Representing Security Policies in Web Information Systems

Félix J. García Clemente

Gregorio Martínez Pérez

Juan A. Botía Blaya

Antonio F. Gómez-Skarmeta

{fgarcia, gregorio, skarmeta}@dif.um.es, juanbot@um.es

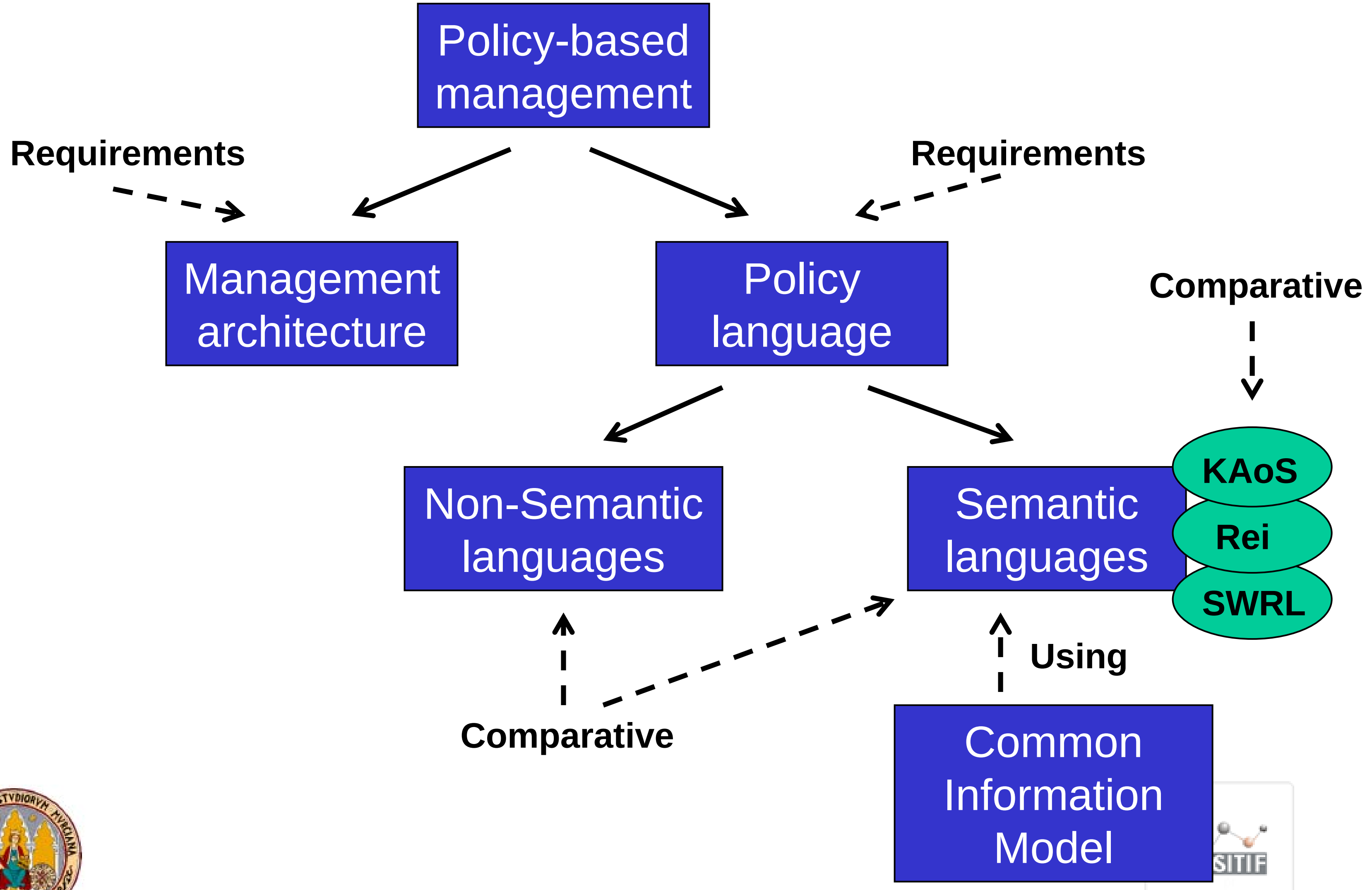
University of Murcia (UMU)

SPAIN

Presentation Overview

- Introduction
- Requirements for a policy framework
- Advantages of semantic security policy framework
- Semantic security policy languages
- Using CIM Ontology with semantic languages
- Conclusions

