An access control model for distributed geospatial information objects

8.6.2004

Andreas Matheus

Technische Universität München

Content of this presentation

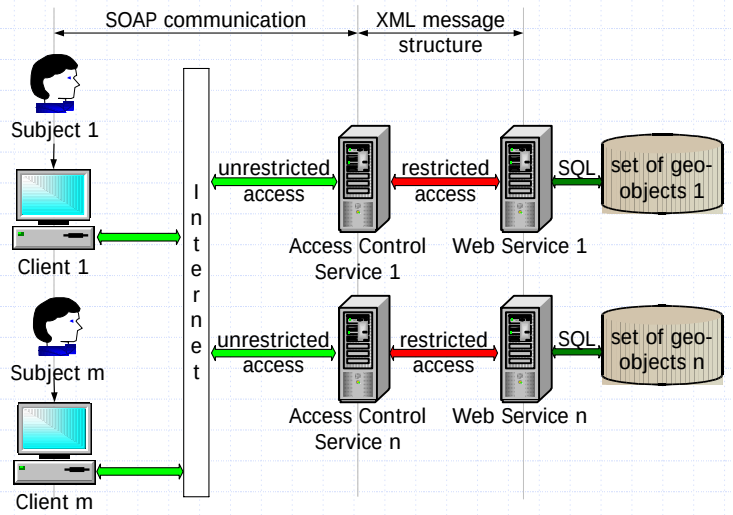
- How to declare access restrictions for distributed geospatial information objects
- Geospatial objects can be accessed via Web Services
  - Service input and output is XML (GML) document
- Use XACML-like policy model for declaring the access restrictions
  - Based on the object structure in an XML document
  - Based on their geospatial characteristics Allow positive and negative access
- Detect and resolve policy inconsistencies

Existing standards

- XACL: Declare access restrictions for XML documents
  - No spatial restrictions
  - Not for distributed access control
- DRM: Declare usage rights for existing content
- XACML: Declare XML encoded access restrictions
  - No spatial restrictions
  - Can be used for distributed access control

Aim: Develop an access control model that extends XACML capabilities

The environment



A geo-spatial information object

... (geo-object) is an information entity about a real world phenomenon with spatial characteristics, such as

- a location relative to the Earth
- geometry that is expressed by coordinates
- unique object identity, which is machine readable
- object type
- optionally non-spatial attributes, which makes it distinguishable from other (geo)object
- optionally a (spatial) relation to another geo-objects

Accessing distributed geo-objects

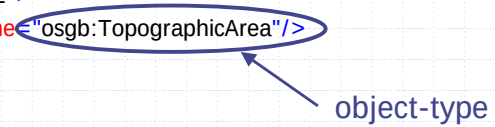
- Distributed geo-objects are stored in different Databases, accessible thru Web Services (WSs)
- Each Web Service provides operations that allow accessing the geo-objects
- Access has multiple aspects
  - Network based communication with the Web Service
  - Requires the authorization to invoke operations of the Web Service
  - Requires the authorization to use (read, write, delete or create) geo-objects, stored inside a MS
  - Select geo-objects by their characteristics; spatial and non-spatial characteristics

The responsibility of the Web Service

- Provides functions that can be used from a client to process the stored geo-objects
- Each function transforms the stored set of geo-objects into a structured set of geo-objects (service output)
- Input parameters (service input) may control
  - the structure,
  - the content and other
  - miscellaneous characteristics of the service output
- Service input and output is valid XML, respectively GML

WFS input example

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- http://vega.galdosinc.com:8880/wfs/http -->
<wfs:GetFeature service="WFS" version="1.0.0"
   xmlns:wfs="http://www.opengis.net/wfs"
   xmlns:gml="http://www.opengis.net/gml"
   xmlns:osgb="http://www.ordnancesurvey.co.uk/xml/namespaces/osgb"
   outputFormat="GML2">
   <wfs:Query typeName="osgb:TopographicArea"/>
</wfs:GetFeature>
```



WFS output example

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:FeatureCollection ...>
  <gml:boundedBy><gml:Box srsName="osgb:BNG">
    <gml:coordinates>278499.45,187424.05
    278563.25,187581.05</gml:coordinates>
  </gml:Box></gml:boundedBy>
  <gml:featureMember>
    <osgb:TopographicArea fid="osgb1000000756121907">
      <osgb:featureCode>10096</osgb:featureCode>
      <osgb:version>1</osgb:version>
      <osgb:versionDate>2001-11-07</osgb:versionDate>
      <osgb:theme>Land</osgb:theme>
      <osgb:calculatedAreaValue>50.956252</osgb:calculatedAreaValue>
      ....
```

