# Rule-based Information Integration

Ander de Keijzer  Maurice van Keulen
Faculty of EEMCS, University of Twente
POBox 217, 7500 AE Enschede, The Netherlands
{a.dekeijzer,m.vankeulen}@utwente.nl

December 13, 2005

**Abstract**

In this report, we show the process of information integration. We specifically discuss the language used for integration. We show that integration consists of two phases, the schema mapping phase and the data integration phase. We formally define transformation rules, conversion, evolution and versioning. We further discuss the integration process from a data point of view.

# Contents

# 1 Introduction

Information Integration is viewing the information from multiple information sources through one uniform interface. As a result, the information sources are, from an application point of view, considered as one information source.

One of the important aspects in integrating information sources, is transforming one schema to another, allowing data to be viewed using different schemas. If multiple source schemas are transformed onto one destination schema, this can be considered as schema integration. These transformations are difficult to establish, but once specified, they can be used for all data using that schema. In this report we will explore possibilities to specify such transformations.

Many other integration systems exist, or are being developed [RB01, CGMH+94, MHH+01]. Within these projects, transformations between elements and sources is also researched [MH03, VMP03, MBDH02, RB01, HM93]. In contrast to these projects, this project has a rule-based approach to schema transformation and integration. Furthermore, integration takes place in a peer-to-peer environment.

## 1.1 Outline

In section 2 the overall process of transformation and integration is shown, definitions of important terms are given and an integration architecture is presented. In section 3 the requirements of a transformation language are discussed along with the exchange of data and schema information between peers in a network. Also, some integration scenarios are presented. Existing query languages are considered in section 4 to see if they can be used for information integration. Properties of these languages are shown and compared. In section 5 issues and future research is discussed and we conclude in section 6.

# 2 The integration process

In this section, we will discuss the overall process and architecture of the integration system. We will then show were the mapping rules are used in this process.

## 2.1 Definitions

In this section will, we will give some definitions of terms used throughout the document

**Definition 1** *An information source $i \in \mathcal{I}$, is a supplier of digital information.*

In this report all information sources will be specified in XML, i.e. an information source consists of an XML document and schema. Metadata about the information source is also specified in XML, but is not part of the actual information source.

**Definition 2** *A schema $s \in \mathcal{S}$ shows the structure of the actual information contained in the information source. A schema description $d \in \mathcal{D}$ describes the properties of the schema.*

In this document, schemas are specified using XMLSchema. A schema description is metadata about the schema and is therefore specified in XML. A schema description on an address book designed by Nokia could be

```
<schemadescription>
 <product> Address book </product>
 <company> Nokia </company>
 <version> R.3 </version>
 <description> Mobile phone address book </description>
</schemadescription>
```

**Definition 3** *All non-schema information in an information source is considered to be actual data $g \in \mathcal{G}$. An information source can now be defined by $i = (s, d, g)$.*

**Definition 4** *The estimated cost of transforming from schema $s_1$ to another schema $s_2$ is indicated by a number $c_{s_1 \to s_2} \in \mathbb{N}$, with $c_{s_1 \to s_2} > 0$. It denotes the cost, in calculation time, accuracy and precision, with which the transformation can at most be performed. If $a$ is the actual calculation time, $a \leq c_{s_1 \to s_2}$ must hold.*

**Definition 5** *Metadata $m \in \mathcal{M}$ holds additional information on transition rules. The exact information contained in $m$ varies from rule to rule, but may include*

- *elements which will be lost by the transition*
- *accuracy of the transition*

**Definition 6** *A transition rule $r \in \mathcal{R}^i$ is defined by $r = (t, m, q, c_{s_1 \to s_2})$ with*

- *$t \in \mathcal{T}^i$, a tuple $(D^i, d_r)$ indicating a transition from schema(s) with descriptions $D \subseteq \mathcal{D}$, $\#D = i$ to schema with description $d_r$,*
- *$m \in \mathcal{M}$, metadata about the transition rule,*
- *$q$, a query specifying the actual transition.*
- *$c_{s_1 \to s_2} \in \mathbb{N}$, the estimated cost of the transition.*

A transition rule transforms data in an information source from one schema to another. By composing specific rules, it is also possible to convert from one source containing both address and calendar information to one source containing only address information. Another rule, then, could transform from the initial source to an information source containing only calendar information.

