Model-driven Rule-based Mediation in XML Data Exchange

Yongxin Liao, Dumitru Roman, Arne J. Berre
SINTEF ICT, Oslo, Norway

October 5, 2010

Outline

- Intro to XML Data Exchange
- FloraMap: Flora2-based XML data transformation
  - Technique: steps and examples
  - Implementation and experiments
- Generic XML Data Exchange Framework
- Related Work
- Conclusions and Future Work

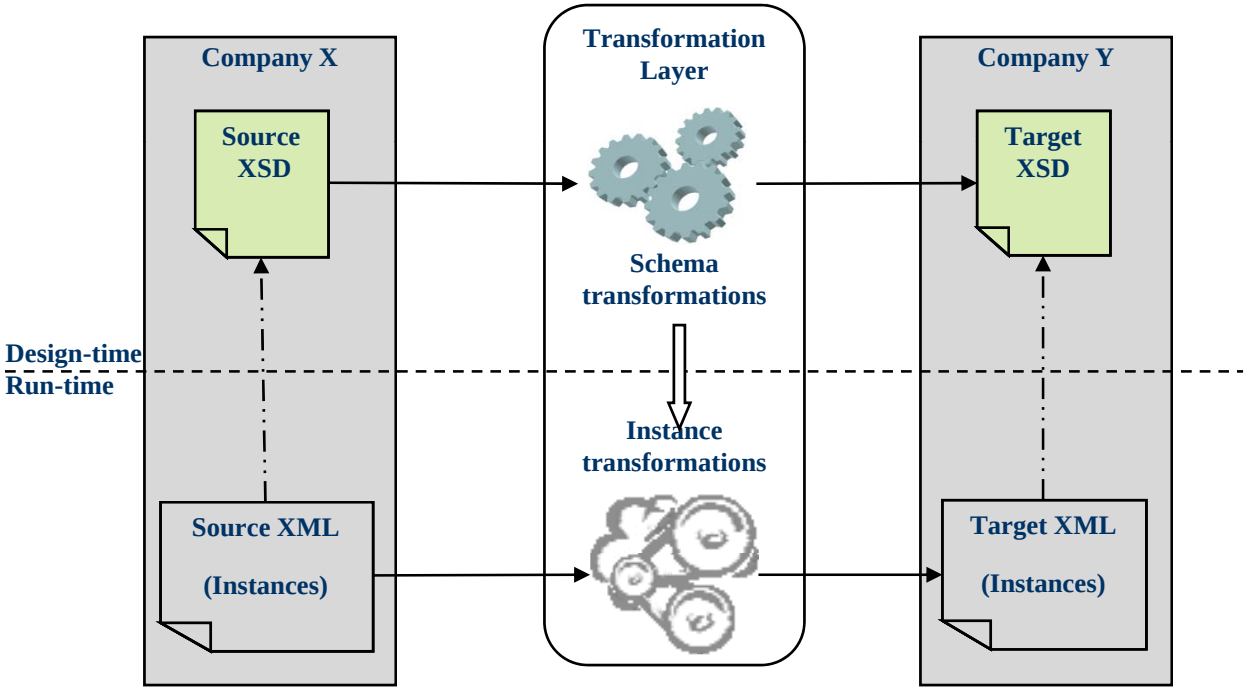
Outline

- Intro to XML Data Exchange
- FloraMap: Flora2-based XML data transformation
  - Technique: steps and examples
  - Implementation and experiments
- Generic XML Data Exchange Framework
- Related Work
- Conclusions and Future Work

XML Data Exchange

- Ubiquitous in B2B collaborations
- Need for agile interoperability and scalability in B2B collaborations
  - Automate as much as possible XML data exchange between enterprise systems
- XML data cannot be directly and fully automatically exchanged between B2B systems
  - Lack of standardized XML canonical models or schemas
  - Semantic differences and inconsistencies between conceptual models
- Option: provide techniques and tools to support humans in reconciling the differences and inconsistencies between the data models of the parties involved in a data exchange

Generic XML Data Exchange



Overall Approach

- Overall Approach
  - A lifting mechanism of XML schemas and instances to an object-oriented model
  - Use an object oriented rule language/engine as the underlying mechanism for providing an abstract, object-oriented model of XML schemas and instances, as well as for specification and execution of the mappings at the model level
- Benefits:
  - Allow the mappings creator to focus on the semantic, object-oriented model behind the XSD schemas and specify the mappings at a more abstract, semantic level
  - Allow both specification and execution of data mappings (i.e. design- and run-time mapping) in a single, unifying framework

Outline

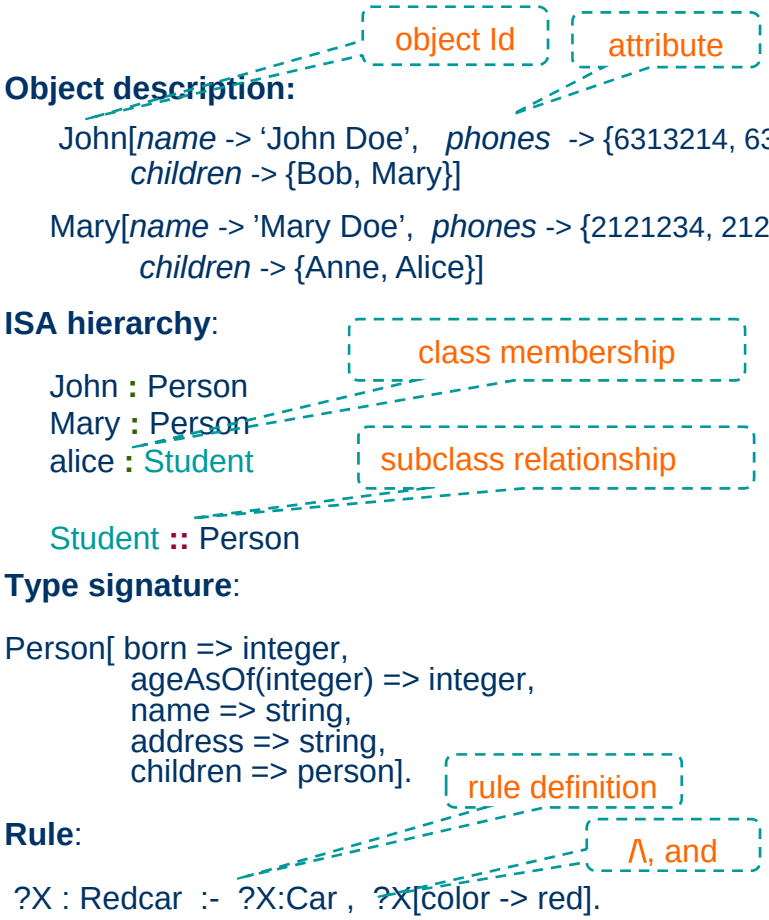
- Intro to XML Data Exchange
- FloraMap: Flora2-based XML data transformation
  - Technique: steps and examples
  - Implementation and experiments
- Generic XML Data Exchange Framework
- Related Work
- Conclusions and Future Work

Why Flora2?

http://flora.sourceforge.net/

- Flora2
  - Declarative, object-oriented knowledge base language
  - Open source programming environment
- Features:
  - Formal logical foundations, semantics
  - Frame-based
  - Rule-based
  - Higher-order
  - Declarative treatment of actions, database dynamics
  - Powerful introspection: querying schema, rules
  - Powerful meta-programming facilities
  - Typing, modularization
  - Powerful compositional aggregation

Flora2 – Simple examples



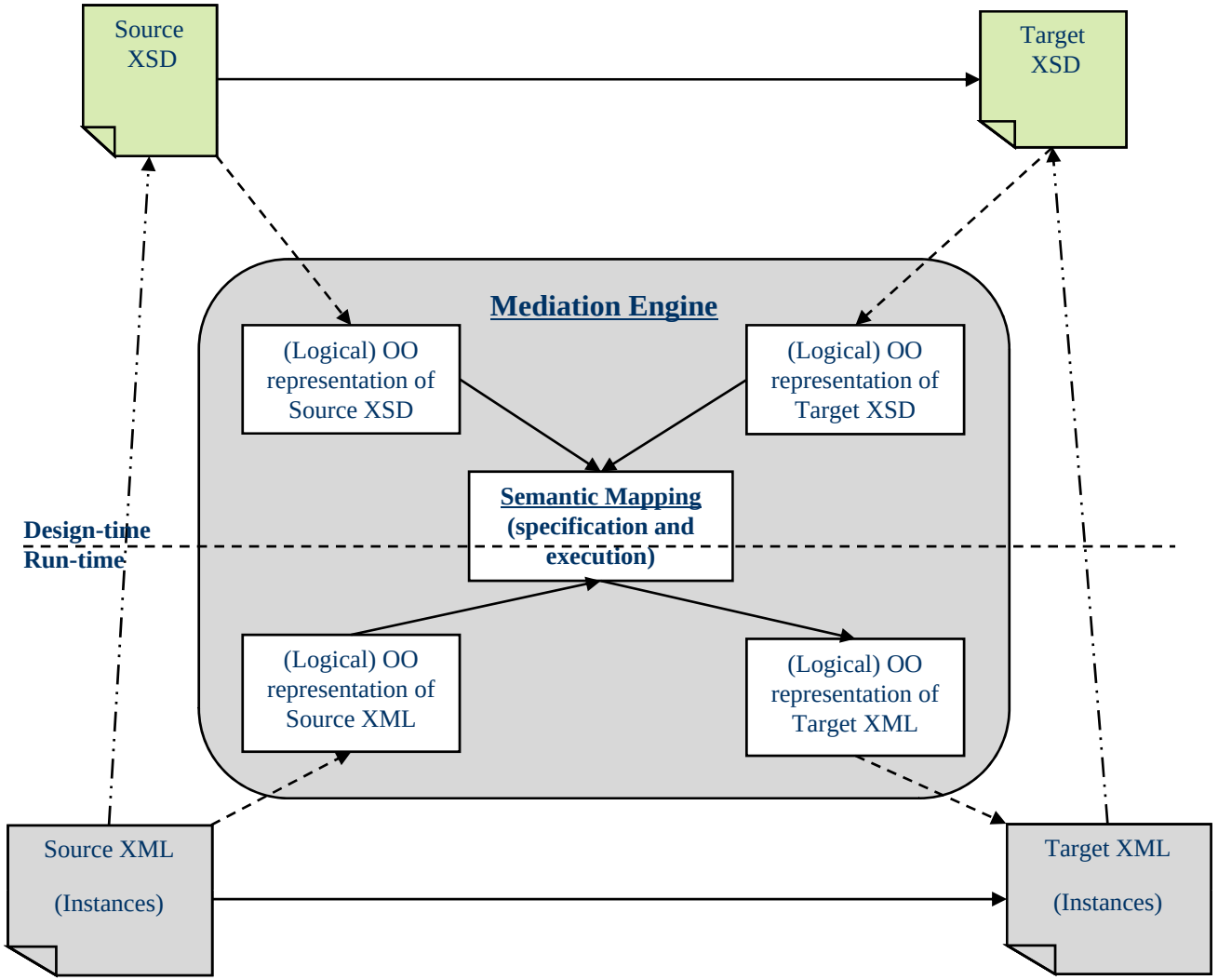
Query: *John's children who were born when he was 30+ years old:*