object-type

object-identity

WFS output example (contd.)

```
      ...
      <osgb:descriptiveGroup>Landform</osgb:descriptiveGroup>
      <osgb:polygon><gml:Polygon srsName="osgb:BNG">
      <gml:outerBoundaryIs><gml:LinearRing srsName="osgb:BNG">
      <gml:coordinates> 278534.100,187424.700 278529.250,187430.900
      278528.700,187431.650 278527.250,187433.600 278525.800,187435.600
      278525.600,187435.900 278524.250,187437.950 278517.750,187448.200
      278515.850,187446.000 278517.500,187443.500 278521.800,187438.200
      278525.450,187433.450 278531.300,187426.350 278533.250,187424.050
      278534.100,187424.700 </gml:coordinates>
      </gml:LinearRing></gml:outerBoundaryIs>
      </gml:Polygon></osgb:polygon>
    </osgb:TopographicArea>
  </gml:featureMember>
</wfs:FeatureCollection>
```

object-geometry

Output schema for GetFeature request

```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="unqualified" elementFormDefault="qualified"
    targetNamespace="http://www.ordnancesurvey.co.uk/xml/namespaces/os
    gb" version="1.0" xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:osgb="http://www.ordnancesurvey.co.uk/xml/namespaces/osgb"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <!-- features -->
    ...
    <element name="TopographicLine"
    substitutionGroup="osgb:_TopographicFeature"
    type="osgb:TopographicLineType"/>
    <element name="TopographicArea"
    substitutionGroup="osgb:_TopographicFeature"
    type="osgb:TopographicAreaType"/>
    ...
</schema>
```

object-type

Distributed access control requirements

- Access control is enforced locally at each service that provides access to geo-objects
  - Local identity for objects is sufficient
  - Disadvantage: Policy inconsistencies can only be solved locally
- Access control is enforced at domain level
  - Identity of objects must be unique within the domain
  - Advantage: Policy inconsistencies can be solved for all objects in the domain
- Scalability due to large number of subjects and objects

Possible access restrictions (1/2)

- Based on the capabilities of the Web Service
  - Format, style and scale of output (e.g. maps); determine the reusability of the output
  - At what time is the function used (e.g. inside business hours, not on Saturday or Sunday)
- Based on client side characteristics
  - From what client is the service used (e.g. a machine inside or outside of the local area network)
  - What application is used to invoke the service
- Based on the communication
  - Is the service input or output confidential, digitally signed or guaranteed for integrity

Possible access restrictions (2/2)

- Based on the object
  - The type of the object (e.g. building, street)
  - The object instance (e.g. “The White House”)
- Based on the geometry of the object
  - The location of the geo-object (e.g. inside the boundary of “Munich”)
  - The spatial relation to other geo-objects (e.g. land parcel A is adjacent to land parcel B)
- Based on the subject authentication process
  - The subject must be identified by a certificate that is released by a trusted CA

Here: Focus on object and geometry

Authorization on object level

- Pre-assumptions
  - Unique identity of a subject
- Possible operations
  - Write (W), Read (R), Create (C) and Delete (D)
- Positive and negative access modes
  - {W+,W-,ε}, {R+,R-,ε}, {C+,C-,ε}, {D+,D-,ε}
  - “ε” means no explicit access mode
- Access restrictions can apply to
  - one or multiple types/instances of objects
  - a region (2D geometry)

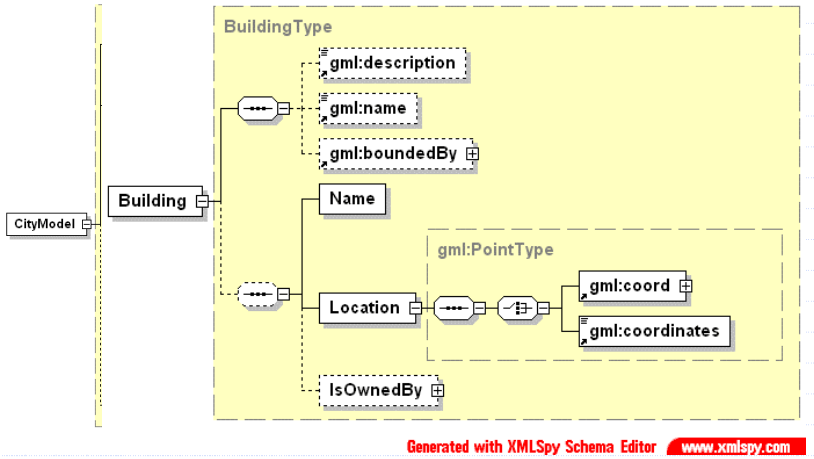
Policy based access control (XACML)