**Definition 7** *A transition graph $G = (D, R)$ with*

- *$D \subseteq \mathcal{D}$, a set of schema descriptions,*
- *$R \subseteq \mathcal{R}$, a set of transition rules.*

We can now define several functions

**Definition 8** *Function $\mathsf{s} : d \mapsto s$, which maps a schema description to a schema. Function $\mathsf{d} : s \mapsto \mathbb{P}d$, which returns all schema descriptions for a given schema. Function $\mathsf{t}^{\mathcal{T}} : d \mapsto \mathbb{P}d$, which returns all schema descriptions that are directly connected to the given schema description through $\mathcal{T}$.*

**Definition 9** *An integration rule, is a transition rule $r \in \mathcal{R}^a$, with $a > 1$.*

In [dK04] we defined evolution and versioning of information sources as follows.

- Schema Evolution
  Schema evolution is accommodated when a database system facilitates the modification of the database schema without loss of existing information. When the schema of the database changes, information already contained in the database, can still be accessed, using the new schema. Also in the case of evolution, there is a distinction between materialized and a mediated. In the case of materialized, the data is converted to the new schema, whenever the schema changes. Data is always stored using the current schema. In case of mediated evolution, the data is stored using the schema used to insert the data into the database. When a query is passed to the database, the mediator transforms the schema to match the schema of the data.

- Schema Versioning
  Schema versioning is accommodated when a database system allows the accessing of all data, both retrospectively and prospectively, through user definable version interfaces. The data is stored in the database using the newest schema at the time the data was inserted into the database. Queries to the database can use any of the schemas ever used in the database. A mediator will have to transform the schema to match that of the data.

Using the previous definitions, we can give a more precise definition of both concepts.

**Definition 10** *Conversion* $conv_{d_1 \mapsto d_2}$ *between* $d_1$ *and* $d_2$ *is possible if there exists an* $x$ *such that*

- $\mathsf{t}_1^{\mathcal{T}} \ldots \mathsf{t}_n^{\mathcal{T}}$, *with* $\mathsf{t}_n^{\mathcal{T}}(\ldots(\mathsf{t}_1^{\mathcal{T}}(d_1))\ldots) = d_2$

Conversion from schema description $d_1$ to $d_2$, is possible when there exists a path in $G$ from $d_1$ to $d_2$. This path is called *conversion path*.

**Definition 11** *Evolution* $evol_{d_1 \mapsto d_2}$ *from* $d_1$ *to* $d_2$ *is supported, when*

- $conv_{d_1 \mapsto d_2}$ *is possible,*
- $\mathsf{t}_j^{\mathcal{T}}(d_1).product = d_1.product$, *for* $0 \leq j \leq x$,
- $\mathsf{t}_j^{\mathcal{T}}(d_1).version < d_1.version$, *for* $0 \leq j < x$.

Evolution is equal to conversion, but with the restriction that schema descriptions in the conversion path all have the same product name and appear in increasing product version order.

**Definition 12** *Versioning between* $d_1$ *to* $d_2$ *is supported, when*

- $evol_{d_1 \mapsto d_2}$ *is possible,*
- $conv_{d_2 \mapsto d_1}$ *is possible,*
- $\mathsf{t}_j^{\mathcal{T}}(d_2).product = d_2.product$, *for* $0 \leq j \leq x$,
- $\mathsf{t}_j^{\mathcal{T}}(d_2).version > d_2.version$, *for* $0 \leq j < x$.

Versioning is even more restrictive than evolution. In fact, versioning can be seen as evolution with the additional requirements that the inverse conversion path also exists.

