntax
- Formal semantics
- Compliant and interoperable with XACML 3.0

FACPL Syntax (the same structure as XACML)

- **Rule**: the basic element defining the first level of access controls
- **Policy**: a list of **rules** or of **policies** themselves
  - Explicit conflicts resolution strategies: *combining algorithms*
- **Request**: a set of **attributes** expressing user’s credentials, capabilities, and actions requested
FACPL Elements

A structured specification: Policies, Rules

- **Policies** specify
  - a set of contained elements (policies or rules)
  - a combining algorithm
  - a target to which the policy applies
  - a set of obligations

- **Rules** specify
  - an effect the access control decision (i.e. permit or deny)
  - a target, which refines the applicability of the rule
  - a condition, which further refines the applicability of the rule
  - a set of obligations

- **Combining algorithms**: strategies for calculating policy results
  (permit-overrides, only-one-applicable, first-applicable, deny-unless-permit, ...)

- **Obligations**:
  - an effect (i.e. permit or deny) for applying the obligation
  - name and arguments of the actions to be performed by the PEP
FACPL Policy Syntax

Policy Authorization Framework

\[ PAF ::= \{ \text{pdp} : PDP \text{ pep} : PEP \} \]

Policy Enforcement Point

\[ PEP ::= \text{base} | \text{deny-biased} | \text{permit-biased} \]

Policy Decision Point

\[ PDP ::= \{ \text{Palg} \text{ policies} : Policy^+ \} \]

Combining algorithms

\[ Alg ::= \text{deny-overrides} | \text{permit-overrides} | \text{deny-unless-permit} \ldots \]

Policies: atomic policies and policy sets

\[ Policy ::= \langle Alg \text{ target} : Target^? \text{ rules} : Rule^+ \text{ obl} : Obligation^* \rangle \]
\[ \quad | \quad \{ Alg \text{ target} : Target^? \text{ policies} : Policy^+ \text{ obl} : Obligation^* \} \]

Rules

\[ Rule ::= (\text{Effect} \text{ target} : Target^? \text{ condition} : BoolExpr^? \text{ obl} : Obligation^* ) \]

Effects

\[ Effect ::= \text{permit} | \text{deny} \]
FACPL Policy Syntax (cont.)

Targets

\[ \text{Target} ::= \text{MatchId}(\text{Value},\text{Name}) \mid \text{Target} \land \text{Target} \mid \text{Target} \lor \text{Target} \]

Matching functions

\[ \text{MatchId} ::= \text{equal} \mid \text{not-equal} \mid \text{greater-than} \mid \text{less-than} \mid \text{greater-than-or-equal} \mid \text{less-than-or-equal} \]

Expression

\[ \text{Expression} ::= \text{BoolExpr} \mid \text{StringExpr} \mid \text{ArithExpr} \mid \text{DateExpr} \]

Names

\[ \text{Name} ::= \text{Identifier}/\text{Identifier} \]

Obligations

\[ \text{Obligation} ::= [\text{Effect Type Action(Expression*})] \]

Type: mandatory or optional

\[ \text{Type} ::= M \mid O \]
A FACPL policy - e-Health case study

Policy medicalRecord < permit- overrides
  target:  
    equal ( "medical record" , resource/id)
  rules:
    Rule rule1 ( permit target:
      equal ("doctor", subject/role) & & equal ("write", action/id)
      condition:
      subset (bag("e-Rec-Create","e-Rec-Access"),subject/permission))
    Rule rule2 ( permit target:
      equal ("doctor", subject/role) & & equal ("read" , action/id)
      condition: 
      subset (bag("e-Rec-Access"), subject/permission))
    Rule rule3 ( deny target:
      equal ("nurse", subject/role) & & equal ("read" , action/id))
  obl: [ permit M log (subject/id, resource/id)]

More compact and more AC rules than the previous XACML policy
FACPL - Formal Semantics

Function $[\cdot]_R$

- **Input**: a set of possible requests (i.e., the set $R$)
- **Output**: a partition of $R$ w.r.t. the PDP and the PEP decisions

1. $R$ is first evaluated by the PDP returning a decision tuple $AD$
   
   \[
   AD = \begin{cases}
   \text{permit} : \{ \langle r_i, FO_i \rangle \}_{r_i \in R'_p} & \text{deny} : \{ \langle r_j, FO_j \rangle \}_{r_j \in R'_d} \\
   \text{not-applicable} : R'_n & \text{indeterminate} : R'_i
   \end{cases}
   \]