- Policy = <Subject, AccessMode, Object, R*>
- Rule is a 4-valued tuple
  - R = < Subject, Operation, Object, Condition>
  - R -> {grant, deny}, if R evaluates to ‘true’
    R -> {N/A}, if R evaluates to ‘false’
    R -> {indeterminate}, for all other cases
- Example
  - <Joe, W+, ABC, (T>8:00 && T<16:00)>
  - Declares that the subject identified as “Joe” can use the Write operation on the object, identified as “ABC”, if the time is between 8 a.m and 4 p.m.

Pre-requisite for object based policies

- The objects exist in a XML structured document that can be validated against the associated XML Schema (XSD)
- Geo-objects are structured in a GML document
- The schema declares
  - the types of objects
  - characteristically properties hold
    - geometry
    - non-spatial information
  - relation properties
  - attributes that identify the object

Example XSD based on GML



Example GML document

```
<?xml version="1.0" encoding="UTF-8"?>
<CityModel ... xsi:schemaLocation="http://www.tum.de/xyz CityModel.xsd" fid="CityModel">
 <gml:boundedBy>
  <gml:Box gid="box1" srsName="foo">
   <gml:coord><gml:X>0</gml:X><gml:Y>0</gml:Y></gml:coord>
   <gml:coord><gml:X>4</gml:X><gml:Y>4</gml:Y></gml:coord>
  </gml:Box>
 </gml:boundedBy>
 <gml:featureMember>
  <Building xsi:type="BuildingType" fid="B1">
   <Name>Building A</Name>
   <Location srsName="foo">
    <gml:coord><gml:X>1</gml:X><gml:Y>0</gml:Y></gml:coord>
   </Location>
   <IsOwnedBy xlink:arcrole="simple" xlink:href="#O1"/>
  </Building>
 </gml:featureMember>
 <gml:featureMember>
  <Person fid="P1">
   <Name>Max Mustermann</Name>
   <Owns xlink:arcrole="simple" xlink:href="#B1"/>
  </Person>
 </gml:featureMember>
 <Name>An example city model</Name>
</CityModel>
```

geo-objects (spatial features)

non geo-object (feature)

Apply a policy to the object type

- Each XML document element of a particular type can be identified by
  - Use name of the element in the XML document
    - element(Building,*)
  - Use type name used in the XSD
    - element(*,BuildingType)
- The Xpath expressions are only valid, if a particular namespace is used
  - namespace-uri(CityModel)=http://www.tum.de/xyz

Example for object type restrictions

- Joe has access to all objects of type “BuildingType” using mode Write and Read.
  - R1 = <Joe, W+, element(*,BuildingType) , >
  - R2 = <Joe, R+, element(*,BuildingType), >
- Notice that nothing is stated about the Create and Delete operations; this means implicit rules exist that use ε access mode
  - R’ = <Joe, Cε, element(*,BuildingType), >
  - R’’ = <Joe, Dε, element(*,BuildingType), >
- Notice that no constraint is present

Apply a policy to the object type

- Each element can be referenced using identifying characteristics
  - //*[@fid="B1"]: The ID (fid)
  - //*/Building/Name="Building A“: Identifying characteristics, e.g. name
- A set of elements can be referenced
  - //*: Refers to all elements of the document
  - //Building: Refers to all elements named “Building”
  - //Building/*: Refers to all next level descendant (child) elements named “Building”
  - //Building//*: Refers to all descendants of elements names “Building”

Example for object instance restrictions

- Joe has Write access to an object named “Building” which is characterized by the sub-element “Name” with the value “Building A”.
  - R1 = <Joe, W+, //Building[Name="Building A"]“, >
- Notice that nothing is stated about the Read, Create and Delete operations; this means implicit rules exist that use ε access mode
  - R’ = <Joe, Cε, //Building[Name="Building A"]“, >
  - R’’ = <Joe, Dε, //Building[Name="Building A"]“, >
  - R’’ = <Joe, Rε, //Building[Name="Building A"]“, >

Example for object instance restrictions

- Joe has Write access to all descendant objects for the element named “Building” which is characterized by the sub-element “Name” with the value “Building A”.
  - R1 = <Joe, W+, //Building[Name="Building A"]/*>
- Notice that nothing is stated about the Read, Create and Delete operations; this means implicit rules exist that use ε access mode
  - R’ = <Joe, Cε, //Building[Name="Building A"]/*>
  - R’’ = <Joe, Dε, //Building[Name="Building A"]/*>
  - R’’ = <Joe, Rε, //Building[Name="Building A"]/*>

The request

- The request can be represented by a 4-valued tuple <Subject_R, Operation_R, Object_R, Condition_R>
  - Operation_R is the intended operation (W, R, C, D)
  - Condition_R is a statement that gives the request context specific information; hence it defines restrictions
- Example request <Joe, W, //*, IP=12.15.1.0>
  - defines a write operation request by the subject Joe, where all objects are referenced; the client machine has the IP-address 12.15.1.0

When does a policy apply?