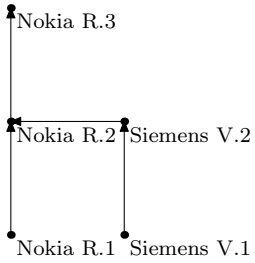


Figure 1: Transition graph address books

| name | building | room |
|---|---|---|
| Mark Hamburg | INF | 3039 |
| Ed King | ZI | 3035 |
| Joe Rough | ZI | 3035 |

(a) Data Source A

| surname | first name | location | phone |
|---|---|---|---|
| Hamburg | Mark | ZI-3039 | 3688 |
| King | Ed | ZI-3035 | 4628 |
| Friend | John | INF-3015 | 3721 |

(b) Data Source B

| name | office | phone |
|---|---|---|
| Mark Hamburg | ZI-3039 | 3688 |
| Ed King | ZI-3035 | 4628 |
| Joe Rough | ZI-3035 | NULL |
| John Friend | INF-3015 | 3721 |

(c) Integration Result

Table 1: Integration of two address books

## 2.2 An integration example

Transformation of information sources occurs at two levels. First the schemas of the data sources need to be transformed. Using this transformation, the data itself needs to be converted. Consider two data sources as shown in Table 1. Suppose the schema of the resulting data source, R, is (name, office, phone), then the following schema transformation for conversion of data source A have to be established:
$A.name \rightarrow R.name$
$A.building \rightarrow R.office$
$A.room \rightarrow R.office$

Similarly, for data source B, the conversions
$B.surname \rightarrow R.name$
$B.firstname \rightarrow R.name$
$B.location \rightarrow R.office$

$B.phone \rightarrow R.phone$
have to be composed. The last two transformation rules of information source A can be combined to $A.building + A.room \rightarrow R.office$ and the first two rules of information source B can be combined to $B.surname + B.firstname \rightarrow R.name$. Note that the plus-sign doesn't show how to convert the attribute values. It merely shows that both surname and firstname are somehow mapped onto name.

Once the schema conversions have been established, the data transformation rules can be composed. In this case, all one to one conversions are just copying the information to the resulting data source. Data source A has no information about phone numbers. In the resulting data source, the missing information will be represented by a *NULL* value. The data transformations of source A are
$R.name = A.name$
$R.office = concat(A.building, ' - ', A.room)$
$R.phone = NULL$
Note that the office attribute in the resulting information source is not just the concatenation of building and room, but also an additional dash is concatenated.

The firstname and surname from source B have to be combined into one name in source R. As was the case with the office attribute when transforming information source A, just concatenating the values is not sufficient, a space has to be inserted between the firstname and surname as well, resulting in the following conversions
$R.name = concat(B.firstname, ' ', B.surname)$
$R.office = B.location$
$R.phone = B.phone$

### 2.2.1 Schema descriptions

The actual schema and data transformation rules can only be applied when source and destination information source are known and supported by the rules. For this purpose, schema descriptions are needed. This description determines the path from source to destination information source. For example, suppose we have the following transformation rules.

| source | | destination | |
|---|---|---|---|
| product | version | product | version |
| A | 1 | A | 2 |
| A | 2 | A | 3 |
| B | 1 | B | 2 |
| B | 2 | A | 2 |

A graph, similar to the one presented in figure 1 has to be constructed. Along with the actual transition rules, metadata about the conversion and the associated schemas has to be available. This metadata is specified in XML and consists of, at least, the following components:

- The transition rule itself,

- Product name, version and identification of both source and destination schema,
- Cost factors, with which the cost of the transition can be determined. A cost factor can simply be a number, or a set of factors, which, if combined, give the cost of the transition.

