Seminari di sistemi informatici 2004 - 2005

Model Management e XML

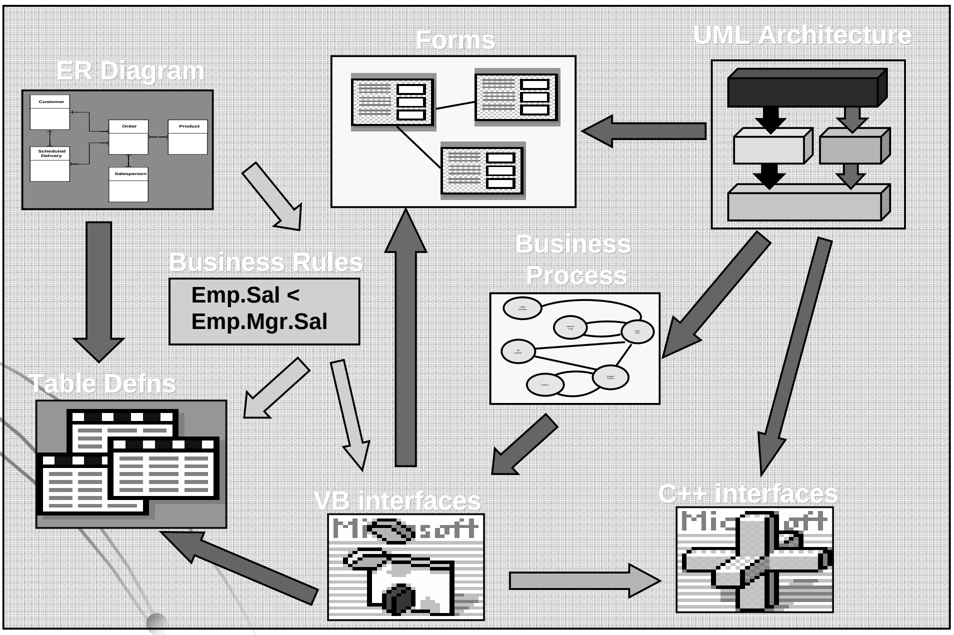
Paolo Papotti
papotti@dia.uniroma3.it
Lab. basi di dati

Cosa vedremo oggi

- Idea *Model Management*
  - Problemi
  - Operatori
- Chameleon
  - *Cosa* fa e *come* lo fa
  - Risultati sperimentali
  - Possibili progetti

Meta Data Management

- Meta data = structural information
  - DB schema, interface defn, web site map, form defns, …



Such Problems are Pervasive

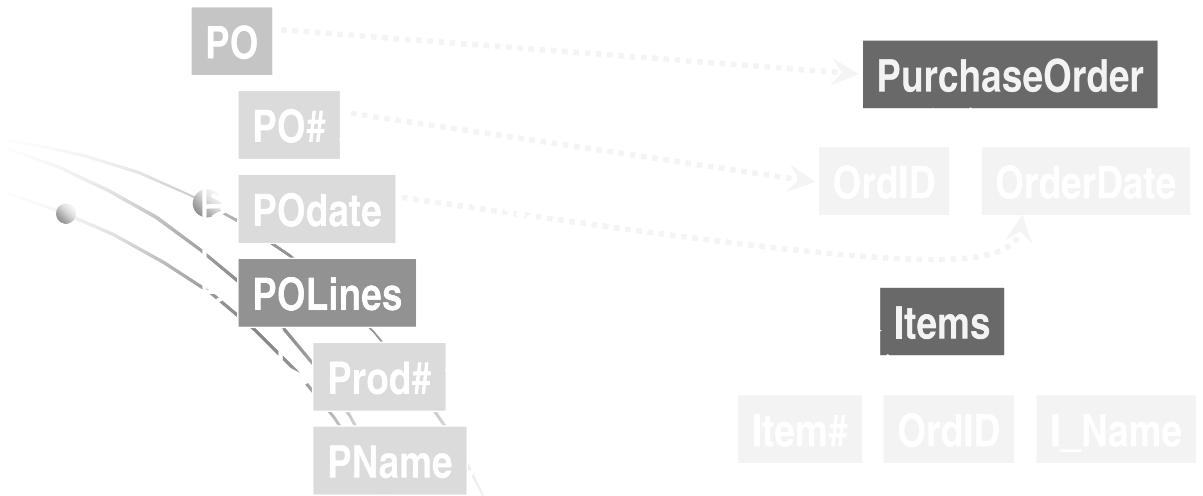
- Data translation
- Schema evolution & data migration
- XML message translation for e-commerce
- Integrate custom apps with commercial apps
- Data warehouse loading (clean & transform)
- Design tool support (DB, UML, …)
- Database-driven portal generation
- OO or XML wrapper generation for SQL DB

Meta Data Problems

- They all involve schemas and mappings
- E.g., data translation between data models

Hierarchical Schema

Relational Schema



PO

PO#

POdate

POLines

Prod#

PName

PurchaseOrder

OrdID

OrderDate

Items

Item#

OrdID

I_Name

Meta Data Solutions

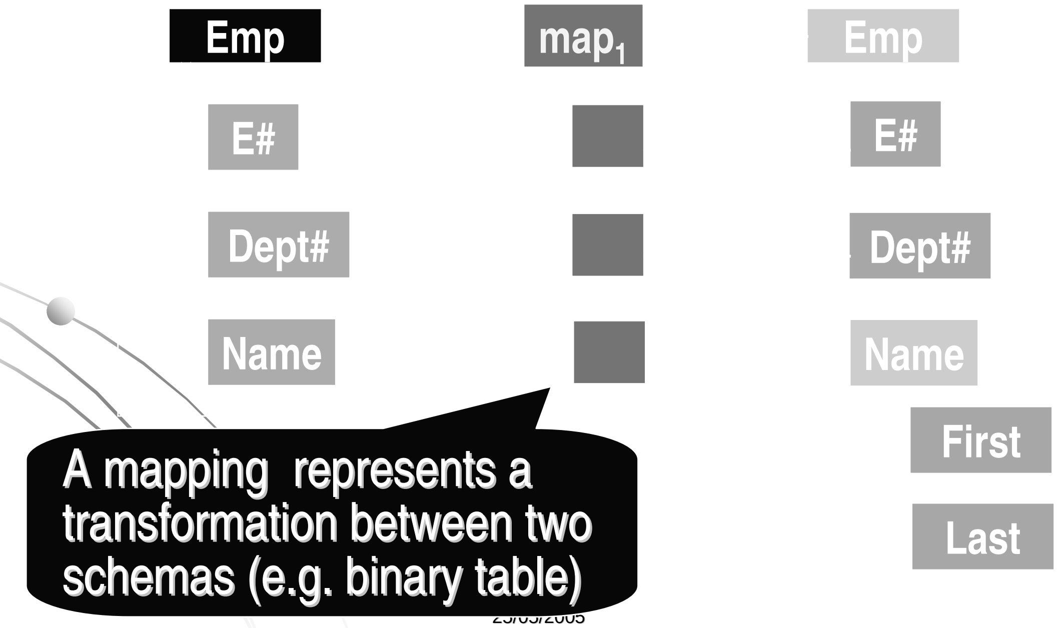
- Solutions strongly resemble each other, but
  - usually are problem-specific
  - usually are language-specific
    SQL, ODMG, UML, XML, RDF, ….
  - usually involve a lot of object-at-a-time programming
- Goals
  - Generic solutions
  - “Set”-at-a-time programming

Model Management

- A generic approach to meta data mgmt
- Model Mgmt operators manipulate *schemas* and *mappings* as bulk objects
  - Their representation is generic
  - Operators:
    - Match, Merge, Diff, Compose, ModelGen, …
- Avoids problem-specific and language-specific solutions

Models and Mappings

A schema **is a rooted directed graph, which represents a complex information structure.**



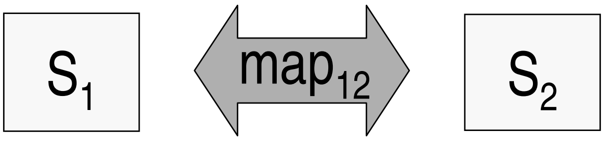
A mapping represents a transformation between two schemas (e.g. binary table)

Model Mgmt Algebra

- map = Match (S_1, S_2)
- $<S_3, map_{13}, map_{23}>$ = Merge (S_1, S_2, map)
- map_3 = Compose(map_1, map_2)
- $<S_2, map_{12}>$ = Diff(S_1, map)
- $<S_2, map_{12}>$ = ModelGen(S_1, model$_2$)
- S_2 = Copy(S_1)
- Apply, Insert, Delete, . . .