Introduction



Presentation Overview

- Introduction
- Requirements for a policy framework
- Advantages of semantic security policy framework
- Semantic security policy languages
- Using CIM Ontology with semantic languages
- Conclusions

Requirements for a policy framework (1/2)

- Requirements for a policy language
  - **Well-defined**.
    - Syntax and structure is clear and no-ambiguous
    - Independent of its particular implementation
  - **Flexibility and extensibility**.
    - Flexible enough to allow new policy information to be expressed
    - Extensible enough to allow new types of policy to be added
  - **Interoperability with other languages**.
    - Interoperability is a must to allow different services or applications from different domains to communicate with each other

Requirements for a policy framework (2/2)

- Requirements for a policy management architecture
  - **Well-defined interface**.
    - Interface independent of the particular implementation in use
    - Interfaces between the components need to be clear and no-ambiguous
  - **Flexibility and definition of abstractions to manage a wide variety of device types**.
    - Flexible enough to allow addition of new types of devices with minimal updates and recoding of existing management components
  - **Interoperability with other architectures (inter-domain)**.
  - **Conflict Detection**.
    - It has to be able to check that a given policy does not conflict with any other existing policy.
  - **Scalability**.
    - It should maintain quality performance under an increased system load.

Presentation Overview

- Introduction
- Requirements for a policy framework
- Advantages of semantic security policy framework
- Semantic security policy languages
- Using CIM Ontology with semantic languages
- Conclusions

Advantages of semantic security policy framework (1/2)

- There are some non-semantic security policy frameworks such as:
  - **Ponder**, is a declarative, object-oriented language developed for specifying management and security policies. Ponder permits to express authorizations, obligations, information filtering, refrain policies, and delegation policies.
  - The eXtensible Access Control Markup Language (**XACML**) describes both an access control policy language and a request/response language.

Advantages of semantic security policy framework (2/2)

	Semantic Languages	Non-Semantic Languages
Abstraction	Multiple levels	Medium and low level
Extensibility	Easy and at runtime	Complex and at compile-time
Representability	Complex environments	Specific environments
Readability	Specialized tools	Direct
Interoperation	By common ontology	By interfaces
Enforcement	Complex	Easy

* Based on [Tonti, et al., Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder.] and complemented with our own analysis

Presentation Overview

- Introduction
- Requirements for a policy framework
- Advantages of semantic security policy framework
- Semantic security policy languages
- Using CIM Ontology with semantic languages
- Conclusions

Semantic security policy languages (1/5)

- Case study:
  - Natural language:
    - “Permit the access to the e-payment service, if the user is in the group of customers registered for this service”.
  - As a set of IF-THEN policy rules:
    - “IF ((<Requester> is member of Payment Customers) AND (<Server> is member of Payment Servers)) THEN (<Requester> granted access to <Server>)”
- Three approaches:
  - KAoS
  - Rei
  - SWRL

Semantic security policy languages (2/5)

- KAoS

```
<owl:Class rdf:ID="PaymentAuthAction">
<owl:intersectionOf
    rdf:parseType="owl:collection">
  <owl:Class
    rdf:about="&action;AccessAction"/>
  <owl:Restriction>
  <owl:onProperty
    rdf:resource="&action;#performedBy"/>
  <owl:toClass
    rdf:resource="&domains;MembersOf
    PayCustomer"/>
  </owl:Restriction>
  <owl:Restriction>
  <owl:onProperty
    rdf:resource="&action;#performedOn"/>
 <owl:toClass
    rdf:resource="&domains;MembersOf
    PayServer"/>
  </owl:Restriction>
</owl:intersectionOf>
</owl:Class>
<policy:PosAuthorizationPolicy
    rdf:ID="PaymentAuthPolicy1">
  <policy:controls rdf:ID="PaymentAuthAction"/>
  <policy:hasSiteOfEnforcement
    rdf:resource="#TargetSite"/>
  <policy:hasPriority>1</policy:hasPriority>
</policy:PosAuthorizationPolicy>
```

Semantic security policy languages (3/5)