- Policy applies, when
  - Subject_R ∈ Subject_P
  - Operation_R ∈ Operation_P
  - Object_R ∈ Object_P
- Condition_R defines restrictions, such as
  - Client machine identity (IP or domain name)
  - Application identity
  - Timestamp, when request was made
  - Area of interest (for spatial policies)

Here: Focus on spatial restrictions

Spatial policy

- Declares access mode for a geospatial area
- Geospatial area is encoded using GML (Version 2.x)
- Object element refers to the object for which the access mode applies
- Rule element “Condition” expresses the geospatial condition that must be met
- Different spatial methods can be used to express the spatial relation
  - Equals, Disjoint, Intersects, Touches, Crosses, Within, Contains, Overlaps

Spatial policy example

- Joe may use the Read operation on all objects of the type “BuildingType” that reside inside the declared boundaries of the policy area named “MUC”

Policy A = <Joe, R+, element(*, BuildingType), Within(//Building/Location, //Polygon[@gid="MUC"])>

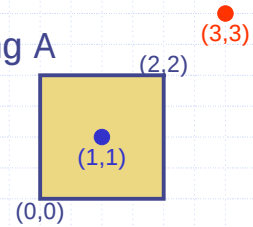
```
<Polygon gid="MUC" srsName="foo">
 <outerBoundaryIs><LinearRing>
    <coordinates>0,0 0,2 2,2 2,0 0,0</coordinates>
  </LinearRing></outerBoundaryIs>
</Polygon>
```

When does a spatial policy apply?

- In general, the policy applies, if
  - Subject_R ∈ Subject
  - Operation_R ∈ Operation
  - Object_R ∈ Object
- A spatial policy applies
  - if at least one geo-object's geometry intersects with the policy area (indirect applicable)
  - if the confinement contains the area of interest (direct applicable)

Spatial policy request example (indirect request)

- <Joe, W, //Building,>
- Results in a set of "Building" elements with location
  - Building A: Location (1,1)
  - Building B: Location (3,3)
- Policy applies because Building A is inside the policy area MUC

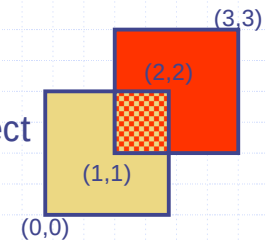


Spatial policy request example (direct request)

- <Joe, W, //Building, AreaOfInterst>

```
<Polygon gid="AreaOfInterest" srsName="foo">
 <outerBoundaryIs><LinearRing>
    <coordinates> 1,1 1,3 3,3 3,1 1,1 </coordinates>
  </LinearRing></outerBoundaryIs>
</Polygon>
```

- Policy applies, because AreaOfInterst and MUC intersect



Classifying policy inconsistencies

- At least two policies with inconsistent access modes apply
- Static inconsistencies can be detected without a particular request
  - P_1 = <Subject_A, op_A, Ob_A, Ø>
  - P_2 = <Subject_A, op_B, Ob_B, Ø>
  - <op_A,ob_B> = {(W+,W-)|(R+,R-)|(C+,C-)|(D+,D-)}[1)]
  - {Ob_A} ∩ {Ob_B} ≠ {∅}
- Dynamic inconsistencies may be detected by evaluating a given request

[1)] All possible combinations, resulting in inconsistencies must be stated

Policy inconsistencies are based on

- Object structure in the XML document
  - Type and instance references apply to the same objects
  - Different instance references apply to the same objects
- Object geometry makes at least two policies applicable
  - Either the geometry intersects with policy areas of at least two policies
  - Or the geometry is inside at least policy areas that overlap

Policy inconsistency examples

- Example 1: Type and instance
  - Policy A: Joe may write all objects (elements) of type “BuildingType”
  - Policy B: Joe may not write object of type “BuildingType” which is named “Building A” and all direct descendants (sub-elements)
- Example 2: Instance and instance
  - Policy A: Joe may write all descendants of object “Building”, named “Building A”
  - Policy B: Joe may not write property “Name” of the object, named “Building A”

