

# Data Model Transformation for Supporting Interoperability

Systems'

Composition and Interoperability:

A World in Transition.

ICCBSS 2007

Gorka Benguria, Xabier Larrucea

# Contents

- Interoperability Problem-a Dimension
- Basic Integration Requirements
- Requirements – Functional/non Functional
- Alternatives – XSLT/Semantic Mapping/Model Transformation
- Model Transformation - MTF/ATL/Java
- Summary of approaches for transformation
- Conclusion

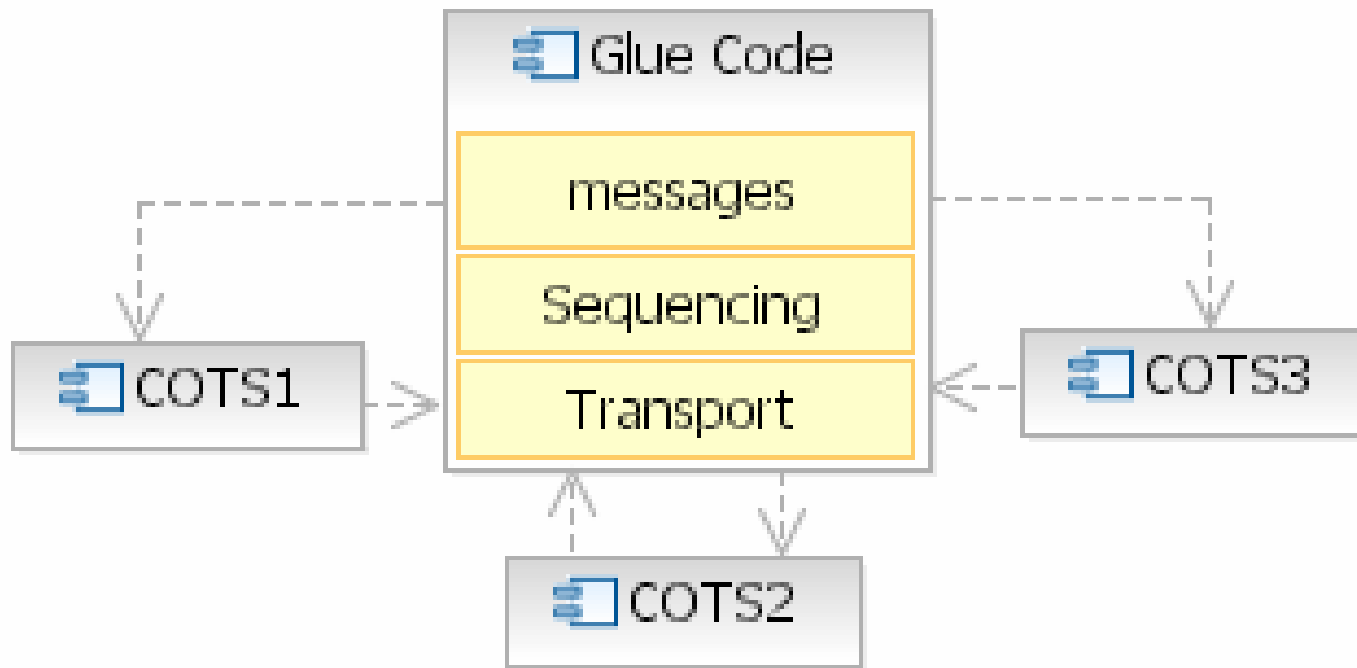
# Interoperability Problem-a Dimension

## Components Integration for Structured Information

*“although glue-code development usually accounts for less than half the total CBS software development effort, the effort per line of glue code averages about three times the effort per line of developed applications code”*



# Basic Integration Requirements



# Functional Requirements

- Changing element and attributes names
- Changing structure
- Multiplicity
- Keeping references
- Combining information
- Combining messages