- Rei

```
<constraint:SimpleConstraint
    rdf:ID="IsPayCustomer"
  constraint:subject="#RequesterVar"
  constraint:predicate="&example;member
    Of"
  constraint:object="&example;
    payCustomer"/>
<constraint:SimpleConstraint
    rdf:ID="IsPayServer"
  constraint:subject="#PayServerVar"
  constraint:predicate="&example;member
    Of"
  constraint:object="&example;
    payServer"/>

<constraint:And
    rdf:ID="ArePayCustomerAndPayServer"
  constraint:first="#IsPayCustomer"
  constraint:second="#IsPayServer"/>
<deontic:Permission
    rdf:ID="PayServerPermission">
  <deontic:actor rdf:resource="#RequesterVar"/>
  <deontic:action
    rdf:resource="&example;access"/>
  <deontic:constraint
          rdf:resource="#ArePayCustomerAndPay
    Server"/>
</deontic:Permission>
<policy:Policy rdf:ID="PaymentAuthPolicy1">
  <policy:grants
    rdf:resource="#PayServerPermission"/>
</policy:Policy>
```

Semantic security policy languages (4/5)

- SWRL

```
<ruleml:imp>
  <ruleml:_head>
    <swrlx:individualPropertyAtom
             swrlx:property="GrantedAccess">
      <ruleml:var>requester</ruleml:var>
      <ruleml:var>server</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head>
<ruleml:_body>
    <swrlx:classAtom>
      <owlx:Class owlx:name="User" />
      <ruleml:var>requester</ruleml:var>
    </swrlx:classAtom>
<swrlx:classAtom>
      <owlx:Class owlx:name="Server" />
      <ruleml:var>server</ruleml:var>
    </swrlx:classAtom>
    <swrlx:individualPropertyAtom
      swrlx:property="Member">
      <ruleml:var>requester</ruleml:var>
      <owlx:Individual owlx:name="#PayCustomer"/>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom
      swrlx:property="Member">
      <ruleml:var>server</ruleml:var>
      <owlx:Individual owlx:name="#PayServer" />
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
</ruleml:imp>
```

Semantic security policy languages (5/5)

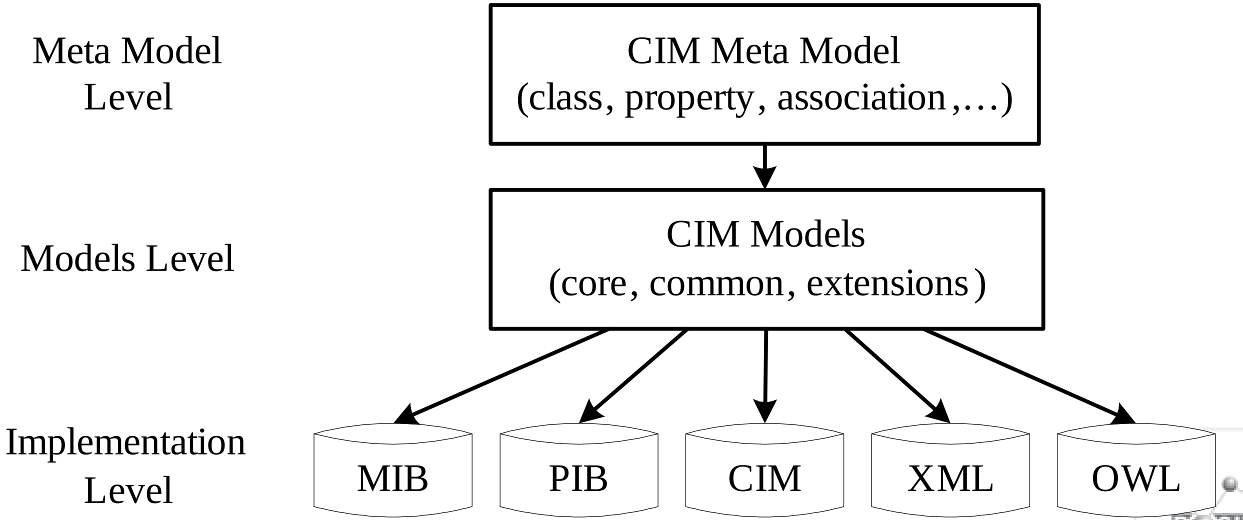
	KAoS	Rei	SWRL
Approach	Deontic Logic	Deontic Logic + Rules	Rules
Specification language	DAML/OWL	Prolog-like syntax + RDF-S	Prolog-like syntax + OWL
Tools for specification	KPAT	No	No
Reasoning	KAoS engine	Prolog engine	Prolog engine
Enforcement	Supported	External Functionality	External Functionality

Presentation Overview

- Introduction
- Requirements for a policy framework
- Advantages of semantic security policy framework
- Semantic security policy languages
- Using CIM Ontology with semantic languages
- Conclusions