2. $AD$ is passed to the PEP which returns an enforceable decision $ED$
   
   \[
   ED = \begin{cases}
   \text{permit} : R_p & \text{deny} : R_d & \text{not-applicable} : R_n & \text{indeterminate} : R_i
   \end{cases}
   \]

   where all $FO_i$ has been discharged

Define the constraints under which a policy evaluates to a certain decision

Formal foundations for defining analysis techniques and implementing evaluation tools
The clause for the atomic target is as follows

\[
\text{\texttt{MatchId}}(\text{Value}, \text{Name}) \in \text{Target} \\
\text{\texttt{match}} : \{ \{ \text{r} \in \text{R} \mid \exists (\text{Name}, \text{Value'}) \in \text{r} : \text{MatchId}(\text{Value}, \text{Value'}) = \text{true} \} \\
\text{no-match : } \{ \{ \text{r} \in \text{R} \mid \forall (\text{Name}, \text{Value'}) \in \text{r} : \text{MatchId}(\text{Value}, \text{Value'}) = \text{false} \} \\
\text{indeterminate : } \{ \{ \text{r} \in \text{R} \mid \exists (\text{Name}, \text{Value'}) \in \text{r} : \text{MatchId}(\text{Value}, \text{Value'}) ⇓ \text{fail} , \exists (\text{Name}, \text{Value'}) \in \text{r} : \text{MatchId}(\text{Value}, \text{Value'}) = \text{true} \} \}
\]

which defines the constraints on values for which a request matches or not.

Given the decision of the internal elements, the clause for the rule is

\[
\text{\texttt{(Effect target : Target condition : BoolExpr obl : Obligation*})} \in \text{Target} \\
\text{\texttt{match}} : \{ \{ \text{r} \in \text{Target}R \downarrow \text{match} , \text{r} \models \text{BoolExpr} = \text{true} \} \\
\text{not-applicable : } \{ \{ \text{r} \in \text{Target}R \downarrow \text{match} \uparrow \text{no-match} \} \cup \{ \{ \text{r} \in \text{Target}R \downarrow \text{match} \uparrow \text{indeterminate} \} \cup \{ \{ \text{r} \in \text{Target}R \downarrow \text{match} \uparrow \text{indeterminate} \} \}
\]

which defines the constraints on values for which a request matches or not.
Finally, we define the clause for policy as follows

\[
\begin{align*}
\langle \text{Alg target : } Target \text{ rules : } Rule^+ \text{ obl : } Obligation^* \rangle_R &= \\
\text{permit} : \{ \langle r, FO \cdot (Obligation^*_{\text{permit}}) \rangle_r \mid \langle r, FO \rangle \in \text{Alg}(Rule^+)_{R_m} \downarrow_{\text{permit}} \} \\
\text{deny} : \{ \langle r, FO \cdot (Obligation^*_{\text{deny}}) \rangle_r \mid \langle r, FO \rangle \in \text{Alg}(Rule^+)_{R_m} \downarrow_{\text{deny}} \} \\
\text{not-applicable} : R_n \cup \text{Alg}(Rule^+)_{R_m} \downarrow_{\text{not-applicable}} \\
\text{indeterminate} : R_i \cup \text{Alg}(Rule^+)_{R_m} \downarrow_{\text{indeterminate}} \\
\cup \{ r \in R_m \mid \langle r, FO \rangle \in \text{Alg}(Rule^+)_{R_m} \downarrow_{\text{permit}}, (Obligation^*_{\text{permit}})_r = \text{undef} \} \\
\cup \{ r \in R_m \mid \langle r, FO \rangle \in \text{Alg}(Rule^+)_{R_m} \downarrow_{\text{deny}}, (Obligation^*_{\text{deny}})_r = \text{undef} \} 
\end{align*}
\]

where \( \text{Alg}(Rule^+) \) implements the chosen combing algorithm.

Each algorithm is defined with a sort of truth table, as e.g.

<table>
<thead>
<tr>
<th>permit-overrides</th>
<th>permit</th>
<th>deny</th>
<th>not-applicable</th>
<th>indeterminate</th>
</tr>
</thead>
<tbody>
<tr>
<td>permit</td>
<td>permit</td>
<td>permit</td>
<td>permit</td>
<td>permit</td>
</tr>
<tr>
<td>deny</td>
<td>permit</td>
<td>deny</td>
<td>deny</td>
<td>deny</td>
</tr>
<tr>
<td>not-applicable</td>
<td>permit</td>
<td>deny</td>
<td>not-applicable</td>
<td>not-applicable</td>
</tr>
<tr>
<td>indeterminate</td>
<td>permit</td>
<td>indeterminate</td>
<td>indeterminate</td>
<td>indeterminate</td>
</tr>
</tbody>
</table>
Outcomes of FACPL semantics

Given a FACPL policy, the formal semantics is used for identifying the constraints on the values of attributes for which requests evaluate to certain decisions.