# Non-Functional Requirements

- Fast
- Robust
- Traceable
- Reusable
- Accessible

# Alternatives

- XSLT
- Semantic Mapping
- Model transformation

# XSLT

```
<xsl:template match="person:person">
  <xsl:element name="client:client">
    <xsl:attribute name="id">
      <xsl:value-of select="./id"/>
    </xsl:attribute>
    <xsl:element name="name">
      <xsl:value-of select="./firstname"/>
    </xsl:element>
    <xsl:element name="surname">
      <xsl:value-of select="./familyname"/>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

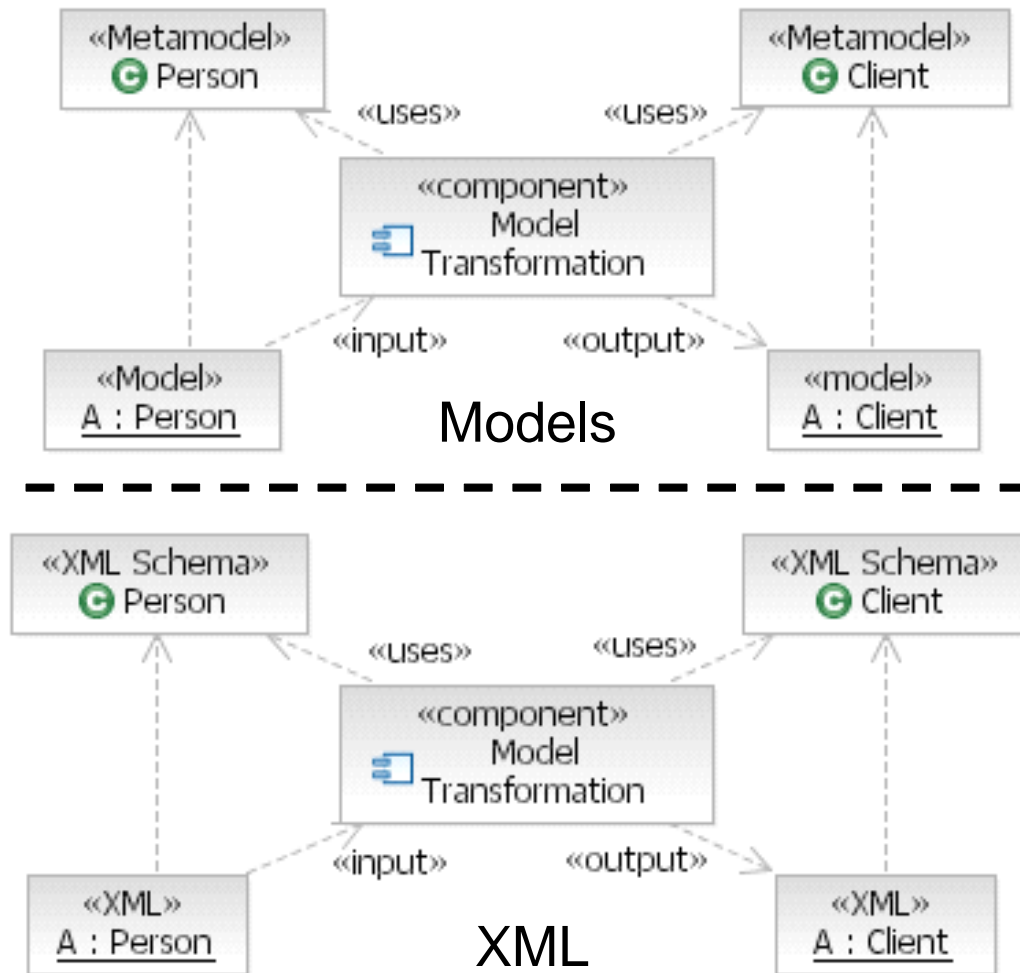
# Semantic Mapping

| Local Ontology (LO)   | Reference Ontology (RO)  |
|---|--|
| <u>Person</u> [<br>ID : <i>literal</i><br>firstName : <i>literal</i><br>familyName : <i>literal</i> ] | <u>Client (ID)</u> [<br>name : <i>literal</i><br>surName: <i>literal</i> ] |

## Semantic Annotation:

|   |
|---|
| <b>LO.Person.ID =: RO. Client.#ID</b>               |
| <b>LO.Person. firstName =: RO. Client.name</b>      |
| <b>LO.Person. familyName =: RO. Client. surName</b> |

# Model Transformation



# Comparison of alternatives

|                            | <b>XSLT</b>                        | <b>Sem. Map.</b> | <b>Mod. Trans.</b>     |
|----------------------------|------------------------------------|------------------|------------------------|
| <b>Uses</b>                | XML                                | Ontologies       | Models                 |
| <b>Trans.<br/>Language</b> | Standard                           | Ad-hoc           | Recent<br>standard QVT |
| <b>Trans.<br/>Engine</b>   | Integrated<br>with web<br>browsers | Ad-hoc           | Prototypes             |

# Sample Transformation