Categorizing Meta Data Problems

- Scheme mapping



S_1

map_{12}

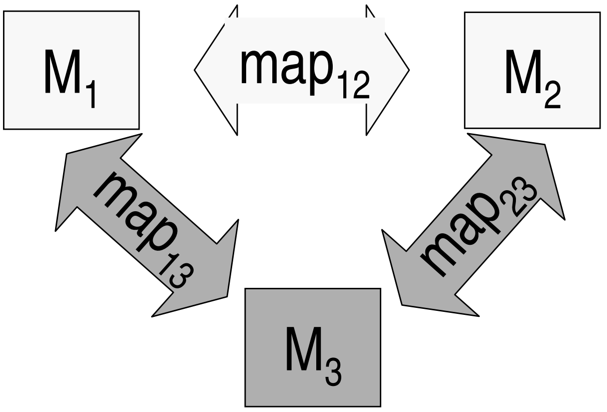
S_2

  - Data translation
  - XML message translation for e-commerce
  - Integrate custom apps with commercial apps
  - Data warehouse loading (clean & transform)

- Solution is the **match** “operator”

Categorizing M D Problems (2)

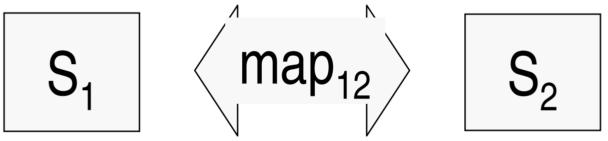
- Scheme integration


M_1
map_{12}
M_2
map_{13}
map_{23}
M_3

  - View integration
  - Data integration
- Solution is the Merge operator

Categorizing M D Problems (3)

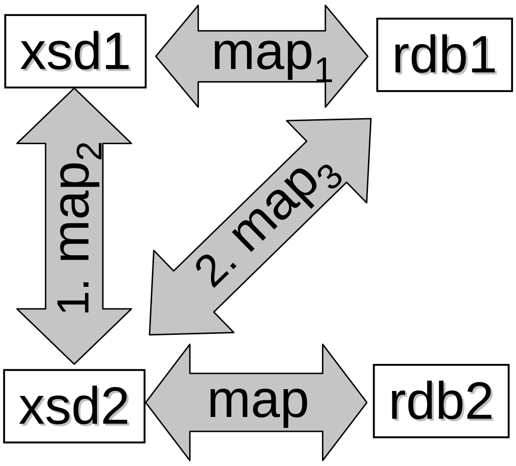
- Scheme and mapping generation


S_1
map_{12}
S_2

  - Design tools (ER → SQL)
  - Wrapper generation (SQL → OO or XML)
- Solution is the **ModelGen** operator
  - $<S_2, map_{12}> = \mathrm{ModelGen}(S_1, \mathrm{model}_2)$

E.g. Change Propagation

- Given
  - map_1 between xsd1 and SQL schema rdb1
  - xsd2, a modified version of xsd1
- Produce
  - rdb2 to store instances of xsd2
  - a mapping between xsd2 and rdb2



Chameleon

An Extensible and Customizable Tool for Web Data Translation

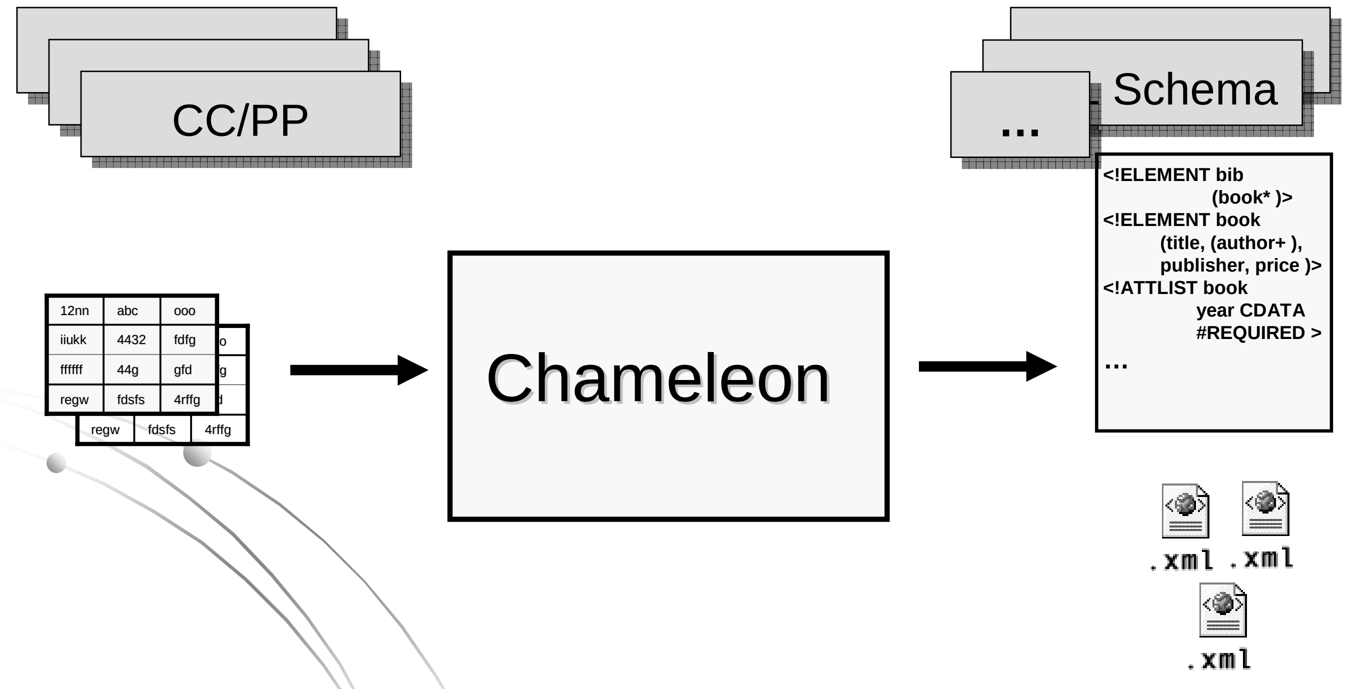
Motivazioni

- Internet incoraggia la condivisione dei dati e lo scambio di questi anche fra fonti eretogenee: XML, modelli relazionali o a oggetti, modelli per dati semistrutturati.
- Anche a livello concettuale, le strutture dati sono spesso descritte con diversi formalismi: modello ER e le sue varianti o sottoinsiemi di UML
- Necessità di gestione integrata della descrizione dei dati ⇨ traduzione facile e flessibile da un modello all’altro

Goals

- Supporting cooperation and data interchange between different organizations with distinct and heterogeneous data sources
- Development of a tool for the automatic translation of schemes and instances between models
  - Models are not fixed a priori

Scheme and instance



12nn	abc	ooo
iiukk	4432	fdfg
ffffff	44g	gfd
regw	fdsfs	4rffg

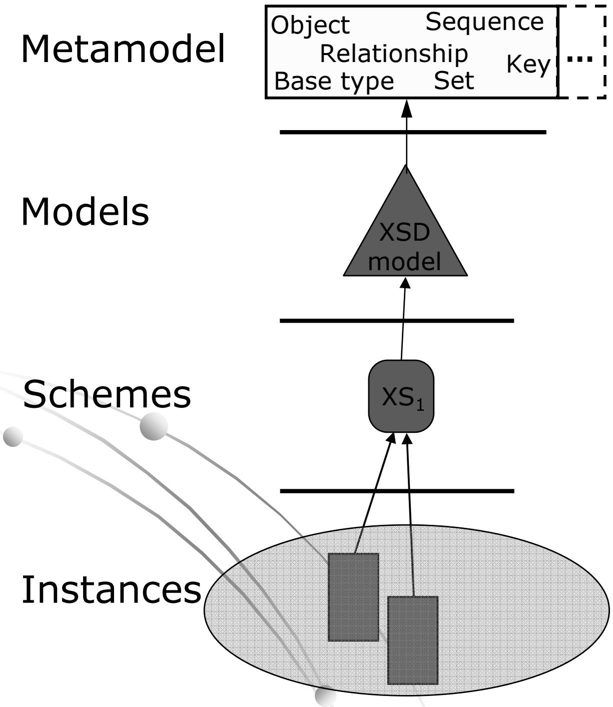
regw	fdsfs	4rffg

```
<!ELEMENT bib
              (book* )>
<!ELEMENT book
         (title, (author+ ),
         publisher, price )>
<!ATTLIST book
              year CDATA
              #REQUIRED >
...
```

Approccio

- **Gestione dei modelli**
  - **Chameleon** è basato su un *metamodello* composto da un insieme di *metaprimitive*
  - Una metaprimitiva corrisponde a una classe di costrutti base per i dati: elemento, attributo, relazione, relationship, tipo base, sequenza, … (Hull&King, 1987)
  - Un modello viene definito specificando le metaprimitive che utilizza per rappresentare i dati e le loro caratteristiche (quando sono ammesse, con che limiti, con che sintassi, ...)