E.g., the first FACPL e-Health policy has the following permit constraints:

\[(\text{resource/id, “medical record”}) \in r \land \left( (\text{(subject/role, “doctor”}) \in r \land (\text{action/id, “write”}) \in r \land \left( (\text{(subject/permission, \{“e-Rec-Create”, “e-Rec-Access”\}}) \in r \right) \lor \left( (\text{(subject/role, “doctor”}) \in r \land (\text{action/id, “read”}) \in r \land \left( (\text{(subject/permission, \{“e-Rec-Access”\}}) \in r \right) \right) \right) \right) \]
FACPL applications: eHealth, cloud and autonomic computing
Developing FACPL Tools

The FACPL Eclipse IDE offers

- **supporting features** for developing FACPL policies
  (code suggestion and completion, cross-references, highlighting of code, etc.)
- **evaluation of FACPL** policy by using the dedicated Java libraries
- **automatic creation of constraint** (to be completed)
The development of FACPL Java libraries and constraint creation has been guided by the formal semantics.
Eclipse development environment (Xtext-based plug-in)
  - Interoperability with XACML
Java Design library
Java Evaluation library
Web Application for evaluating policies directly in the browser
FACPL Applications

Some Application Domains

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

**eHealth** Application:

- Ensuring confidentiality of *high sensitive* medical information
- Easy for security architects to manage, rather than XML
- Case-Study from the European epSOS Project

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

On a formal and user-friendly linguistic approach to access control of electronic health data

Cloud Scenario

- A IaaS provider offers customers a range of pre-configured VMs: TYPE_1 and TYPE_2
- Each type of VM uses specific amounts of computing capacity
- SLA guarantees on the instantiation of TYPE_2 VM

- Resource Usage: allocation of the right amount of resources needed to instantiate new VMs, while respecting committed SLAs
- Adaptation: freezing some TYPE_1 VMs to re-configure the system in order to instantiate a TYPE_2 VM (SLA guarantees)
- Access Control: allowing user to use Cloud platform services only if user’s profile has the needed credentials

How Policies interact with Cloud Platform

- Context Handling: policy evaluation is based on contextual information
- Obligations: actions modifying the cloud platform
A. Margheri, M. Masi, R. Pugliese, F. Tiezzi
Developing and Enforcing Policies for Access Control, Resource Usage, and Adaptation. A Practical Approach
FACPL Application: Policing autonomic systems

1. FACPL has been integrated in a language for modelling autonomic systems (called SCEL) in order to define adaptation strategies.

2. FACPL tools has been included in the SCEL evaluation environment.

Two levels of Linguistic Abstractions:
- the *computational behaviour of each component* (i.e. process actions)
- the *interaction and adaptation logic among components* (i.e. FACPL policies)

Policy-Based Adaptation

FACPL policies decide whether an action is *authorised* and if additional actions are needed before the process continuation.

We provide two different mechanisms for adaptation:
- *Obligations* → actions enforcing adaptation strategies
- *Policy automaton* → dynamic changing of adaptation strategies
More on the Formal Language

We have defined

- inference rules expressing the FACPL semantics
- the integration of FACPL within SCEL operational semantics linking the authorisation of a process action to the FACPL inference rules

The authorisation predicates, e.g. $I : t \triangleright J$, use the FACPL inference rules for calculating a decision $d$ and a sequence $s$ of obligations $\pi, r \vdash d, s$

An example of the inference rules defined for FACPL is as follows

**Policy Inference rules: FACPL rule**

\[
\begin{array}{c}
\tau, r \vdash \text{applicable} \quad o^*, r, \text{permit} \vdash s \\
(\text{permit target} : \tau \text{ obl} : o^*), r \vdash \text{permit}, s
\end{array}
\]

\[
\begin{array}{c}
\tau, r \vdash \text{applicable} \quad o^*, r, \text{deny} \vdash s \\
(\text{deny target} : \tau \text{ obl} : o^*), r \vdash \text{deny}, s
\end{array}
\]
Language Implementation: jRESP environment

Basic design principles
- Java-based (direct integration with FACPL libraries)
- no centralised control (one thread for each process)
- heavy use of *recurrent* patterns

jRESP has been employed in various autonomic case studies, e.g. Robot Swarm and collaboration Cloud Platform
Language Implementation: jRESP environment

Basic design principles

- Java-based (direct integration with FACPL libraries)
- no centralised control (one thread for each process)
- heavy use of recurrent patterns

jRESP has been employed in various autonomic case studies, e.g. Robot Swarm and collaboration Cloud Platform

A.Margheri, R.Pugliese, F.Tiezzi
Linguistic Abstractions for Programming and Policing Autonomic Computing Systems
ATC 2013

M. Loreti, A.Margheri, R.Pugliese, F.Tiezzi
On Programming and Policing Autonomic Computing Systems
ISoLA 2014
Thank you!

For further details about FACPL language, visit

http://rap.dsi.unifi.it/facpl/
References

A. Margheri, M. Masi, R. Pugliese, F. Tiezzi
On a formal and user-friendly linguistic approach to access control of electronic health data

A. Margheri, M. Masi, R. Pugliese, F. Tiezzi
Developing and Enforcing Policies for Access Control, Resource Usage, and Adaptation. A Practical Approach

A. Margheri, R. Pugliese, F. Tiezzi
Linguistic Abstractions for Programming and Policing Autonomic Computing Systems
Autonomic and Trusted Computing (*ATC*) - IEEE, 2013

M. Loreti, A. Margheri, R. Pugliese, F. Tiezzi
On Programming and Policing Autonomic Computing Systems
*ISoLA* - Springer, 2014