?– John[*born* -> ?Y, *children* -> ?C], C[*born* -> ?B], ?B > ?Y+30.

Syntax basics:

- Object ids:
  - Terms like in Prolog – John, abc, f(john,34), Car(red,20000)
- IsA hierarchy:
  - O:C – object O is a ***member*** of class C
  - C::S – C is a ***subclass*** of S
- Structure (*object-atoms*):
  - O [*Method* –> Value] – invocation of method
- Type (*signature-atoms*):
  - Class [*Method* => Class] – a method signature
- Combinations of the above:
  - \/, /\, negation, quantifiers

FloraMap – Semantic-based transformation of XML data



Source XSD: Company X

```
<element name="InvoiceCompanyX">
 <complexType>
  <sequence>
    <element name="Bizszam" type="xs:string“/>
    <element name="Ev" type="xs:string“/>
    <element name="Kanyvho" type="xs:string“/>
    <element name="Bizkelt" type="xs:string“/>
    <element name="city" type="string" minOccurs="0"/>
    <element name="zip" type="int" minOccurs="0"/>
    <element name="street" type="string"
                                  minOccurs="0"/>
  </sequence>
 </complexType>
</element>
```

Target XSD: Company Y

```
<element name="InvoiceCompanyY">
 <complexType>
  <sequence>
    <element name="InvoiceNumber" type="string"/>
    <element name="AccDate" type="string"/>
    <element name="InvoiceDate" type="string"/>
    <element name="DeliveryAddress" minOccurs="0">
      <complexType>
        <sequence>
          <element name="city" type="string" minOccurs="0"/>
          <element name="zip" type="string" minOccurs="0"/>
          <element name="DoorNo" type="string"
                                          minOccurs="0"/>
          <element name="street" type="string" minOccurs="0"/>
        </sequence>
      </complexType>
    </element>
  </sequence>
 </complexType>
</element>
```

Mapping rules

InvoiceCompanyX in source is mapped to *InvoiceCompanyY* in target:

1. *Bizszam* in source is the same as *InvoiceNumber* in target

2. *Bizkelt* in source is the same as *InvoiceDate* in target

3. *city* in source is the same as *DeliveryAddress.city* in target

4. *zip* in source is the same as *DeliveryAddress.zip* in target

5. *street* in source is the same as *DeliveryAddress.street* in target

6. *AccDate* in target is a concatenation of *Ev* in the source, a delimiter, *Kanyvho* in the source, a delimiter, and the string ’01’, i.e. *AccDate* = (*Ev*+‘_’+*Kanyvho*+‘_’+’01’)

Example – XML Data Exchange

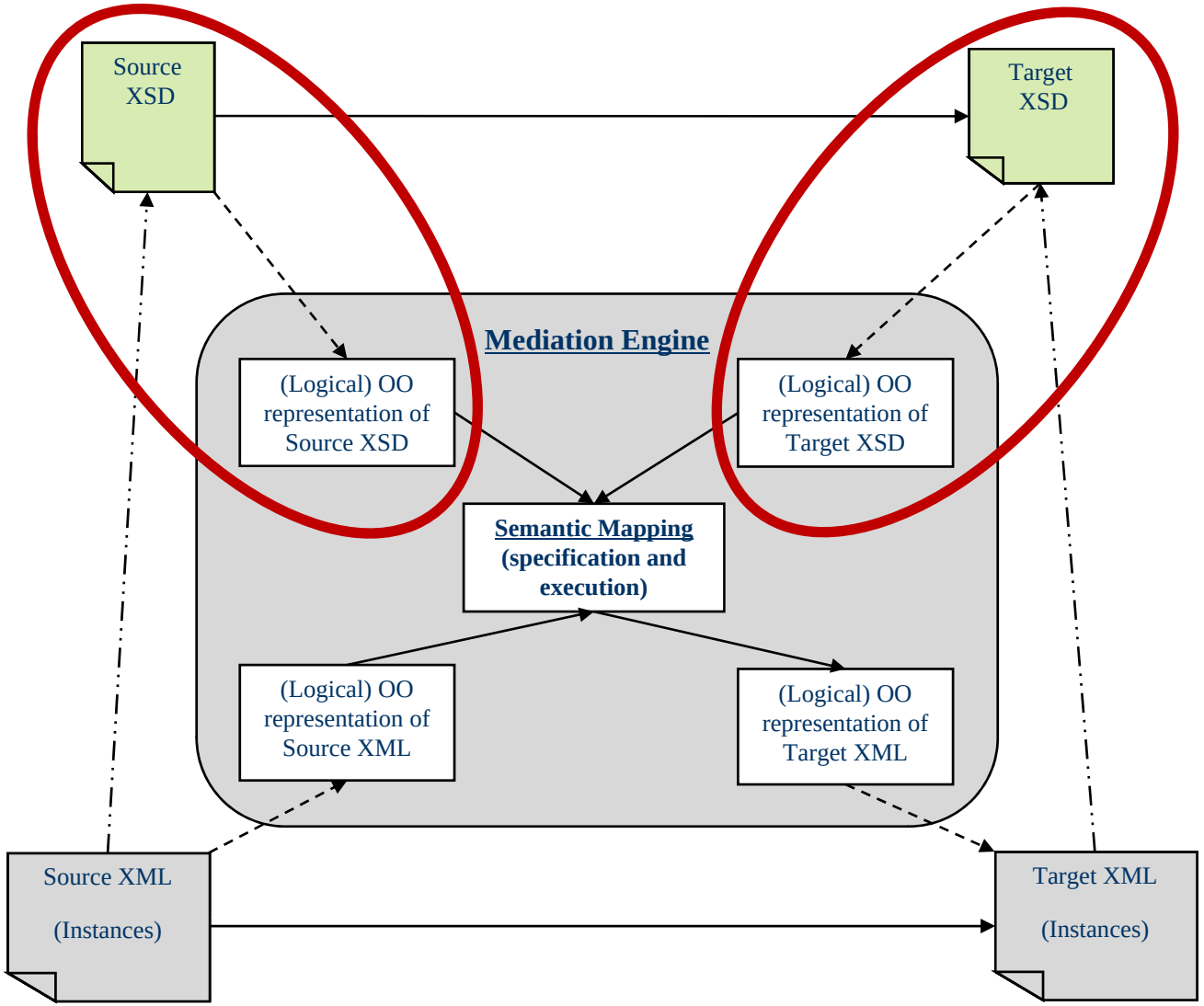
Source XML: Company X

```
<InvoiceCompanyX>
  <Bizszam>I_001</Bizszam>
  <Ev>2010</Ev>
  <Kanyvho>05</Kanyvho>
  <Bizkelt>2010-05-18</Bizkelt>
  <city>Oslo</city>
  <zip>1234</zip>
  <street>First Street</street>
</InvoiceCompanyX>
```

Target XML: Company Y

```
<?xml version="1.0"?>
< InvoiceCompanyY >
   <InvoiceNumber>I_001</InvoiceNumber>
   <AccDate>2010_05_01</AccDate>
    <InvoiceDate>2010-05-18</InvoiceDate>
   <DeliveryAddress>
      <city>Oslo</city>
      <zip>1234</zip>
      <DoorNo> </DoorNo>
      <street>First Street </street>
    </DeliveryAddress>
</ InvoiceCompanyY>
```

XSD to Flora2



XSD to Flora2 mapping

Situation	XSD	Flora2 Abstract (the "clean" conceptual model of the schema, i.e. no XML/XSD assumptions)	Flora2 Special (XML/XSD specific information, needed for generating structure of target instances)
Top-level Element with BaseType	<Element name="name" type="string"/>	name[name {1:1} *=>string].	
Top-level Element with ComplexType	<Element name="name"> <ComplexType> <Sequence> <Element name="firstname" type="string"/> <Element name="lastname" type="string"/> <Sequence> </ComplexType> </Element>	name[firstname {1:1} *=>string]. name[lastname {1:1} *=> string].	Elements[name->firstname]. Elements[name->lastname]. Sequences[name->[firstname,lastname]].
Top-level Element with SimpleType	<Attribute name="age"> <SimpleType> <restriction base="int"> <maxInclusive value="200"/> </restriction> </SimpleType> </Attribute >	age[base {1:1} *=>int]. age[maxInclusive->'200'].	
Top-level Attribute with BaseType	<Attribute name="age" type="integer"/>	age[age {1:1} *=>integer].	
Top-level Attribute with SimpleType	<Attribute name="age"> <SimpleType> <restriction base="int"> <maxInclusive value="200"/> <minInclusive value="0"/> </restriction> </SimpleType> </Attribute >	age[base {1:1} *=>int]. age[maxInclusive->'200']. age[minInclusive->'0'].	