Metamodel

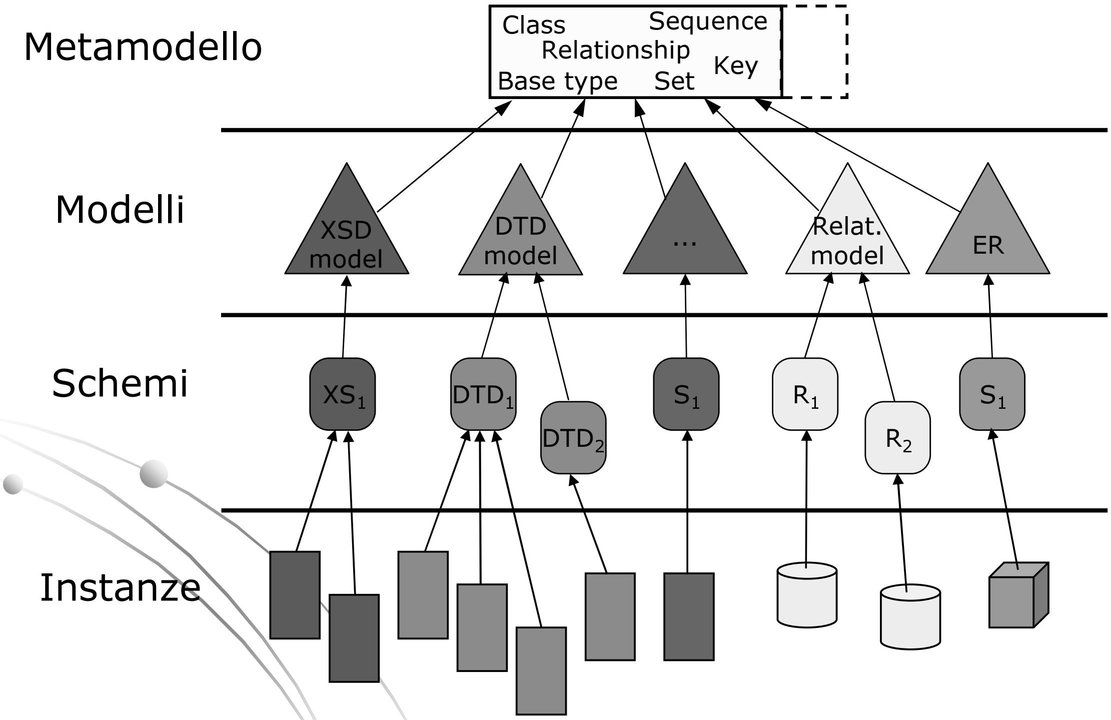


- Metamodel:
  - Set of classes of constructs
- Model:
  - Set of constructs to define schemes
- Scheme:
  - XSD and DTD files
  - Database schemes
- Data:
  - Relational tables
  - XML files
  - Semi structured data

Steps

1. the definition of a “meta-formalism” that captures:
   - main primitives adopted by different schema languages for (semi)structured data
   - basic constructs used by traditional database models
2. the definition of an effective method for the translation between models, which makes use of the meta-formalism as a level of reference

Scenario riferimento



First step: metamodel

- Classification of primitives adopted by the various models into classes (*metaprimitive)*
- Supermodel
- A model is defined by associating its primitives with the metaprimitive in the metamodel (syntax translation)
- Metaprimitives: Abstract Object, Concrete Object, Base type, User define type, Ordered sequence, Unordered sequence, Choice, Cardinality, Key, Foreign key, ...
- XML-based:
  - models and schemes represented in XML

Why?

- Two positive aspects:

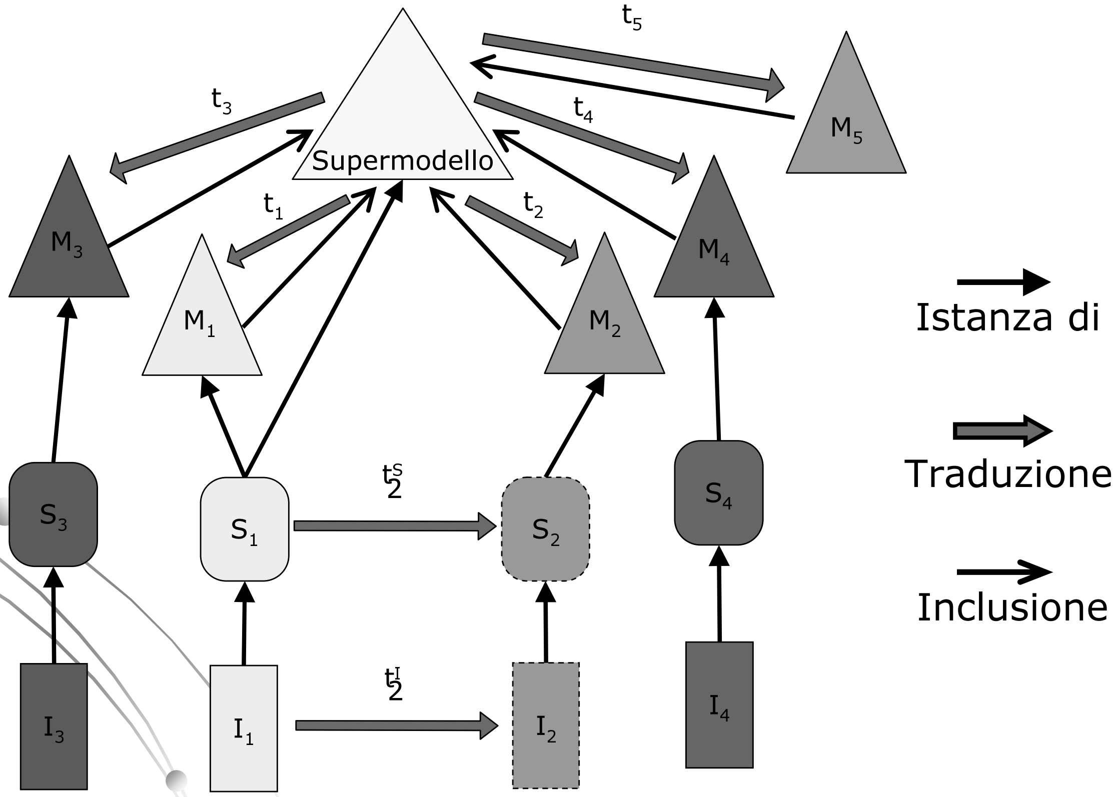
1. Representation of schemes and models with common constructs
   - Add easily new models and constructs
2. Reuse of translations between constructs
   - Translate between models with shared procedures

Second step: translation

- Library of *basic procedures:* set of transformations implementing translations between individual (or combinations of) metaprimitives
- Complex translation can be obtained as composition of elementary steps
- XML Based: XSLT and XQuery

- **<u>Goal</u>: Automatic** generation of a **sequence** of procedures to translate complex schemes and instances

Tecnica traduzione



Esempio traduzione

Schema sorgente (XML Schema)

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="Order" type="OrderType"/>
 <xsd:complexType name="OrderType">
  <xsd:sequence>
   <xsd:element name="destination" type="USAddress"/>
   <xsd:element name="items"  type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
 </xsd:complexType>
 <xsd:complexType name="USAddress">
  <xsd:all>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city"   type="xsd:string"/>
   <xsd:element name="zip"    type="xsd:decimal"/>
  </xsd:all>
  <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
 </xsd:complexType>
 <xsd:complexType name="Items">
  <xsd:sequence>
   <xsd:element name="item" minOccurs="0" maxOccurs="10">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="productName" type="xsd:string" />
      <xsd:element name="quantity" type="xsd:integer" />
      <xsd:element name="USPrice"  type="xsd:decimal"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

Sorgente nel supermodello

```
<META source="xsd">
 <element name="Order" type="OrderType">
  <sequence cardinality="1:1">
   <element name="destination" type="USAddress" cardinality="1:1">
    <unorderedSequence cardinality="1:1" >
     <element name="street" type="string" cardinality="1:1" />
     <element name="city" type="string" cardinality="1:1" />
     <element name="zip" type="decimal" cardinality="1:1" />
    </unorderedSequence>
    <attribute name="country" type="string" cardinality="0:1">
     <fixed>US</fixed>
    </attribute>
   </element>
   <element name="items" type="Items" cardinality="1:1">
    <sequence cardinality="1:1">
     <element name="item" cardinality="0:10">
      <sequence cardinality="1:1">
       <element name="productName" type="string" cardinality="1:1" />
       <element name="quantity" type="integer" cardinality="1:1" />
       <element name="USPrice" type="decimal" cardinality="1:1" />
      </sequence>
     </element>
    </sequence>
   </element>
  </sequence>
  <attribute name="orderDate" type="date" cardinality="0:1" />
 </element>
</META>
```