Using CIM Ontology with semantic languages (1/5)

- The Common Information Model (CIM) is an approach from the DMTF provides a common definition of management-related information for systems, networks, users, and services.
  - Independent of any implementation or specification
  - It can be mapped to structured specifications such as OWL



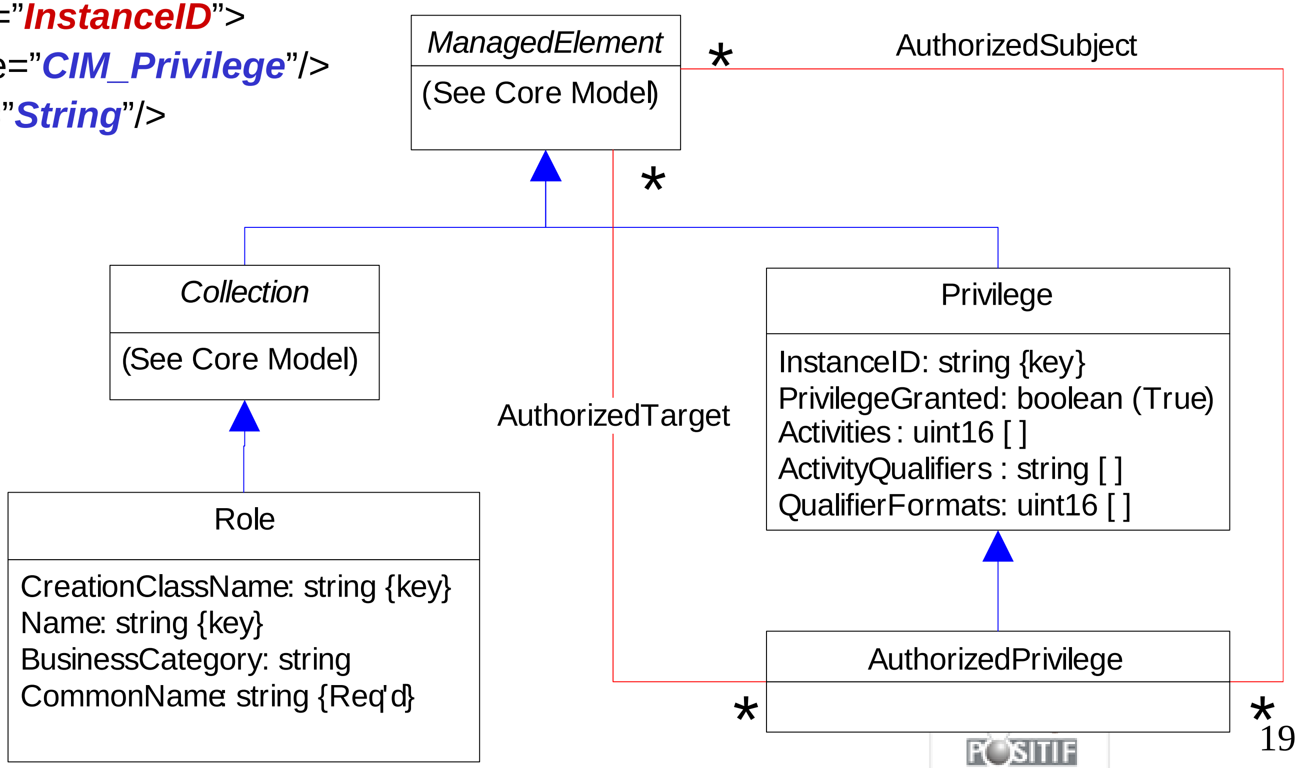
Using CIM Ontology with semantic languages (2/5)

- Main principles identified as part of the mapping of CIM into OWL:
  - Every CIM class generates a new OWL class using the tag <owl:Class>.
  - Every CIM generation (inheritance) is expressed using the tag <rdfs:subClassOf>.
  - Every CIM class attribute is specified using the tag <owl:DatatypeProperty> for literal values or <owl:ObjectProperty> as references to class instances.
  - Every CIM association is expressed as an OWL class with two <owl:ObjectProperty> where their identifiers (i.e., <rdf:ID>) are the names of the properties of the CIM association; this is the most suitable general-purpose mechanism currently available.