XSD to Flora2 mapping (cont')

Situation	XSD	Flora2 Abstract	Flora2 Special
Top-level ComplexType	<ComplexType type="nameType"> <Sequence> <Element name="firstname" type="string"> <Element name="lastname" type="string"> <Sequence> </ComplexType>	nameType[firstname {1:1} *=>string]. nameType[lastname {1:1} *=>string].	Elements[nameType->firstname]. Elements[nameType->lastname]. Sequences[nameType->[firstname, lastname]].
Top-level SimpleType	<SimpleType name='nameType'> <restriction base="string"> <enumeration value="A"> <enumeration value="B"> </restriction> </SimpleType>	nameType[base {1:1} *=>string]. nameType[Enumeration->'A']. nameType[Enumeration->'B'].	
Top-level Group	<group name="nameGroup"> <sequence> <element name="firstname" type="string"/> <element ref="lastname"/> </sequence> </group>	nameGroup[firstname {1:1} *=>string]. nameGroup[lastname*=> lastname].	Groups[nameGroup->nameGroup] Elements[nameGroup->fristname]. Elements[nameGroup->lastname]. Sequences[nameGroup ->[firstname,lastname]].
Top-level AttributeGroup	<attributeGroup name="IdentifyGroup"> <attribute name="job" type="string"/> <attribute ref="title"/> </attributeGroup >	IdentifyGroup [job {1:1} *=>string]. IdentifyGroup [title {1:1} *=>title].	attributeGroups[IdentifyGroup -> IdentifyGroup]. Attributes[IdentifyGroup -> job]. Attributes[IdentifyGroup -> title].
Include	<include schemaLocation="person.xsd"/>	#include "path/person_ Abstract.flr"	#include "path/person_Special.flr"

XSD to Flora2 mapping (cont')

Situation	XSD	Flora2 Abstract	Flora2 Special
Sequence ,Choice, All	<Element name="name"> <ComplexType> <Sequence> <Element name="firstname" type="string"> <Choice> <Element name="title" type="string"> <Element name="job" type="string"> </Choice> <Sequence> </ComplexType> </Element>	name[firstname {1:1} *=> string]. name[title {1:1} *=> string]. name[job {1:1} *=> string].	Elements[name->firstname]. Elements[name->title]. Elements[name->job]. Sequences[name->[firstname, Sub_name3]]. Choices[Sub_name3->[title,job]].
Extension (simplecontent, complexcontent)	<ComplexType name="nameType"> <simpleContent> <extension base="PersonNameType"> <attribute name="title" type="string"/> </extension> </simpleContent> </ComplexType>	name :: PersonNameType. nameType [title {1:1} *=> string].	Attribute[nameType->title].

XSD to Flora2 mapping (cont')

- XSD import to Flora2 import
  - Import namespace in XSD
    - ['filename_Abstract.flr'>>namespace] in Flora2 abstract file
    - ['filename_Special.flr'>>namespace] in Flora2 special file
    - Keep the element name and replace the ":" with "_" in the type

XSD Import	Flora2 Abstract	Flora2 Special
<schema xmlns:ccts="abcd"> <import namespace="abcd" schemaLocation="../Information.xsd"/> <element name="person"> <complexType> <sequence> <element name="name" type="ccts:nameType"/> <element ref="ccts:age"/> <element name="work"> <complexType> <simpleContent> <extension base="ccts:workType"/> </simpleContent> </complexType> </element> </sequence> </complexType> </element> </schema>	?- ['path/Information_ Abstract.flr'>>ccts] person[name {1:1} *=> ccts_nameType]. person['ccts:age' {1:1} *=> ccts_age]. person[work {1:1} *=> personwork]. personwork['ccts:workType' {1:1} *=> ccts_workType].	?- ['path/Information_Special.flr'>>ccts] Elements[person -> name]. Elements[person -> 'ccts:age']. Elements[person -> work].

Example

Source XSD: Company X

```
<xs:element name="InvoiceCompanyX">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Bizszam" type="xs:string“/>
      <xs:element name="Ev" type="xs:string“/>
      <xs:element name="Kanyvho" type="xs:string“/>
      <xs:element name="Bizkelt" type="xs:string“/>
      <xs:element name="city" type="xs:string"
                                          minOccurs="0"/>
      <xs:element name="zip" type="xs:int"
                                          minOccurs="0"/>
      <xs:element name="street" type="xs:string"
                                         minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Flora2 Abstract: Company X

```
Namespace[value->'xs:'].
InvoiceCompanyX [Bizszam{1:1}*=>'xs:string'].
InvoiceCompanyX [Ev{1:1}*=>'xs:string'].
InvoiceCompanyX [Kanyvho{1:1}*=>'xs:string'].
InvoiceCompanyX [Bizkelt{1:1}*=>'xs:string'].
InvoiceCompanyX [city{0:*}*=>'xs:string'].
InvoiceCompanyX [zip{0:*}*=>'xs:int'].
InvoiceCompanyX [street{0:*}*=>'xs:string'].
```

Flora2 Special: Company X

```
Sequences[InvoiceCompanyX ->['Bizszam','Ev',
'Kanyvho',’Bizkelt','city','zip','street']].
Elements[InvoiceCompanyX ->Bizszam].
Elements[InvoiceCompanyX ->Ev].
Elements[InvoiceCompanyX ->Kanyvho].
Elements[InvoiceCompanyX ->Bizkelt].
Elements[InvoiceCompanyX ->city].
Elements[InvoiceCompanyX ->zip].
Elements[InvoiceCompanyX ->street].
```

Example

Target XSD: Company Y

```
<xs:element name="InvoiceCompanyY">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="InvoiceNumber" type="xs:string"/>
   <xs:element name="AccDate" type="xs:string"/>
   <xs:element name="InvoiceDate" type="xs:string"/>
   <xs:element name="DeliveryAddress" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="city" type="xs:string"
                                          minOccurs="0"/>
      <xs:element name="zip" type="xs:string"
                                          minOccurs="0"/>
      <xs:element name="DoorNo" type="xs:string"
                                          minOccurs="0"/>
      <xs:element name="street" type="xs:string"
                                          minOccurs="0"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