Trasformazione delle metaprimitive

```
<META source="xsd" target="dtd">
 <element name="Order" root="true">
  <sequence cardinality="1:1">
   <element name="destination" cardinality="1:1">
    <sequence cardinality="0:N">
     <element name="street" type="string" cardinality="1:1" />
     <element name="city" type="string" cardinality="1:1" />
     <element name="zip" type="string" cardinality="1:1" />
    </sequence>
    <attribute name="country" type="string" cardinality="0:1" >
     <fixed>US</fixed>
    </attribute>
   </element>
   <element name="items" cardinality="1:1">
    <sequence cardinality="1:1">
     <element name="item" cardinality="0:N">
      <sequence cardinality="1:1">
       <element name="productName" type="string"  cardinality="1:1" />
       <element name="quantity" type="string" cardinality="1:1" />
       <element name="USPrice" type="string" cardinality="1:1" />
      </sequence>
     </element>
    </sequence>
   </element>
  </sequence>
  <attribute name="orderDate" type="string" cardinality="0:1" />
 </element>
</META>
```

Destinazione nel supermodello

Schema destinazione (DTD)

```
<!DOCTYPE Order[
   <!ELEMENT Order (destination,items)>
   <!ELEMENT destination (street,city,zip)>
   <!ELEMENT street (#PCDATA)>
   <!ELEMENT city (#PCDATA)>
   <!ELEMENT zip (#PCDATA)>
   <!ELEMENT items (item*)>
   <!ELEMENT item (productName,quantity,USPrice)>
   <!ELEMENT productName (#PCDATA)>
   <!ELEMENT quantity (#PCDATA)>
   <!ELEMENT USPrice (#PCDATA)>
   <!ATTLIST Order orderDate CDATA #IMPLIED>
   <!ATTLIST destination country CDATA #FIXED "US">
   ]>
```

Procedure

1. **Nidicazioni di oggetti complessi** Se un elemento complesso ha all'interno degli elementi atomici (o attributi) allora non è identificabile come nidicazione. Al contrario due elementi complessi sì.
2. **Verica presenza chiave** per ogni elemento complesso ed eventuale creazione (Skolem e contatori).
3. **Verifica presenza namespace** e gestione: eliminazione/scrittura e creazione/riscrittura

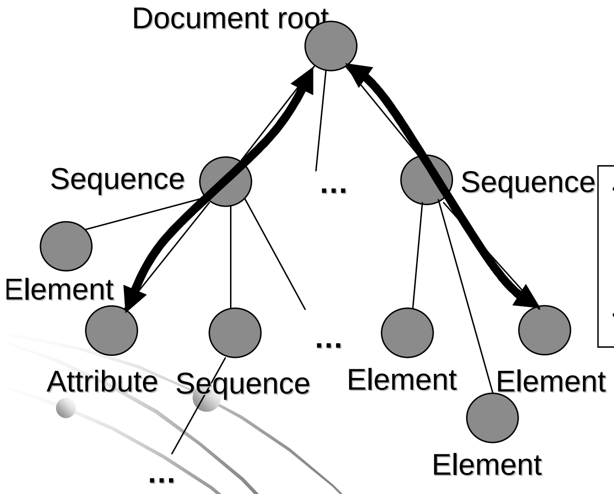
Procedure 2

4. **Gestione cardinalità** Un modello può non esprimere determinate cardinalità (es. DTD: non esiste la possibilita di indicarne esatte tipo 1:12, ma solo 1:1 o 1:n). A volte basta estendere, a volte limitare
   - dall'ER al relazionale: le relazioni con cardinalità n devono essere tradotte in relazioni binarie o viceversa
5. **Verifica presenza sequenze ordinate** e gestione: tradurre come sequenza non ordinata o invocare una procedura per conservare l'ordinamento?
6. **Individuazione delle generalizzazioni, estensioni, restrizioni** (ricostruzione?)

Library of Procedures

- Nesting/unnesting of complex and atomic elements
- Key/foreign key creation
- Management of ordered/unordered sequence
- Management of cardinality (restriction, extension)
- Addition/removal of namespace
- Management of generalization hierarchies/unions
- Management of built in/extended types
- …

Key creation



```
<Dept>
  <DeptName>Storage</DeptName>
  <CreationDate>1999-01-07</CreationDate>
</Dept>
```

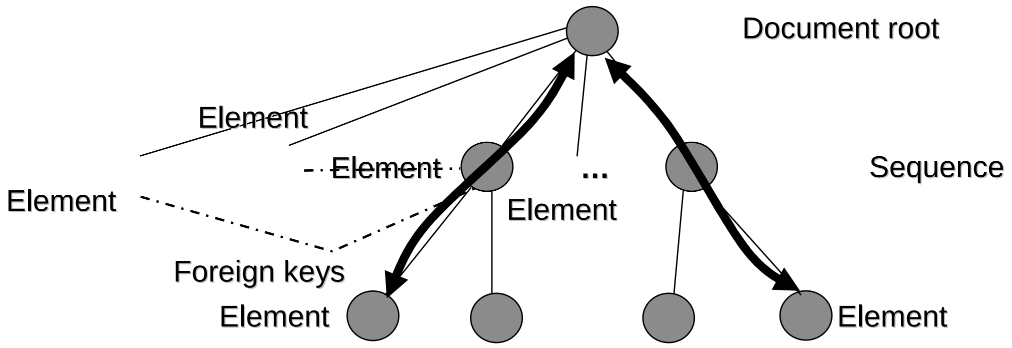
```
<Ingredient quantity="20" unit="g">
  …
</Ingredient>
```

```
<Dept>
  <DeptName>Storage</DeptName>
  <CreationDate>1999-01-07</CreationDate>
  <Dept-New-Key>1(Storage,1999-01-07)</Dept-New-Key>
</Dept>
```

```
<Ingredient quantity="20" unit="g">
  …
  <Ingredient-New-Key>1g</Ingredient-New-Key>
</Ingredient>
```

Create key with counter

Unnesting (scheme)



Document root

Element

Element

...

Sequence

Element

Element

Foreign keys

Element

Element

creation of new references.

Model translation

- Input: a scheme $\mathbf{S_S}$ of a model $\mathbf{M_S}$, a library of procedures **L**, and the target model $\mathbf{M_T}$
- Output: a scheme $\mathbf{S_T}$ for $\mathbf{M_T}$, a set of procedures **t**, a residual **r**
    - For each instance **I of $\mathbf{S_S}$,** t(I) is an instance of $\mathbf{S_T}$
- Algorithm
    1. Serialization (if needed)
    2. **Syntactical translation** of the scheme into the supermodel
    3. Model matching: identification of metaprimitives to be transformed
    4. Selection of **procedures** from the library
    5. Application of **procedures**

Esempio

- Portare dati su dipartimenti e impiegati da un insieme di documenti XML a un database relazionale
- Conosciamo lo schema di partenza (XMLSchema) e le istanze (documenti XML)

Schema sorgente

```
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name = "Dept" >
        <xsd:complexType>
           <xsd:sequence>
              <xsd:element name="DeptName" t
                                   ype="xsd:string"/>
              <xsd:element name="CreationDate”
                                   type="xsd:date"/>
              <xsd:element name = "Emps" >
                 <xsd:complexType>
                    <xsd:sequence>
                       <xsd:element name = "Emp“
                                   maxOccurs="unbounded">
                          <xsd:complexType>
                             <xsd:sequence>
                                <xsd:element name="EmpID“
                                        type="xsd:integer"/>
                                <xsd:element name="EmpName“
                                        type="xsd:string"/>
                             </xsd:sequence>
                          </xsd:complexType>
                       </xsd:element>
                    </xsd:sequence>
                 </xsd:complexType>
              </xsd:element>
           </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Istanza sorgente

```
<Dept>
  <DeptName>Storage</DeptName>
  <CreationDate>1999-01-07</CreationDate>
  <Emps>
    <Emp>
      <EmpID>37</EmpID>
      <EmpName>Paul</EmpName>
    </Emp>
    <Emp>
      <EmpID>48</EmpID>
      <EmpName>Andrew</EmpName>
    </Emp>
  </Emps>
</Dept>
```

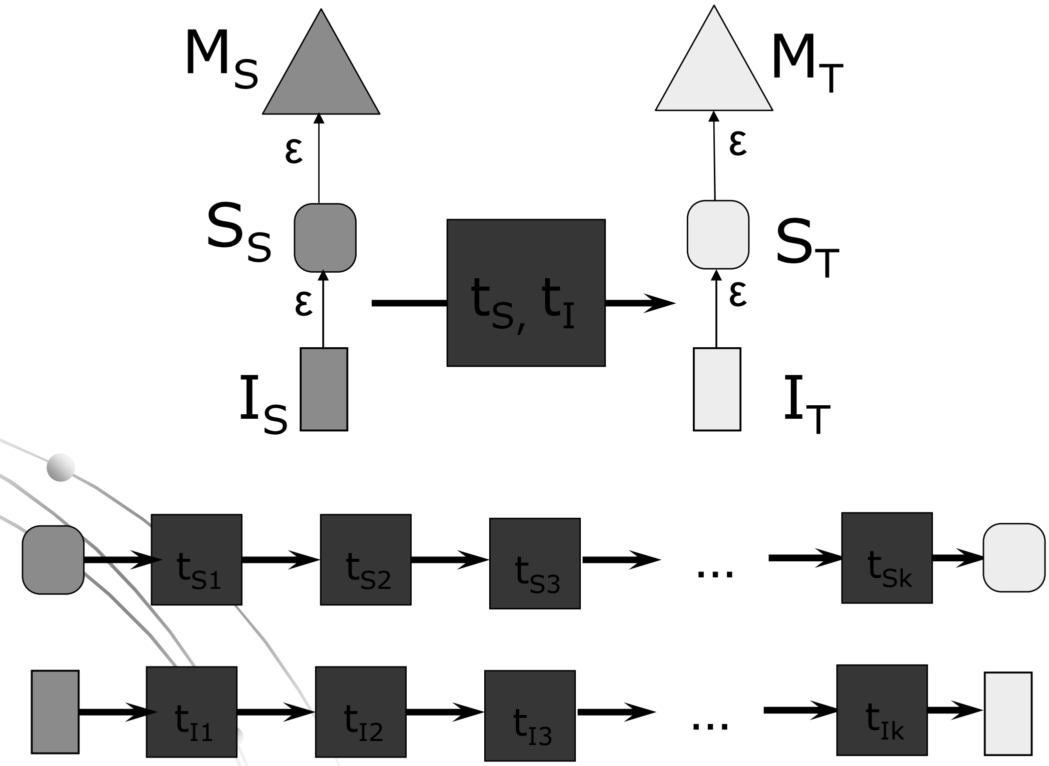
Supermodello 1