Using CIM Ontology with semantic languages (3/5)

```
<owl:Class rdf:ID="CIM_Privilege">
  <rdfs:subClassOf rdf:resource="CIM_ManagedElement"/>
</owl:Class>
<owl:Class rdf:ID="CIM_AuthorizedSubject">
    <rdfs:subClassOf rdf:resource="LogicalEntity"/>
</owl:Class>
<owl:DatatypeProperty rdf:ID="InstanceID">
    <rdfs:domain rdf:resource="CIM_Privilege"/>
    <rdfs:range rdf:resource="String"/>
</owl:DatatypeProperty>
```

```
<owl:ObjectProperty rdf:ID="Privilege">
    <rdfs:domain
    rdf:resource="CIM_AuthorizedSubject"/>
    <rdfs:range
    rdf:resource="CIM_ManagedElement"/>
</owl:ObjectProperty>
```



Using CIM Ontology with semantic languages (4/5)

- Note that the OWL representation of CIM can be used in semantic policy languages (e.g., SWRL).

```
<ruleml:imp>
 <ruleml:_body>
  <swrlx:classAtom>
   <owlx:Class owlx:name="CIM_Role"/>
   <ruleml:var>server</ruleml:var>
  </swrlx:classAtom>
  <swrlx:classAtom>
   <owlx:Class owlx:name="CIM_Role" />
   <ruleml:var>requester</ruleml:var>
  </swrlx:classAtom>
  <swrlx:classAtom>
   <owlx:Class
   owlx:name="CIM_AuthorizedPrivilege" />
   <ruleml:var>privilege</ruleml:var>
  </swrlx:classAtom>
```

```
 <swrlx:individualPropertyAtom
   swrlx:property="Name">
   <ruleml:var>server</ruleml:var>
   <owlx:Individual owlx:name="#PayServer" />
  </swrlx:individualPropertyAtom>
  <swrlx:individualPropertyAtom
   swrlx:property="Name">
   <ruleml:var>requester</ruleml:var>
   <owlx:Individual owlx:name="#PayCustomer" />
  </swrlx:individualPropertyAtom>
  <swrlx:individualPropertyAtom
   swrlx:property="Name">
   <ruleml:var>privilege</ruleml:var>
   <owlx:Individual
   owlx:name="#GrantedAccess" />
  </swrlx:individualPropertyAtom>
 </ruleml:_body>
```

Using CIM Ontology with semantic languages (5/5)

- Note that the OWL representation of CIM can be used in semantic policy languages (e.g., SWRL).

```
<ruleml:_head>
  <swrlx:classAtom>
    <owlx:Class
    owlx:name="CIM_AuthorizedTarget" />
    <ruleml:var>authtarget</ruleml:var>
  </swrlx:classAtom>
  <swrlx:classAtom>
    <owlx:Class
    owlx:name="CIM_AuthorizedSubject" />
    <ruleml:var>authsubject</ruleml:var>
  </swrlx:classAtom>
  <swrlx:individualPropertyAtom
    swrlx:property="Privilege">
    <ruleml:var>authtarget</ruleml:var>
    <ruleml:var>privilege</ruleml:var>
  </swrlx:individualPropertyAtom>
  <swrlx:individualPropertyAtom
    swrlx:property="TargetElement">
    <ruleml:var>authtarget</ruleml:var>
    <ruleml:var>server</ruleml:var>
  </swrlx:individualPropertyAtom>
  <swrlx:individualPropertyAtom
    swrlx:property="Privilege">
    <ruleml:var>authsubject</ruleml:var>
    <ruleml:var>privilege</ruleml:var>
  </swrlx:individualPropertyAtom>
  <swrlx:individualPropertyAtom
    swrlx:property="PrivilegedElement">
    <ruleml:var>authsubject</ruleml:var>
    <ruleml:var>requester</ruleml:var>
  </swrlx:individualPropertyAtom>
</ruleml:_head>
</ruleml:imp>
```

Presentation Overview

- Introduction
- Requirements for a policy framework
- Advantages of semantic security policy framework
- Semantic security policy languages
- Using CIM Ontology with semantic languages
- Conclusions

Conclusions

- Semantic languages facilitates the policy management (reasoning, interoperation, …)
- KAoS presents a full solution that includes from the policy language to the policy enforcement, while the rest of approaches lacks some components
- SWRL is not limited to deontic policies as it happens in Rei and KAoS
- The mapping of CIM to a valid representation for WIS is beneficial, since it permits to model components using the DMTF methodology and hence obtain a standard and interoperable representation of it

Acknowledgements

This work has been partially funded by the EU POSITIF (Policy-based Security Tools and Framework) IST project (IST-2002-002314)

http://www.positif.org/