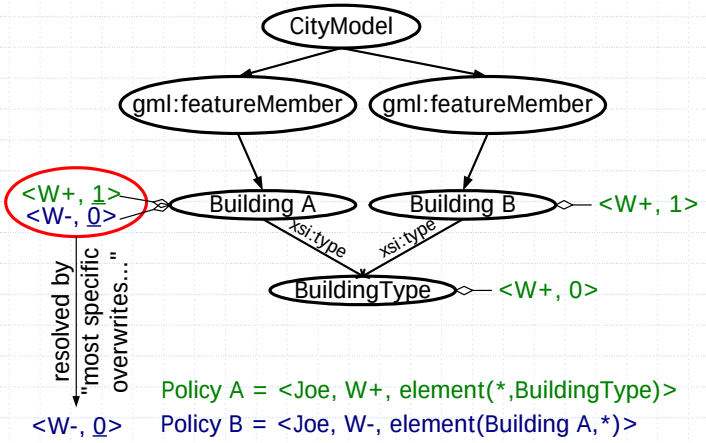
Representing policy inconsistencies

- Representation of an Xpath expression as a tree for each operation
  - Each node holds
    - a depth counter
    - access mode (+,-,ε)
- The depth counter is
  - 0 for the elements that are referenced directly by the Xpath (origin element)
  - ++1 for each element that is a descendant of the origin element; distance to the origin element

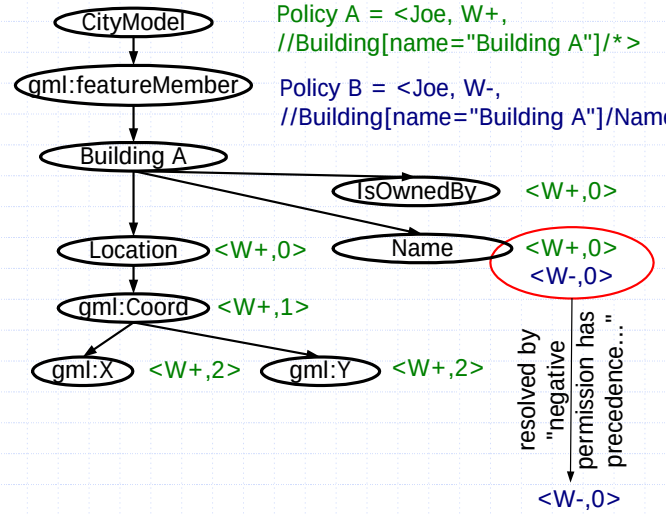
Resolving policy inconsistencies

- Resolving rule 1:
  “most specific overwrites less specific”
- Resolving rule 2:
  “negative permission has precedence over positive permission” if equal specific
  - use truth table to evolve negative result for combinations of (+,-), (+, ε), (-, ε)
    - (+, -) -> - (positive AND negative is negative)
    - (+, ε) -> - (positive AND nothing is negative)
    - (-, ε) -> - (negative AND nothing is negative)

Resolve inconsistency for example 1

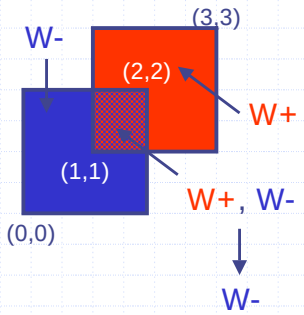


Resolve inconsistency for example 2



Spatial policy inconsistency example

- P_A = <Joe, W+, //*, Within(//Location,PolicyAreaA)>
- P_B = <Joe, W-, //*, Within(//Location,PolicyAreaB)>



- Static inconsistency can be detected by the overlapping of the policy areas
- Inconsistency can be resolved, using the truth table
  - (+,-) -> resolves to –
- So, for all geo-objects residing in the overlapping area the permission is W-

Resolve spatial policy inconsistencies

- Spatial policy inconsistencies are dynamic
  - Even the geometry of policy areas is known and
  - overlaps can be detected,
  - it is not known which geo-objects will be requested and what their geometry will be
- The rule “most specific overwrites” can only be used, if hierarchy for policy areas is known
  - This requires topology
  - Geometry is not sufficient
- Therefore –without topology-, only the rule “negative takes precedence” can be used

Summary access control

- The introduced access control model supports
  - positive and negative access modes on geo-objects
    - Object type based permissions
    - Object instance based permissions
  - spatial policies that declare
    - permissions for a geo-spatial area
    - permissions for geo-objects (and their properties), having particular geometry

The final slide

Thank you for your attention

Questions?