```
<META source="XSD">
 <element name="Dept">
  <sequence occurs="1:1">
   <element name="DeptName" type="string"
                           occurs="1:1"/>
   <element name="creationDate" type="date"
                           occurs="1:1"/>
   <element name="Emps" occurs="1:1">
    <sequence occurs="1:1">
     <element name="Emp" occurs="1:N">
      <sequence occurs="1:1">
       <element name="EID" type="integer"
                                  occurs="1:1"/>
       <element name="EName" type="string"
                                    occurs="1:1"/>
      </sequence>
     </element
    </sequence>
   </element>
  </sequence>
 </element>
</META>
```

All’interno del supermodello

- Esamina con il DOM lo schema in generale
- Conosce perfettamente la semantica e la sintassi delle metaprimitive
- Esegue algoritmo ricerca soluzione = sequenza di procedure necessarie per andare nel modello destinazione

Traduzione



Supermodello 2

```
<META source="Relational">
  <element name="Depts" occurs="0:N">
      <attribute name="DeptName" occurs="1:1" type="string"/>
      <attribute name="CreationDate" occurs="1:1" type="string"/>
      <attribute name="Dept-New-Key" type="key" occurs="1:1"/>
  </element>

  <element name="Emps" occurs="0:N">
      <attribute name="Depts-Emps-Key" type="string">
          <keyref name="Depts-Emps-Key-Est" refer="Dept-New-
Key"/>
      </attribute>
      <attribute name="Emps-New-Key" type="key" occurs="1:1"/>
  </element>

  <element name="Emp" occurs="0:N">
      <attribute name="Emps-Emp-Key" type="string">
          <keyref name="Emps-Emp-Key-Est" refer="Emps-New-Key"/>
      </attribute>
      <attribute name="EmpID" occurs="1:1" type="string"/>
      <attribute name="EmpName" occurs="1:1" type="string"/>
  </element>
</META>
```

Schema target

```
<database>
  <table name="Dept">
    <tuple>
      <field name="DeptName" occurs="1:1" type="string"/>
      <field name="CreationDate" occurs="1:1" type="string"/>
      <field name="Dept-New-Key" type="key" occurs="1:1"/>
    </tuple>
  </table>

  <table name="Emps">
    <tuple>
      <field name="Depts-Emps-Key" type="string">
          <keyref name="Depts-Emps-Key-Est" refer="Dept-New-
Key"/>
      </field>
      <field name="Emps-New-Key" type="key" occurs="1:1"
    </tuple>
  </table>

  <table name="Emp">
    <tuple>
      <field name="Emps-Emp-Key" type="string">
          <keyref name="Emps-Emp-Key-Est" refer="Emps-New-
Key"/>
      </field>
      <field name="Emp-New-Key" type="key" occurs="1:1" />
      <field name="EmpID" occurs="1:1" type="string" />
      <field name="EmpName" occurs="1:1" type="string" />
    </tuple>
  </table>
</database>
```

Istanza target

```
<Dept>
  <tuple>
    <DeptName>Storage</DeptName>
    <CreationDate>1999-01-07</CreationDate>
    <Dept-New-Key>sk1(Storage,1999-01-07)</Dept-New-Key>
  </tuple>
</Dept>

<Emps>
  <tuple>
    <Depts-Emps-Key>sk1(Storage,1999-01-07)</Depts-Emps-Key>
    <Emps-New-Key>1<Emps-New-Key>
  </tuple>
</Emps>

<Emp>
  <tuple>
    <Emps-Emp-Key>1</Emps-Emp-Key>
    <Emp-New-Key>sk2(37,Paul)</Emp-New-Key>
    <EmpID>37</EmpID>
    <EmpName>Paul</EmpName>
  </tuple>
  <tuple>
    <Emps-Emp-Key>1</Emps-Emp-Key>
    <Emp-New-Key>sk2(48, Andrew)</Emp-New-Key>
    <EmpID>48</EmpID>
    <EmpName>Andrew</EmpName>
  </tuple>
</Emp>
```

Istanza finale

- Data: l'istanza sorgente:
- Realizzazione dell'istanza di destinazione secondo il modello relazionale:

Istanza Sorgente (XML)

```
<Biblioteca>
 <NomeBiblio>Feltrinelli</NomeBiblio>
 <CatalogoLibri>
<Genere>Avventura</Genere>
  <Libro>
   <Titolo>Il signore degli Anelli</Titolo>
   <Autore>Tolkien</Autore>
   <Editore>Mondadori</Editore>
   <Prezzo>20.00</Prezzo>
 </Libro>
  <Libro>
   <Titolo>I Promessi Sposi</Titolo>
   <Autore>Manzoni</Autore>
   <Editore>Einaudi</Editore>
   <Prezzo>28.00</Prezzo>
  </Libro>
 </CatalogoLibri>
</Biblioteca>
```

Istanza Destinazione (Relational Model)

```
<Biblioteca>
 <tuple>
  <NomeBiblio>Feltrinelli</NomeBiblio>
  <ChiaveBiblio>Fn(Feltrinelli)</ChiaveBiblio>
 </tuple>
</Biblioteca>

<CatalogoLibri>
 <tuple>
  <Genere>Avventura</Genere>
  <ChiaveRifBiblio>Fn(Feltrinelli)</ChiaveRifBiblio>
  <ChiaveCatalogoLibri>Fn(Avventura)</ChiaveCatalogoLibro>
 </tuple>
</CatalogoLibri>

<Libro>
 <tuple>
  <ChiaveRifCatalogoLibri>Fn(Avventura)</ChiaveRifCatalogoLibri>
  <ChiaveLibro>Fn(Il Signore degli Anelli, Tolkien, Mondadori, 20.00)</ChiaveLibro
  <Titolo>Il Signore degli Anelli</Titolo>
  <Autore>Tolkien</Autore>
  <Editore>Mondadori</Editore>
  <Prezzo>20.00</Prezzo>
 </tuple>
 <tuple>
  <ChiaveRifCatalogoLibri>Fn(Avventura)</ChiaveRifCatalogoLibri>
  <ChiaveLibro>Fn(I Promessi Sposi, Manzoni, Einaudi, 28.00)</ChiaveLibro>
  <Titolo>I Promessi Sposi</Titolo>
  <Autore>Manzoni</Autore>
  <Editore>Einaudi</Editore>
  <Prezzo>28.00</Prezzo>
 </tuple>
</Libro>
```

Serializzazione

```
<table name="Employees">
   <tuple>
      <SSN>32</SSN>
      <Name>Paul</Name>
      <Dept>Sales<Dept>
      <Salary>40K<Salary>
   </tuple>
   <tuple>
      <SSN>44</SSN>
      <Name>Anne</Name>
      <Dept>Press<Dept>
      <Salary>30K<Salary>
   </tuple>
</table>
```

→

Employees

SSN	Name	Dept	Salary
32 44	Paul Anne	Sales Press	40K 30K

Esempio Query

- Rappresentazione nel modello relazionale Libro
- [illegible] XQuery [illegible] elemento atomico

```
<Biblioteca>
   <NomeBiblio>Feltrinelli</NomeBiblio>
   <CatalogoLibri>
      <Genere>Avventura</Genere>
      <Libro>
         <Titolo>Il Signore degli Anelli</Titolo>
         <Autore>Tolkien</Autore>
         <Editore>Mondadori</Editore>
         <Prezzo>20.00</Prezzo>
      </Libro>
      <Libro>
         <Titolo>I Promessi Sposi</Titolo>
         <Autore>Manzoni</Autore>
         <Editore>Einaudi</Editore>
         <Prezzo>28.00</Prezzo>
      </Libro>
   </CatalogoLibri>
</Biblioteca>
```

Controllo