```
<?xml version="1.0" encoding="UTF-8" ?>
<person:person
xmlns:person="http://ICCBSS/person.xsd">
  <firstname>John</firstname>
  <familyName>Doe</familyName>
  <id>1234</id>
</person:person>
```



```
<?xml version="1.0" encoding="UTF-8" ?>
<client:client xmlns:client="http://ICCBSS/client.xsd"
id="1234">
  <name>John</name>
  <surname>Doe</surname>
</client:client>
```

# Sample transformation XML Schemas

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:this="http://ICCBSS/person.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://ICCBSS/person.xsd">
  <xsd:element name="person" type="this:PersonType" />
  <xsd:complexType name="PersonType">
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string" />
      <xsd:element name="familyName" type="xsd:string" />
      <xsd:element name="id" type="xsd:integer" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```



```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:this="http://ICCBSS/client.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://ICCBSS/client.xsd">
  <xsd:element name="client" type="this:ClientType" />
  <xsd:complexType name="ClientType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="surname" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:integer" />
  </xsd:complexType>
</xsd:schema>
```

# MTF rule code example

```
relate personDrt2clientDrt
  (person:DocumentRoot personDrt,
   client:DocumentRoot clientDrt)
{
  person2client (personDrt.person,
                 clientDrt.client)
}

relate person2client(person:PersonType person,
                      client:ClientType client)
{
  equals (person.id,client.id),
  equals (person.firstname,client.name),
  equals (person.familyName,client.surname)
}
```

# ATL code

```
rule Person2Client {  
from  
    person : person!person  
to  
    client : client!client (  
        ID <- person.ID,  
        name <- person.firstname,  
        surname <- person.familyName  
    )  
}
```

# Java code for data reconciliation

```
private void relatePersonDrt2ClientDrt
    (iccbss.person.DocumentRoot personDrt,
     iccbss.client.DocumentRoot clientDrt)
{
    //this is for GETTING the person
    PersonType person = personDrt.getPerson();

    //this is for CREATING the client
    ClientType client;
    client = ClientFactory.eINSTANCE.
        createClientType();
    clientDrt.setClient(client)

    relatePerson2Client(person,client);
}
```

```
private void relatePerson2Client
    (PersonType person,
     ClientType client)
{
    client.setId(person.getId());
    client.setName(person.getFirstname());
    client.setSurname(person.getFamilyName());
}
```

# Summary of approaches for transformation

|                              | <b>MTF</b> | <b>ATL</b> | <b>Coding</b>         |
|------------------------------|------------|------------|-----------------------|
| <b>Changing names</b>        | Yes        | Yes        | Yes                   |
| <b>Changing structure</b>    | Not Always | Not Always | Yes                   |
| <b>Multiplicity</b>          | Not Always | Not Always | Yes                   |
| <b>Keeping references</b>    | Not Always | Not Always | Yes                   |
| <b>Combining information</b> | Not Always | Not Always | Yes                   |
| <b>Combining messages</b>    | No         | No         | Yes                   |
| <b>Fast</b>                  | Not Always | Not Always | Depends on programmer |
| <b>Robust</b>                | Yes        | Yes        | Depends on programmer |
| <b>Traceable</b>             | Yes        | Yes        | Depends on programmer |
| <b>Reusable</b>              | Yes        | Yes        | Depends on programmer |
| <b>Accessible</b>            | Feasible   | Feasible   | Difficult             |

# Conclusion

- The development environments are taking advantage of the data definition
- There are languages that focus in the transformation hiding lot of complexity (error management, types management,...) but they are not still capable of dealing with complex transformations with acceptable performance
- For code development there are lot of good practices that greatly depends on the experience of the programmer
- The platforms and development environments are continuously evolving to address more and more interoperability issues, but not everything can be solved at technical level.

**Gorka Benguria Elguezabal**  
R&D Projects Area  
[gorka.benguria@esi.es](mailto:gorka.benguria@esi.es)

Parque Tecnológico, # 204  
E-48170 Zamudio  
Bizkaia (Spain)  
Tel.: +34 94 420 95 19  
Fax: +34 94 420 94 20  
[www.esi.es](http://www.esi.es)