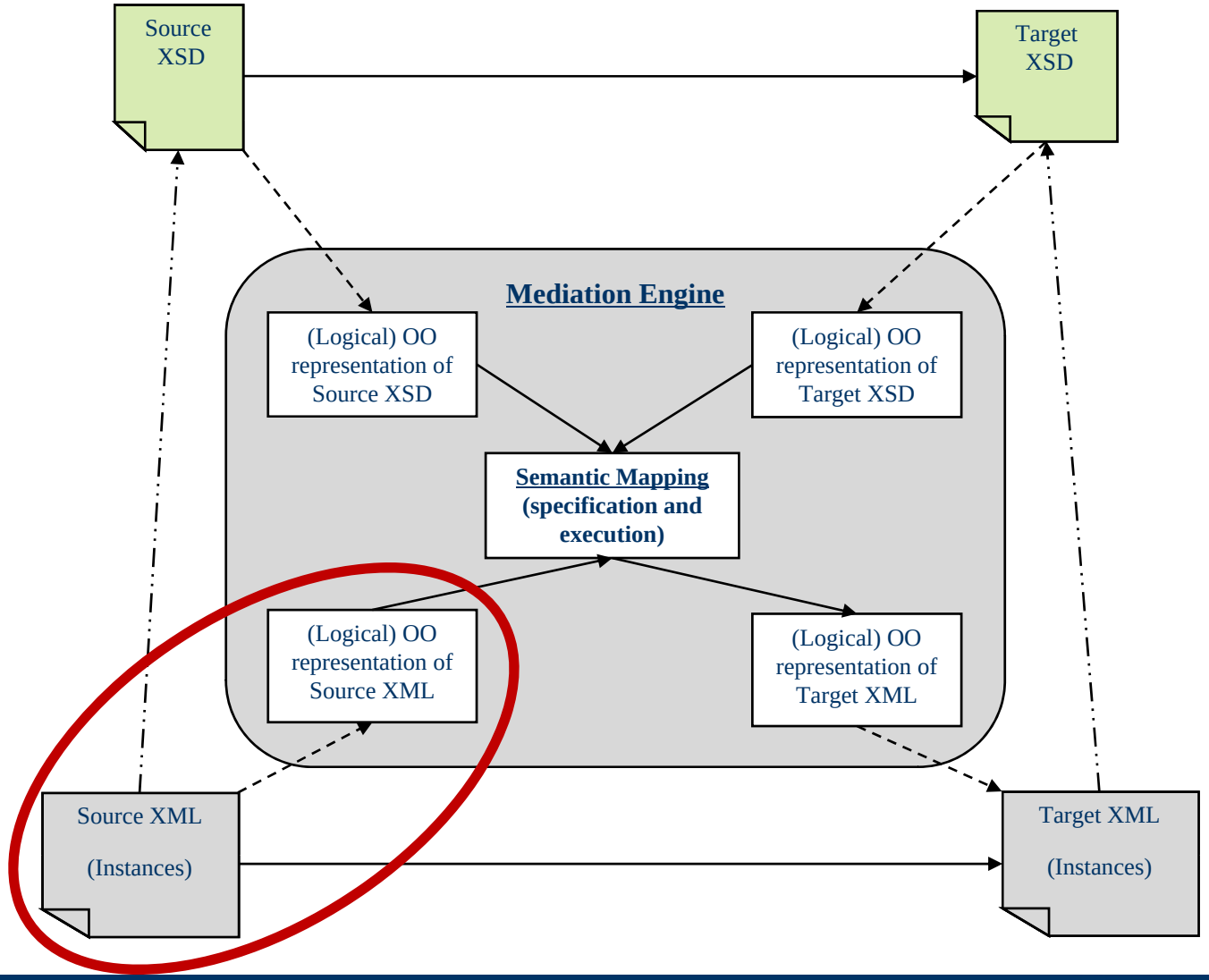
Flora2 Abstract: Company Y

```
Namespace[value->'xs:'].
YInvoice[DocumentNumber{1:1}*=>'xs:string'].
YInvoice[DocumentDate{1:1}*=>'xs:string'].
YInvoice[InvoiceDate{1:1}*=>'xs:string'].
YInvoice[DeliveryAddress{1:1}*=>YInvoiceDeliveryAddress]
YInvoiceDeliveryAddress[city{1:1}*=>'xs:string'].
YInvoiceDeliveryAddress[country{1:1}*=>'xs:string'].
YInvoiceDeliveryAddress[zip{1:1}*=>'xs:string'].
YInvoiceDeliveryAddress[DoorNumber{1:1}*=>'xs:string'].
YInvoiceDeliveryAddress[StreetNumber{1:1}*=>'xs:string'].
```

Flora2 Special: Company Y

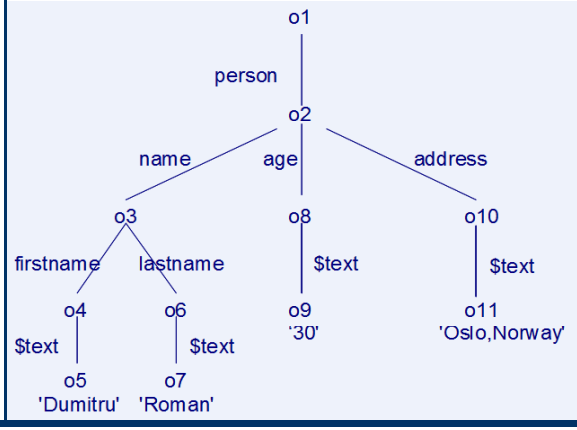
```
Sequences[LogoInvoice->['DocumentNumber','DocumentDate
','InvoiceDate','DeliveryAddress',TheOrderEnd]].
Elements[YInvoice->DocumentNumber].
Elements[YInvoice->DocumentDate].
Elements[YInvoice->InvoiceDate].
Elements[YInvoice->DeliveryAddress].
Sequences[YInvoiceDeliveryAddress->['city','country','zip',
'DoorNumber','StreetNumber',TheOrderEnd]].
Elements[YInvoiceDeliveryAddress->city].
Elements[YInvoiceDeliveryAddress->country].
Elements[YInvoiceDeliveryAddress->zip].
Elements[YInvoiceDeliveryAddress->DoorNumber].
Elements[YInvoiceDeliveryAddress->StreetNumber].
```

XML source to Flora2



XML source instances to Flora2 objects

- Steps:
  - XML instance file loaded in Flora2, resulting in a Flora2 tree
  - Flora2 abstract source file loaded in Flora2
  - Generate the Flora2 object structure according to the Flora2 abstract and query the value from Flora2 tree
    - Object names are constructed by concatenating “obj_” + a unique number (e.g. 1_1_2) generated from the unique location in the tree

XML instance (person.xml)	Flora2 tree	Flora2 objects
<person> <name> <firstname>Dumitru</firstname> <lastname>Roman</lastname> </name> <age>30</age> <address>Oslo, Norway</address> </person>		obj_:person[name->{obj_1}]. obj_1:personname[firstname->’ Dumitru ’]. obj_1:personname[lastname->’ Roman’]. obj_:person[age->30]. obj_:person[address->’ Oslo, Norway’].

Example

Source XML: Company X

```
<InvoiceCompanyX>
  <Bizszam>I_001</Bizszam>
  <Ev>2010</Ev>
  <Kanyvho>05</Kanyvho>
  <Bizkelt>2010-05-18</Bizkelt>
  <city>Oslo</city>
  <zip>1234</zip>
  <street>First Street</street>
</InvoiceCompanyX>
```

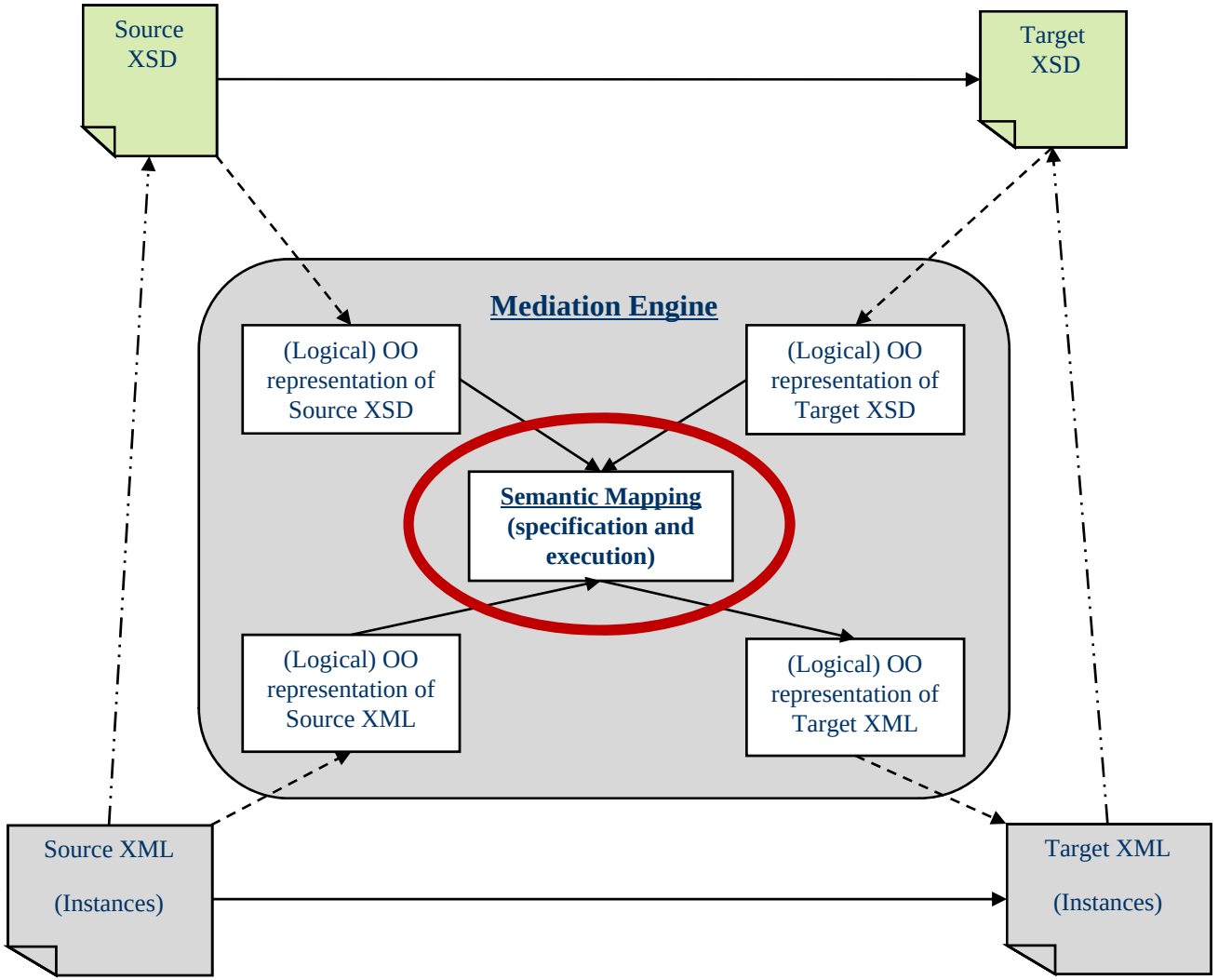
Flora2 Abstract: Company X

```
Namespace[value->'xs:'].
InvoiceCompanyX [Bizszam{1:1}*=>'xs:string'].
InvoiceCompanyX [Ev{1:1}*=>'xs:string'].
InvoiceCompanyX [Kanyvho{1:1}*=>'xs:string'].
InvoiceCompanyX [Bizkelt{1:1}*=>'xs:string'].
InvoiceCompanyX [city{0:*}*=>'xs:string'].
InvoiceCompanyX [zip{0:*}*=>'xs:int'].
InvoiceCompanyX [street{0:*}*=>'xs:string'].
```