```
let $l := doc("SourceIstanza.xml")//Libro[1]/parent::node()[1] return
if(name($l)) then
let $f := doc("SourceIstanza.xml")//Libro/*
where some $f in doc("SourceIstanza.xml")//Libro/* satisfies (not(exists($f/*))) return

let $a := count(doc("SourceIstanza.xml")//Libro) return
let $l := doc("SourceIstanza.xml")//Libro[1]/parent::node()[1] return

element {name($l[1]/Libro[1])}
{for $i in 1 to $a for $t in doc("SourceIstanza.xml")//Libro[$i] return
element tuple {for $c in $t/* where count($c/*)=0 return ($c),
element {concat("Chiave", name($l[1]/Libro[1]))}
{ "Fn(", for $e in $t/* where count($e/*)=0 return concat("|", string($e),"|"),")"},
element {concat("ChiaveRif", name($l[1]))}
 { "Fn(", for $c in doc("SourceIstanza.xml")//$t/parent::node() for $p in $c/* where
not(exists($p/*)) return concat("|", string($p),"|") ,")"} }}

else()
```

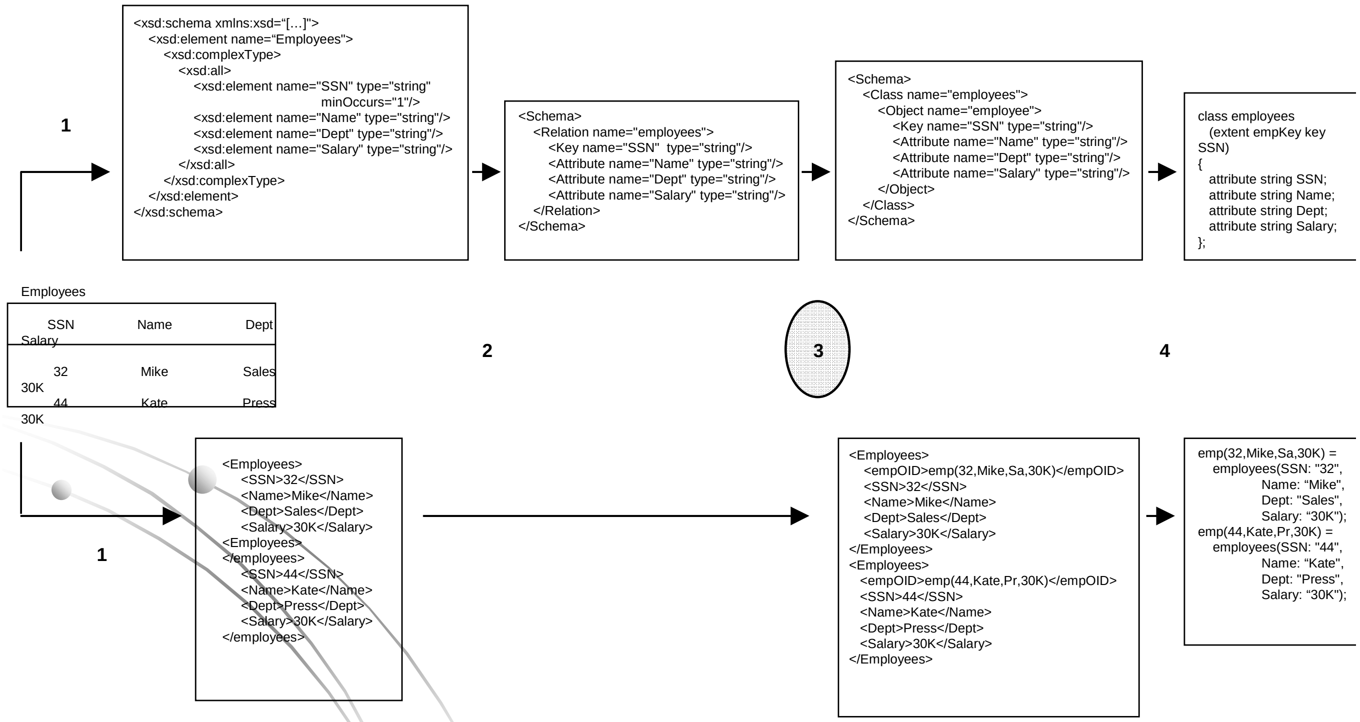
```
<Libro>
  <tuple>
   <ChiaveRifCatalogoLibri>Fn(Avventura)</ChiaveRifCatalogoLibri>
   <ChiaveLibro>Fn(Il Signore degli Anelli, Tolkien, Mondadori, 20.00)</ChiaveLibro>
   <Titolo>Il Signore degli Anelli</Titolo>
   <Autore>Tolkien</Autore>
   <Editore>Mondadori</Editore>
   <Prezzo>20.00</Prezzo>
  </tuple>
  <tuple>
   <ChiaveRifCatalogoLibri>Fn(Avventura)</ChiaveRifCatalogoLibri>
   <ChiaveLibro>Fn(I Promessi Sposi, Manzoni, Einaudi, 28.00)</ChiaveLibro>
   <Titolo>I Promessi Sposi</Titolo>
   <Autore>Manzoni</Autore>
   <Editore>Einaudi</Editore>
   <Prezzo>28.00</Prezzo>
  </tuple>
</Libro>
```

←

Libro

Traduzione

Inferenza della trasformazione



```
<xsd:schema xmlns:xsd="[…]">
   <xsd:element name="Employees">
      <xsd:complexType>
         <xsd:all>
            <xsd:element name="SSN" type="string"
                                    minOccurs="1"/>
            <xsd:element name="Name" type="string"/>
            <xsd:element name="Dept" type="string"/>
            <xsd:element name="Salary" type="string"/>
         </xsd:all>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

```
<Schema>
   <Relation name="employees">
      <Key name="SSN"  type="string"/>
      <Attribute name="Name" type="string"/>
      <Attribute name="Dept" type="string"/>
      <Attribute name="Salary" type="string"/>
   </Relation>
</Schema>
```

```
<Schema>
   <Class name="employees">
      <Object name="employee">
         <Key name="SSN" type="string"/>
         <Attribute name="Name" type="string"/>
         <Attribute name="Dept" type="string"/>
         <Attribute name="Salary" type="string"/>
      </Object>
   </Class>
</Schema>
```

```
class employees
  (extent empKey key
SSN)
{
  attribute string SSN;
  attribute string Name;
  attribute string Dept;
  attribute string Salary;
};
```

Employees

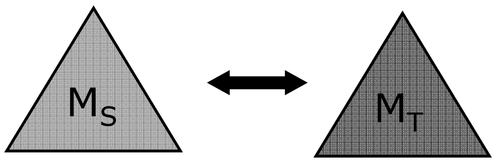
SSN	Name	Dept	Salary
32	Mike	Sales	30K
44	Kate	Press	30K

```
<Employees>
    <SSN>32</SSN>
    <Name>Mike</Name>
    <Dept>Sales</Dept>
    <Salary>30K</Salary>
<Employees>
</employees>
    <SSN>44</SSN>
    <Name>Kate</Name>
    <Dept>Press</Dept>
    <Salary>30K</Salary>
</employees>
```

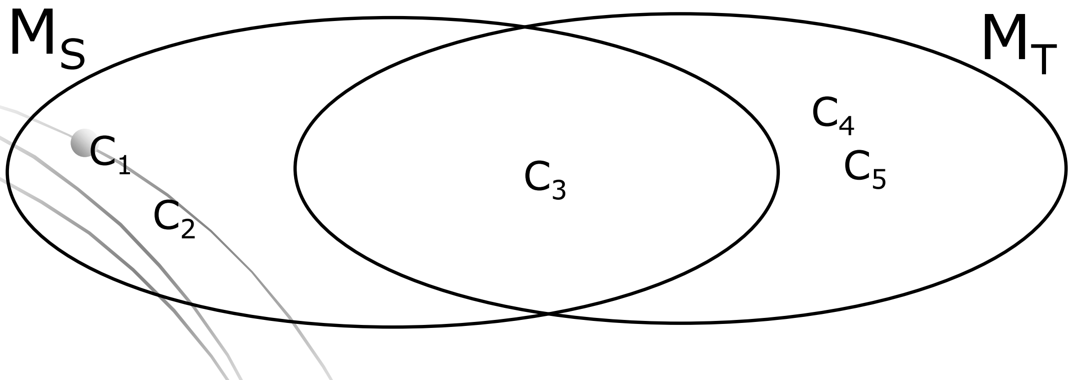
```
<Employees>
   <empOID>emp(32,Mike,Sa,30K)</empOID>
   <SSN>32</SSN>
   <Name>Mike</Name>
   <Dept>Sales</Dept>
   <Salary>30K</Salary>
</Employees>
<Employees>
  <empOID>emp(44,Kate,Pr,30K)</empOID>
  <SSN>44</SSN>
  <Name>Kate</Name>
  <Dept>Press</Dept>
  <Salary>30K</Salary>
</Employees>
```

```
emp(32,Mike,Sa,30K) =
   employees(SSN: "32",
             Name: “Mike",
             Dept: "Sales",
             Salary: “30K");
emp(44,Kate,Pr,30K) =
   employees(SSN: "44",
             Name: “Kate",
             Dept: "Press",
             Salary: “30K");
```