Flora2 Objects: Company X

```
obj: InvoiceCompanyX ['Bizszam'->'I_001'].
obj: InvoiceCompanyX ['Ev'->'2010'].
obj: InvoiceCompanyX ['Kanyvho'->'05'].
obj: InvoiceCompanyX ['Bizkelt'->'2010-05-18'].
obj: InvoiceCompanyX ['city'->'Oslo'].
obj: InvoiceCompanyX ['zip'->'1234'].
```

Semantic Mapping



Example
(design time)

Schema Mapping Rules: CompanyX2CompanyY

```
OneToOne([InvoiceCompanyX],[ InvoiceCompanyY]).
OneToOne([InvoiceCompanyX,Bizszam],[ InvoiceCompanyY,InvoiceNumber ]).
OneToOne([InvoiceCompanyX,Bizkelt],[InvoiceCompanyY,InvoiceDate ]).
OneToOne([InvoiceCompanyX,City],[InvoiceCompanyY,DeliveryAddress, city]).
OneToOne([InvoiceCompanyX,Zip],[InvoiceCompanyY,DeliveryAddress, zip]).
OneToOne([InvoiceCompanyX,Street],[InvoiceCompanyY,DeliveryAddress, stree]).
ManyToOne([[InvoiceCompanyX,EV],'_',[InvoiceCompanyX,KANYVHO],'_','01'],[InvoiceCompanyY, AccDate]).
```

Flora2 Abstract: Company X

```
Namespace[value->'xs:'].
InvoiceCompanyX [Bizszam{1:1}*=>'xs:string'].
InvoiceCompanyX [Ev{1:1}*=>'xs:string'].
InvoiceCompanyX [Kanyvho{1:1}*=>'xs:string'].
InvoiceCompanyX [Bizkelt{1:1}*=>'xs:string'].
InvoiceCompanyX [city{0:*}*=>'xs:string'].
InvoiceCompanyX [zip{0:*}*−>'xs:int'].
InvoiceCompanyX [street{0:*}*=>'xs:string'].
```

Flora2 Abstract: Company Y

```
Namespace[value->'xs:'].
YInvoice[DocumentNumber{1:1}*=>'xs:string'].
YInvoice[DocumentDate{1:1}*=>'xs:string'].
YInvoice[InvoiceDate{1:1}*=>'xs:string'].
YInvoice[DeliveryAddress{1:1}*=>YInvoiceDeliveryAddress]
YInvoiceDeliveryAddress[city{1:1}*=>'xs:string'].
YInvoiceDeliveryAddress[country{1:1}*=>'xs:string'].
YInvoiceDeliveryAddress[zip{1:1}*=>'xs:string'].
YInvoiceDeliveryAddress[DoorNumber{1:1}*=>'xs:string'].
YInvoiceDeliveryAddress[StreetNumber{1:1}*=>'xs:string'].
```

Executable Mapping Rules:CompanyX2Company Y

```
?- ['InvoiceCompanyX.flr'>>SourceInstances].
?-?h: CompanyX@SourceInstances,newoid{?t},newoid{?t_4},
insert{ ?t: InvoiceCompanyY[InvoiceNumber->?t_1],
        ?t: InvoiceCompanyY [AccDate->?t_2],
        ?t: InvoiceCompanyY [InvoiceDate->?t_3],
        ?t: InvoiceCompanyY [DeliveryAddress->?t_4],
        ?t_4: InvoiceCompanyYDeliveryAddress[city->?t_4_1],
        ?t_4: InvoiceCompanyYDeliveryAddress[zip->?t_4_2],
        ?t_4: InvoiceCompanyYDeliveryAddress[street->?t_4_4]
        |
        ?t_1=?h.Bizszam@SourceInstances,
         flora_concat_items([?h.Ev@SourceInstances,_,
           ?h.Kanyvho@SourceInstances,_01],?t_2)@_plg(flrporting),
        ?t_3=?h.Bizkelt@SourceInstances,
        ?t_4_1=?h.city@SourceInstances,
        ?t_4_2=?h.zip@SourceInstances,
        ?t_4_4=?h.street@SourceInstances}.
```

Example

(run time)

Executable Mapping Rules:CompanyX2Company Y

```
?- [‘InvoiceCompanyX.flr'>>SourceInstances].
?-?h: CompanyX@SourceInstances,newoid{?t},newoid{?t_4},
insert{ ?t: InvoiceCompanyY[InvoiceNumber->?t_1],
        ?t: InvoiceCompanyY [AccDate->?t_2],
        ?t: InvoiceCompanyY [InvoiceDate->?t_3],
        ?t: InvoiceCompanyY [DeliveryAddress->?t_4],
        ?t_4: InvoiceCompanyYDeliveryAddress[city->?t_4_1],
        ?t_4: InvoiceCompanyYDeliveryAddress[zip->?t_4_2],
        ?t_4: InvoiceCompanyYDeliveryAddress[street->?t_4_4]
        |
        ?t_1=?h.Bizszam@SourceInstances,
         flora_concat_items([?h.Ev@SourceInstances,_,
           ?h.Kanyvho@SourceInstances,_01],?t_2)@_plg(flrporting),
        ?t_3=?h.Bizkelt@SourceInstances,
        ?t_4_1=?h.city@SourceInstances,
        ?t_4_2=?h.zip@SourceInstances,
        ?t_4_4=?h.street@SourceInstanc}.
```

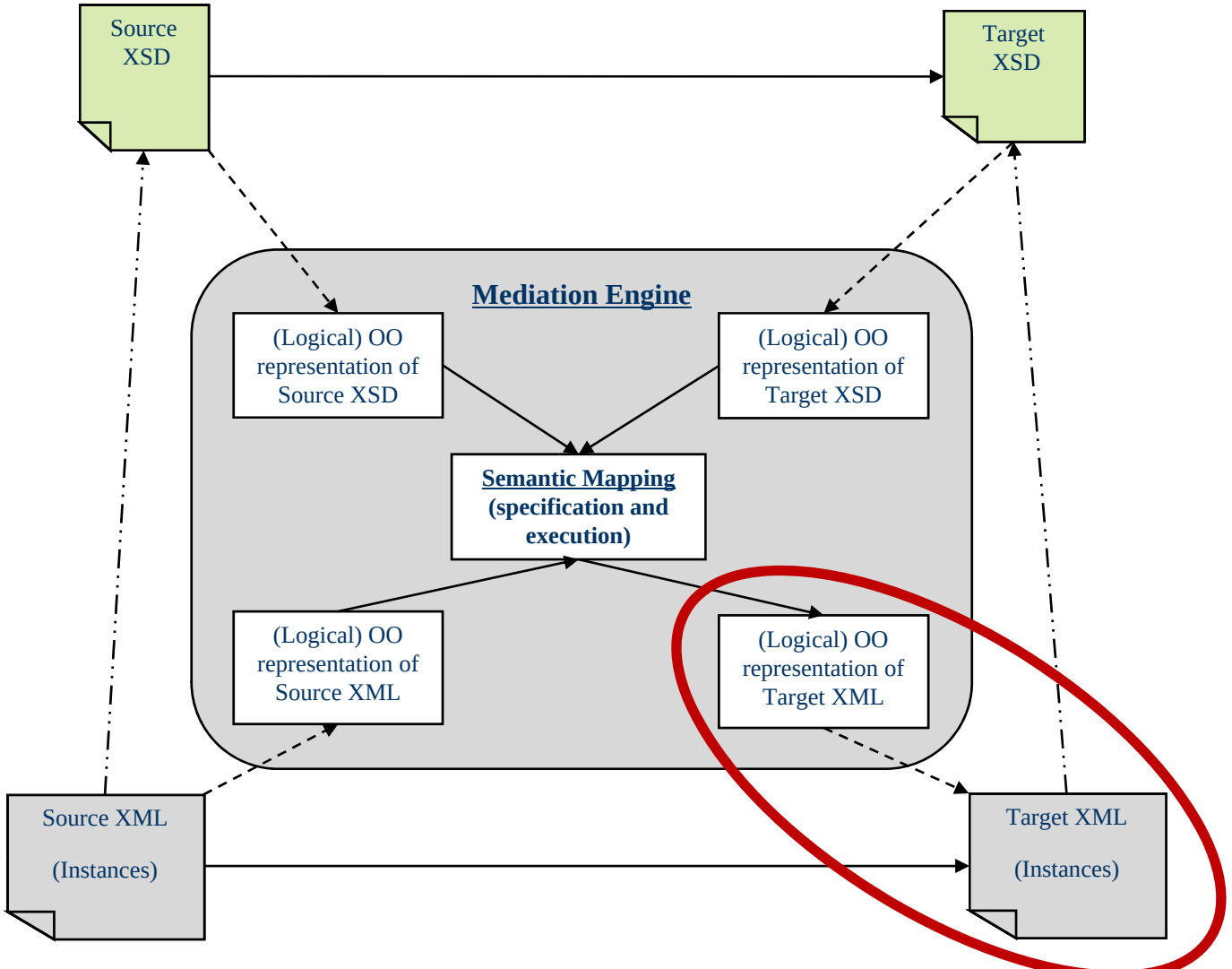
Flora2 Objects: Company X

```
obj: InvoiceCompanyX ['Bizszam'->'I_001'].
obj: InvoiceCompanyX ['Ev'->'2010'].
obj: InvoiceCompanyX ['Kanyvho'->'05'].
obj: InvoiceCompanyX ['Bizkelt'->'2010-05-18'].
obj: InvoiceCompanyX ['city'->'Oslo'].
obj: InvoiceCompanyX ['zip'->'1234'].
obj: InvoiceCompanyX ['street'->'First Street'].
```