Model matching



Primitive (model)		**Metaprimitive** (metamodel)
All (XSD)	➡	Unordered sequence
Table (REL)	➡	Relation of lexicals



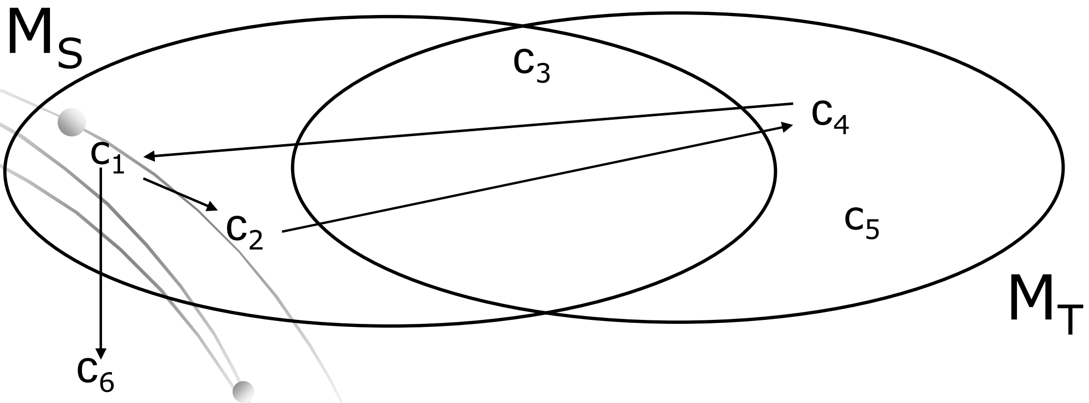
Model matching

SOURCE Scheme
-C1
-C2
-C3

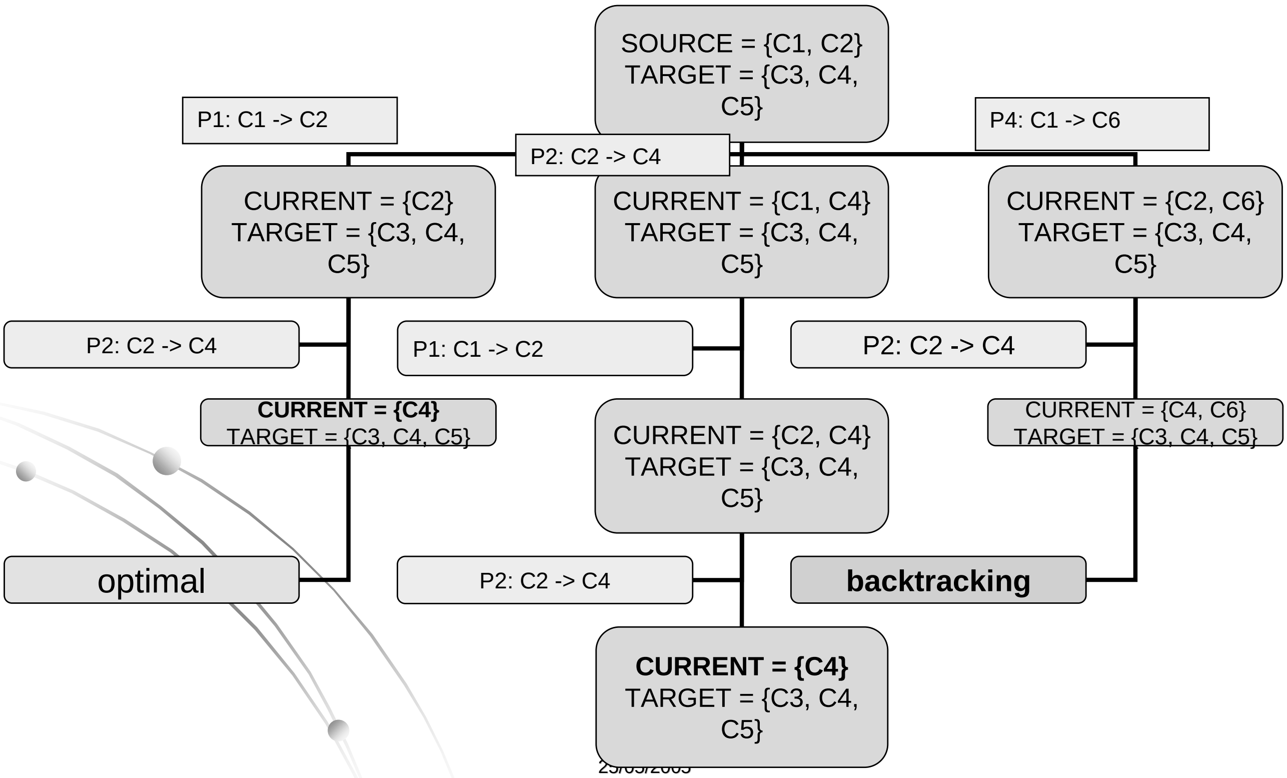
TARGET Model
-C3
-C4
-C5

PROCEDURE LIBRARY

- P1: C1 -> C2
- P2: C2 -> C4
- P3: C4 -> C1
- P4: C1 -> C6



Model matching



SOURCE = {C1, C2}
TARGET = {C3, C4, C5}

P1: C1 -> C2

P2: C2 -> C4

P4: C1 -> C6

CURRENT = {C2}
TARGET = {C3, C4, C5}

CURRENT = {C1, C4}
TARGET = {C3, C4, C5}

CURRENT = {C2, C6}
TARGET = {C3, C4, C5}

P2: C2 -> C4

P1: C1 -> C2

P2: C2 -> C4

CURRENT = {C4}
TARGET = {C3, C4, C5}

CURRENT = {C2, C4}
TARGET = {C3, C4, C5}

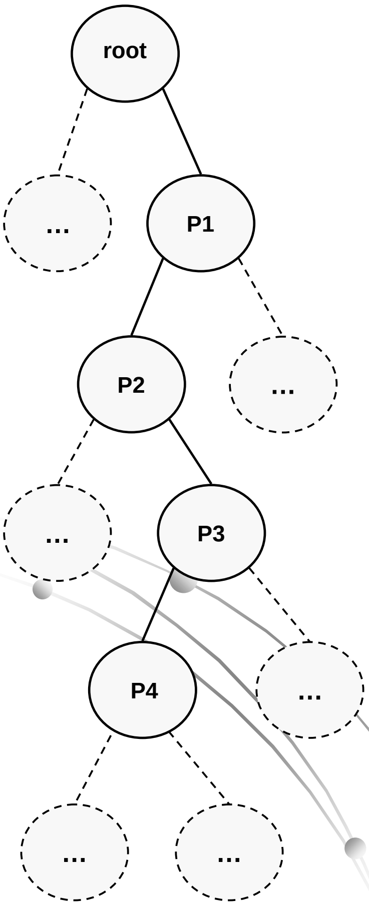
CURRENT = {C4, C6}
TARGET = {C3, C4, C5}

optimal

P2: C2 -> C4

backtracking

CURRENT = {C4}
TARGET = {C3, C4, C5}



CurrentTranslation = { }
CurrentModel = {C1, C2, C4, C5, C6}
CandidateProcedures = {P1, P4, … }

CurrentTranslation = {P1}
CurrentModel = {C2, C3, C4, C5, C6, C8}
CandidateProcedures = {P2, P3, … }

CurrentTranslation = {P1, P2}
CurrentModel= {C3, C4, C5, C6, C8, C9, C10}
CandidateProcedures = {P3, P5, … }

CurrentTranslation = {P1, P2, P3}
CurrentModel = {C3, C4, C5, C8, C9, C10}
CandidateProcedures = {P4, P10, … }

CurrentTranslation = {P1, P2, P3, P4}
CurrentModel = {C4, C5, C7, C8, C9, C10}
CandidateProcedures = {P12, … }

C1	AtomicElement
C2	NestedComplexElement
C3	FlatComplexElement
C4	Choice
C5	OrderedSequence
C6	Attribute
C7	Relation
C8	AttributeOfRelation
C9	Key
C10	ForeignKey
…	….

P1	{C1}, {C8}
P2	{C2}, {C3, C9, C10}
P3	{C6}, {C8}
P4	{C3 }, {C7}
…	….