Flora2 Objects : Company Y

```
Obj:CanonicalModelInvoice[InvoiceNumber->’I_001’].
Obj:CanonicalModelInvoice[AccDate->'2010_05_01'].
Obj:CanonicalModelInvoice[InvoiceDate->’ 2010-05-18'].
Obj:CanonicalModelInvoice[DeliveryAddress->{obj_4}].
obj_4:CanonicalModelInvoiceDeliveryAddress[city->’Oslo,
Norway'].
obj_4:CanonicalModelInvoiceDeliveryAddress[zip->‘1234'].
obj_4:CanonicalModelInvoiceDeliveryAddress[street->
‘First Street'].
```

Flora2 target objects to XML target instances



Example

Flora2 Abstract: Company Y

```
Namespace[value->'xs:'].
YInvoice[DocumentNumber{1:1}*=>'xs:string'].
YInvoice[DocumentDate{1:1}*=>'xs:string'].
YInvoice[InvoiceDate{1:1}*=>'xs:string'].
YInvoice[DeliveryAddress{1:1}*=>YInvoiceDeliveryAddress]
YInvoiceDeliveryAddress[city{1:1}*=>'xs:string'].
YInvoiceDeliveryAddress[country{1:1}*=>'xs:string'].
YInvoiceDeliveryAddress[zip{1:1}*=>'xs:string'].
YInvoiceDeliveryAddress[DoorNumber{1:1}*=>'xs:string'].
YInvoiceDeliveryAddress[StreetNumber{1:1}*=>'xs:string'].
```

Flora2 Objects: Company Y

```
Obj:CanonicalModelInvoice[InvoiceNumber->'I_001'].
Obj:CanonicalModelInvoice[AccDate->'2010_05_01'].
Obj:CanonicalModelInvoice[InvoiceDate->' 2010-05-18'].
Obj:CanonicalModelInvoice[DeliveryAddress->{obj_4}].
obj_4:CanonicalModelInvoiceDeliveryAddress[city->
'Oslo, Norway'].
obj_4:CanonicalModelInvoiceDeliveryAddress[zip->
'1234'].
obj_4:CanonicalModelInvoiceDeliveryAddress[street->
'First Street'].
```

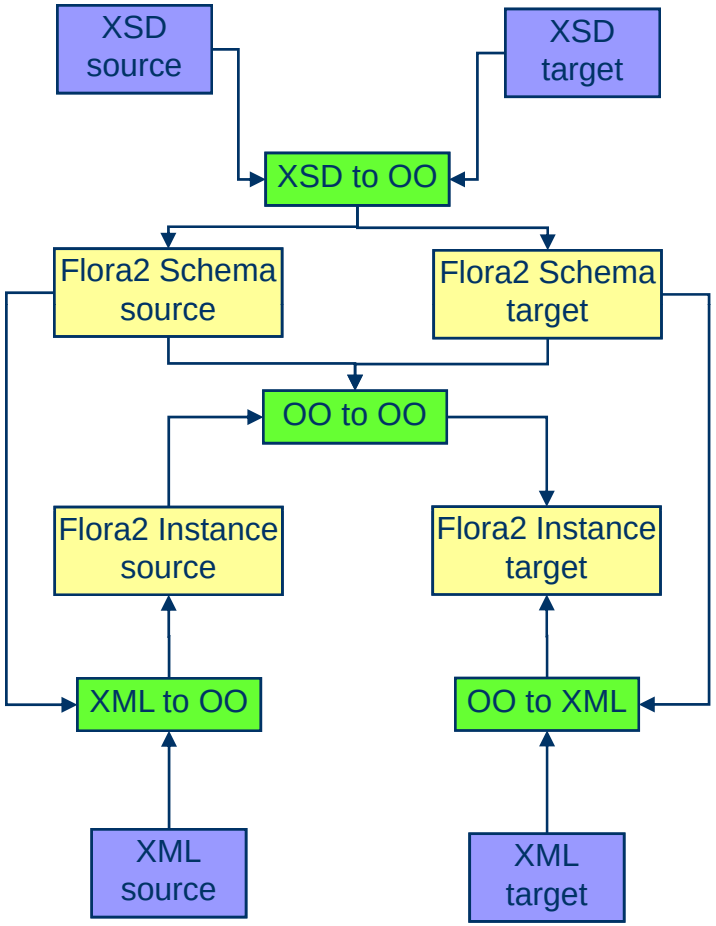
Flora2 Special: Company Y

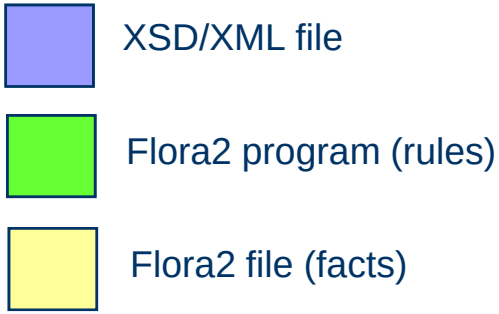
```
Sequences[LogoInvoice->['DocumentNumber','DocumentDate
','InvoiceDate','DeliveryAddress',TheOrderEnd]].
Elements[YInvoice->DocumentNumber].
Elements[YInvoice->DocumentDate].
Elements[YInvoice->InvoiceDate].
Elements[YInvoice->DeliveryAddress].
Sequences[YInvoiceDeliveryAddress->['city','country','zip',
'DoorNumber','StreetNumber',TheOrderEnd]].
Elements[YInvoiceDeliveryAddress->city].
Elements[YInvoiceDeliveryAddress->country].
Elements[YInvoiceDeliveryAddress->zip].
Elements[YInvoiceDeliveryAddress->DoorNumber].
Elements[YInvoiceDeliveryAddress->StreetNumber].
```

XML Instances: Company Y

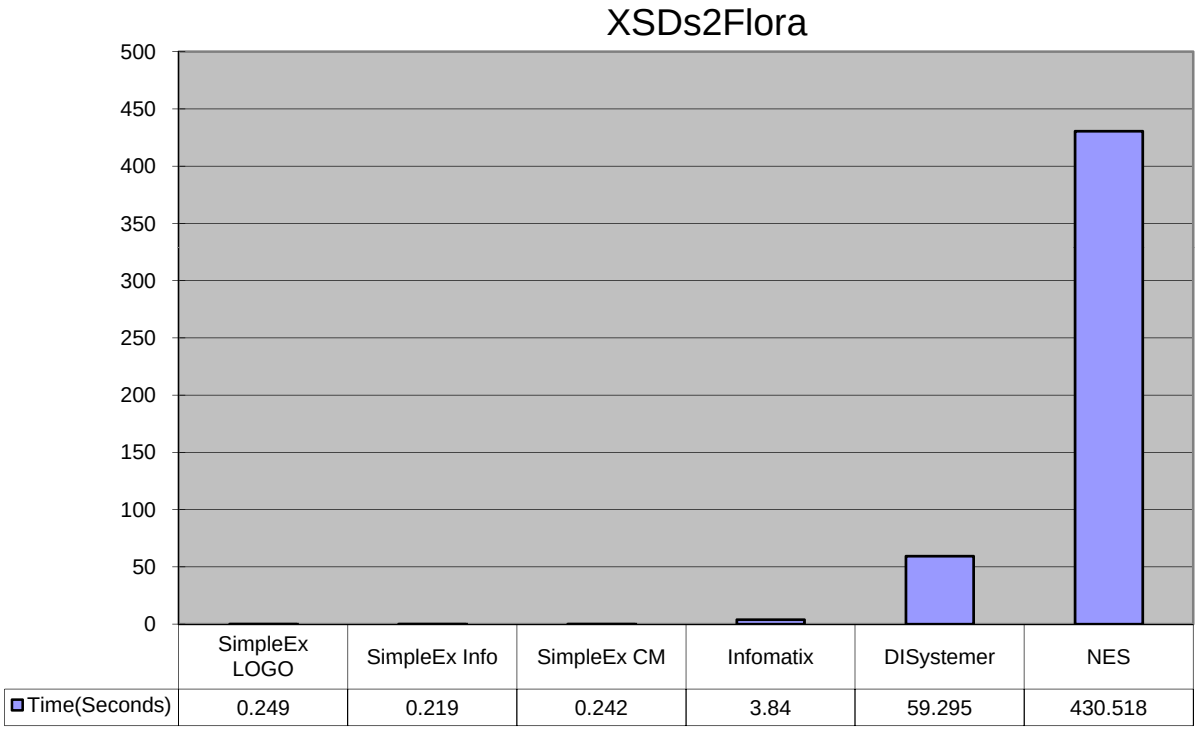
```
<?xml version="1.0"?>
< InvoiceCompanyY >
   <InvoiceNumber>I_001</InvoiceNumber>
   <AccDate>2010_05_01</AccDate>
    <InvoiceDate>2010-05-18</InvoiceDate>
   <DeliveryAddress>
      <city>Oslo</city>
      <zip>1234</zip>
      <DoorNo> </DoorNo>
      <street>First Street </street>
    </DeliveryAddress>
</ InvoiceCompanyY>
```