Heuristics

- Avoid loops: verify whether the selected procedure introduces a metaconstruct that has been deleted

- Choosing the right procedure:
  - Minimize the set of constructs not allowed in the target
  - Define (partial) order between procedures
  - Assign cost functions to procedures

Problems and some solution

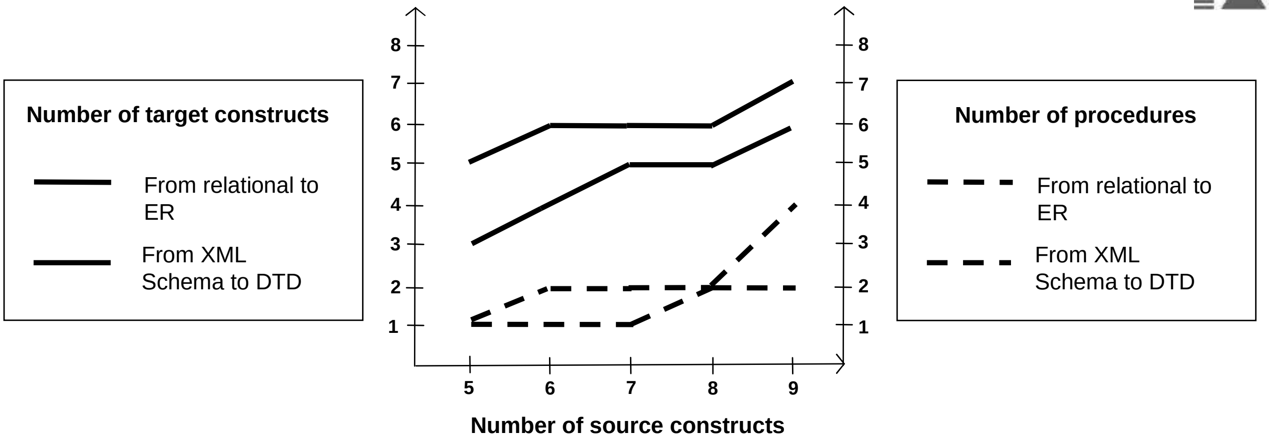
- Different translation: cost function or user choice
  - Elimination of hierarchies
- Loss of information: residual
  - Namespaces
- Degradation of information: residual
  - n-ary cardinality

Source Model	Size	Target Model	Number of Solutions	Over all time	Min. length	Solutions with min. length	Max. recall	Solutions with max. recall	Optimal solutions	First solution	
										Len.	Rec.
XMLSCHEMA	4	ODL	384	2,9	3	2	8	16	0	4	5
XMLSCHEMA	4	ER	367	3,4	1	1	6	4	0	2	6
XMLSCHEMA	5	DTD	3	2,4	1	1	3	3	1	1	3
XMLSCHEMA	5	RELATIONAL	352	2,6	3	2	5	176	2	3	5
RELATIONAL	5	XMLSCHEMA	8	1,4	3	2	5	1	1	4	5
RELATIONAL	5	ER	2	0,4	1	1	5	6	1	1	5
RELATIONAL	5	DTD	6	1,3	3	2	7	16	0	4	6
RELATIONAL	6	XMLSCHEMA	24	4,2	4	1	6	24	1	6	6
RELATIONAL	6	ODL	8	4,2	3	1	6	4	0	3	5
DTD	6	XMLSCHEMA	121	187	2	1	5	89	1	2	5
RELATIONAL	8	ODL	13	5,1	2	1	7	4	0	2	6
RELATIONAL	8	DTD	6	5,4	6	1	5	6	0	7	5

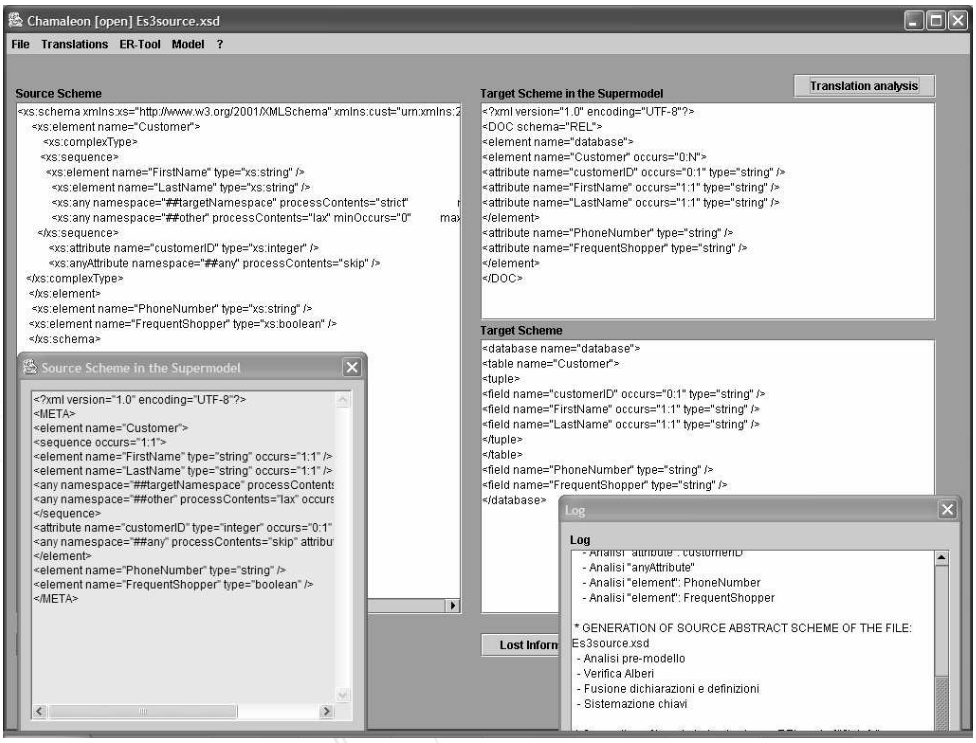
- EFFICIENZA: numero di procedure applicate

- QUALITA': numero di meta costrutti finali

UNA SOLUZIONE EFFICIENTE ED EFFICACE
E' UNA SOLUZIONE OTTIMA



Chameleon



Source Scheme

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:cust="urn:xmlns:2
   <xs:element name="Customer">
     <xs:complexType>
    <xs:sequence>
     <xs:element name="FirstName" type="xs:string" />
      <xs:element name="LastName" type="xs:string" />
      <xs:any namespace="##targetNamespace" processContents="strict"          r
      <xs:any namespace="##other" processContents="lax" minOccurs="0"      max
   </xs:sequence>
     <xs:attribute name="customerID" type="xs:integer" />
     <xs:anyAttribute namespace="##any" processContents="skip" />
 </xs:complexType>
 </xs:element>
  <xs:element name="PhoneNumber" type="xs:string" />
 <xs:element name="FrequentShopper" type="xs:boolean" />
 </xs:schema>
```

Source Scheme in the Supermodel

```
<?xml version="1.0" encoding="UTF-8"?>
<META>
<element name="Customer">
<sequence occurs="1:1">
<element name="FirstName" type="string" occurs="1:1" />
<element name="LastName" type="string" occurs="1:1" />
<any namespace="##targetNamespace" processContents
<any namespace="##other" processContents="lax" occurs
</sequence>
<attribute name="customerID" type="integer" occurs="0:1"
<any namespace="##any" processContents="skip" attribu
</element>
<element name="PhoneNumber" type="string" />
<element name="FrequentShopper" type="boolean" />
</META>
```

Target Scheme in the Supermodel

```
<?xml version="1.0" encoding="UTF-8"?>
<DOC schema="REL">
<element name="database">
<element name="Customer" occurs="0:N">
<attribute name="customerID" occurs="0:1" type="string" />
<attribute name="FirstName" occurs="1:1" type="string" />
<attribute name="LastName" occurs="1:1" type="string" />
</element>
<attribute name="PhoneNumber" type="string" />
<attribute name="FrequentShopper" type="string" />
</element>
</DOC>
```

Target Scheme

```
<database name="database">
<table name="Customer">
<tuple>
<field name="customerID" occurs="0:1" type="string" />
<field name="FirstName" occurs="1:1" type="string" />
<field name="LastName" occurs="1:1" type="string" />
</tuple>
</table>
<field name="PhoneNumber" type="string" />
<field name="FrequentShopper" type="string" />
</database>
```

Log

```
  - Analisi "attribute": customerID
  - Analisi "anyAttribute"
  - Analisi "element": PhoneNumber
  - Analisi "element": FrequentShopper

 * GENERATION OF SOURCE ABSTRACT SCHEME OF THE FILE:
Es3source.xsd
 - Analisi pre-modello
 - Verifica Alberi
 - Fusione dichiarazioni e definizioni
 - Sistemazione chiavi
```

- VLDB Demo, Berlin 2003

Progetti

- Gruppi massimo da tre persone (preferibilmente due)
  - Studio problemi all'interno del progetto (su tutti la correttezza trasformazioni)
    - Lettura articoli e verifica
  - Studio approfondito degli strumenti "concorrenti"
    - Lettura articolo, esperimenti, relazione/demo
- Progetti da concordare caso per caso a seconda degli interessi