Implementation in Flora2

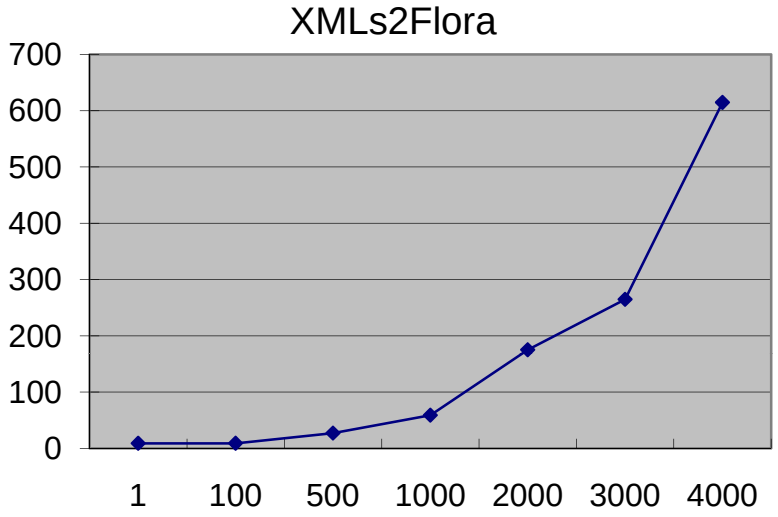


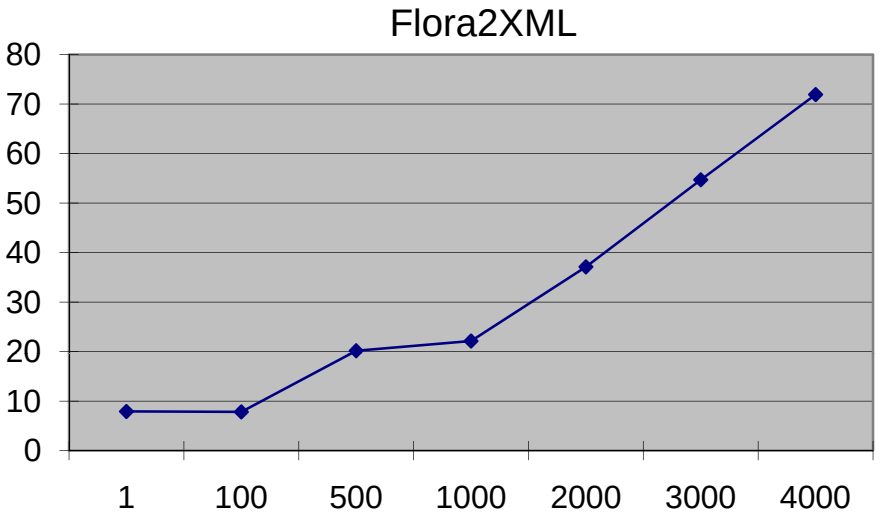


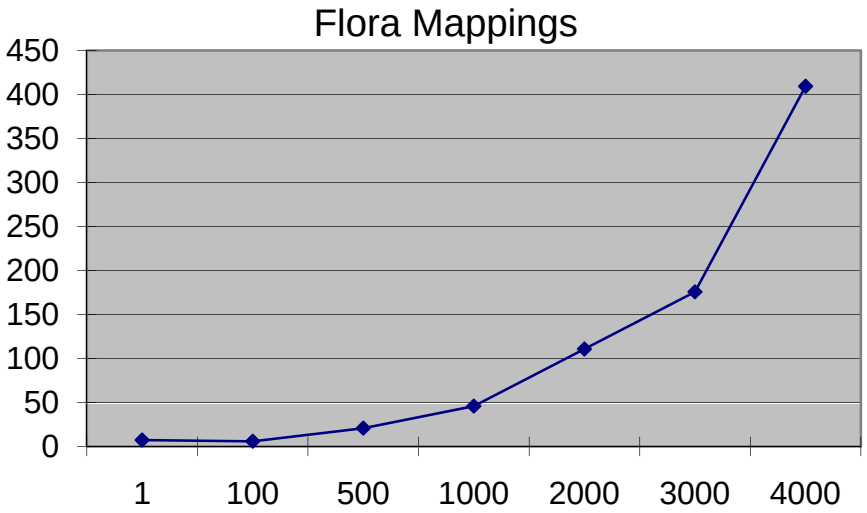
Experiment: XSDs to Flora

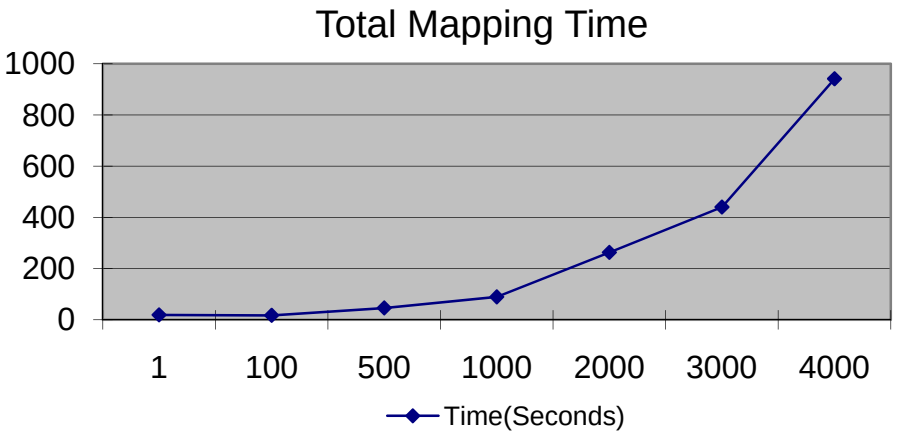


Experiment: Complete mapping





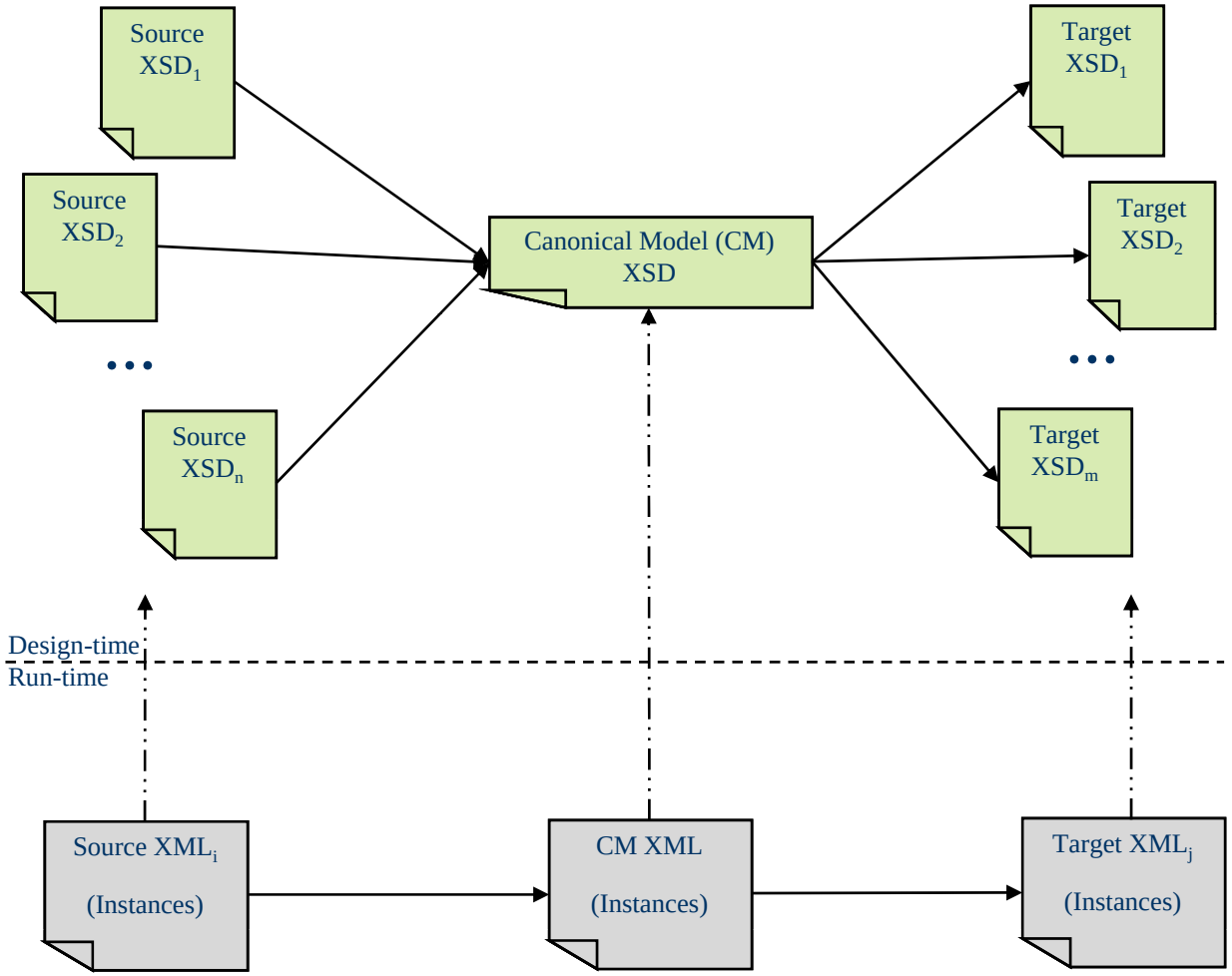




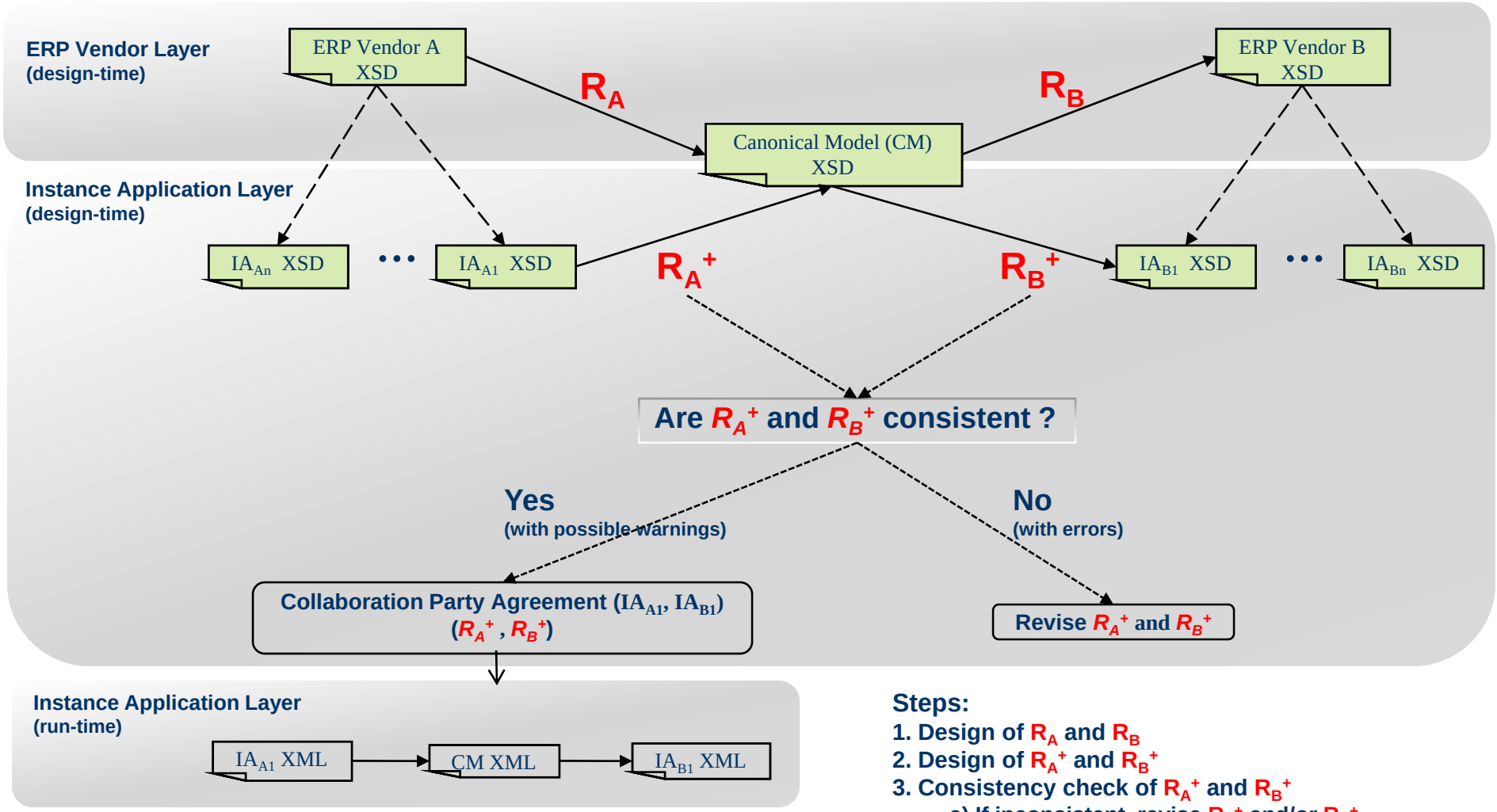
Outline

- Intro to XML Data Exchange
- FloraMap: Flora2-based XML data transformation
  - Technique: steps and examples
  - Implementation and experiments
- Generic XML Data Exchange Framework
- Related Work
- Conclusions and Future Work

Generic M-N desing- and run-time XML data transformation



XML data transformation in B2B – Overview
(General scenario)

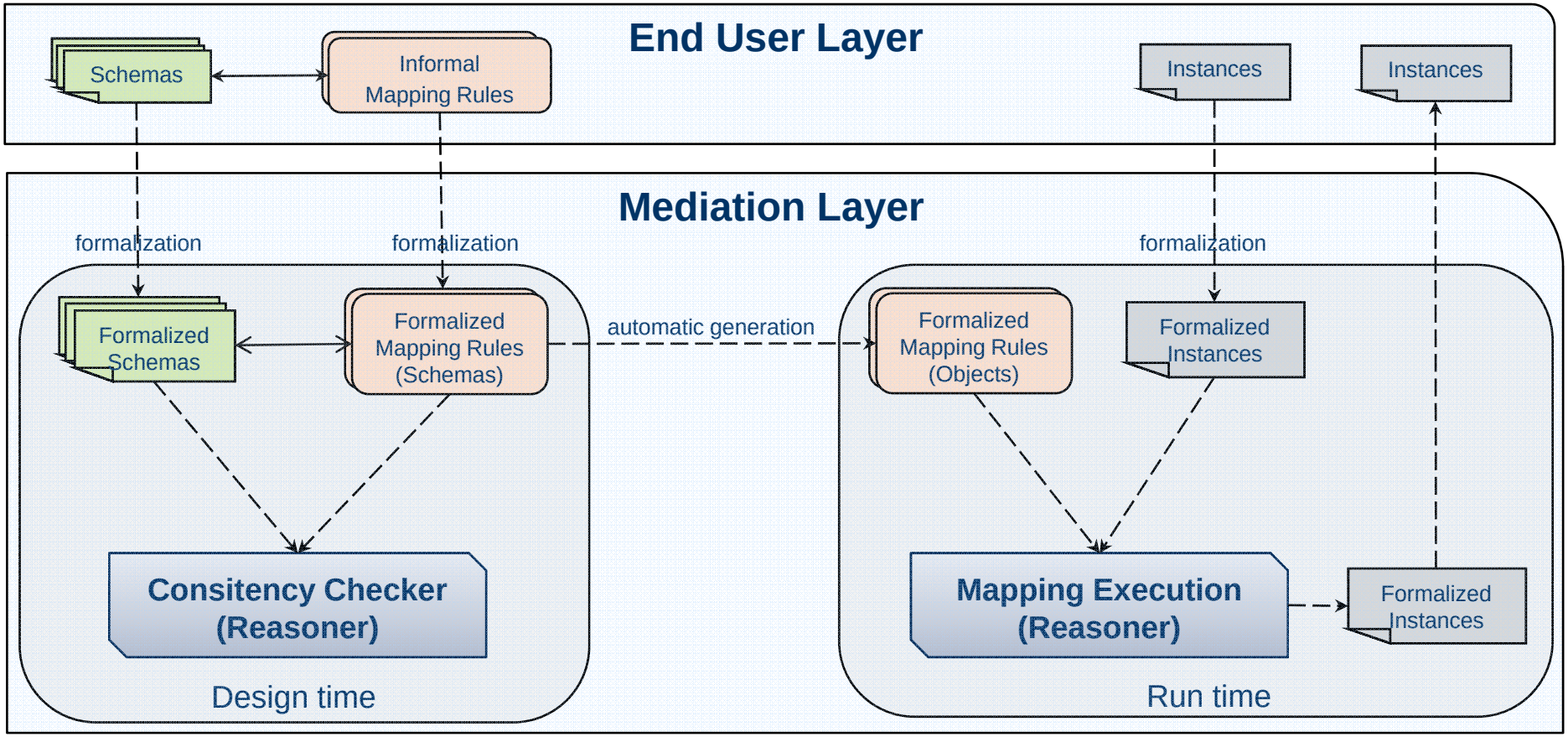


Steps:

1. Design of R_A and R_B
2. Design of R_A^+ and R_B^+
3. Consistency check of R_A^+ and R_B^+
   a) If inconsistent, revise R_A^+ and/or R_B^+
   b) else, run-time transformation

R_X – mapping rules at the ERP Vendor Layer
R_X^+ – mapping rules at the Instance Application Layer

Generic Mapping Framework



Outline

- Intro to XML Data Exchange
- FloraMap: Flora2-based XML data transformation
  - Technique: steps and examples
  - Implementation and experiments
- Generic XML Data Exchange Framework
- Related Work
- Conclusions and Future Work

Areas of Related Work

- Object-oriented representations of XML/XSD
- MDE model transformations
- The use of rule-based logical systems for data mapping/exchange hasn’t been yet widely investigated in the community

Outline

- Intro to XML Data Exchange
- FloraMap: Flora2-based XML data transformation
  - Technique: steps and examples
  - Implementation and experiments
- Generic XML Data Exchange Framework
- Related Work
- Conclusions and Future Work

Conclusions and Future Work

- FloraMap – promising technique for XML data exchange
  - Allows both specification and execution of data mappings in a single, unifying framework
  - End-to-end solution to the problem of XML data exchange
- Planned extensions:
  - End-to-end mappings between multiple sources and multiple targets
  - Consistency checking
  - Inclusion of other types of schemas (not only XSDs)
  - (Semi-)Automated generation of executable mapping rules

Thank you!

Q&A