The metadata associated with the rule for the conversion of the data from table 1 from source B to the result, is

```
<rule>
  <metadata>
    <cost > 1 </cost>
  </metadata>
  <schema>
    <description > Mobile phone address book </description>
    <product > Nokia </product>
    <version > R.3 </version>
  </schema>
  <schema>
    <description > Result address book </description>
    <product > User defined </product>
    <version > 1 </version>
  </schema>
  <xquery>
    <book>{
    for $s in doc(...)/book/person
    return
      <person>
        <name>{
          concat($s/firstname/text(),
          $s/surname/text())
        }</name>

        <office>{$s/location/text()}</office>
        {$s/phone}
      </person>
    }</book>
  </xquery>
</rule>
```

Both schema transformation rules and data transformation rules can be seen in the xquery. Whenever a name of an element changes, the element is reconstructed. The same holds for one to many conversions (name in the example). The name of the XML document, will be determined at query time.

## 2.3 Framework

The overall architecture of the integration system, is shown in figure 2. From the application point of view, all data (located at the bottom of the figure) is accessed through the mediator, which acts as global information source. This mediator communicates with the individual information sources through wrappers. Wrappers in our case, are simple components, that only convert the data in a uniform data model. The schema transformation is handled by the mediator.

The mediator itself, consists of a rule engine and an integrator. The rule engine translates queries posed to the mediator into queries to the underlying information sources. When data is returned, the integrator combines the data into one global result.

The mediator can use information from an external source, the so called *Oracle*, to improve the integration result. This oracle contains additional information about the real world, or the data stored in the data sources. This data can contain ontology information, indicating relations between terms or attributes. Learners [Doa02] can also be used as external information, in order to rule out, or confirm, certain mappings.

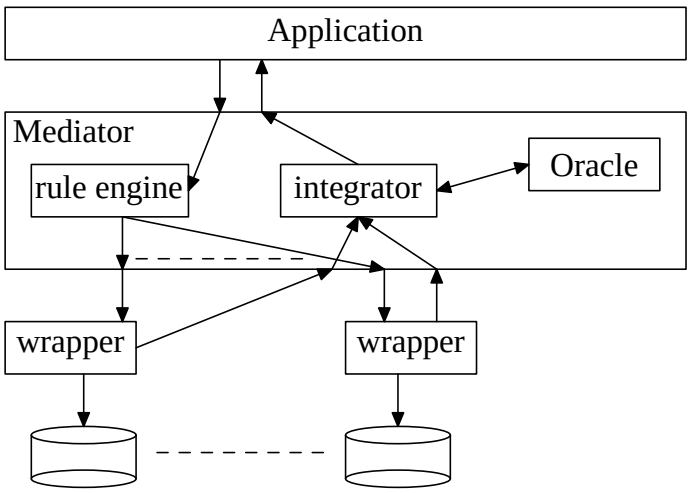


Figure 2: Integration Architecture

An instance of the integration architecture is shown in Figure 3. In this instance, both PC and PDA have multiple applications. On the PC there are two information sources available, while on the PDA there are three.

## 2.4 Example

In this section we will show a larger example, demonstrating all components in the architecture.

In this example, we will use two addressbooks. The first addressbook is shipped with a product called WhoWhere, version 2. This database is a relational one. The second addressbook is an integrated addressbook on a mobile phone produced by company Mobi. It is the first version of the addressbook database and is stored in XML. We will refer to the first addressbook as WW2 and to the second as Mobi1. The schemas for these databases are given in Figure 4.

In this example, the application used is the third version of a mobile phone addressbook application, called Mobi 3. The schema of this Mobi 3 is also shown in Figure 4. Similar to the first version of this application, the data is stored in XML format. The internal data model of the mediator is XML. As a result, there is no need for a wrapper for the Mobi 1 database. However, in order to use the WW2 database, a wrapper converting from relational data to XML, is needed for the WW2 database. At this point, both databases can be accessed by the mediator.

**evolution** At first, we will assume only data from Mobi 1 is available, while using the Mobi 3 application. A rule specifying the transformation from Mobi1 to Mobi2 and a rule specifying the transformation from Mobi2 to Mobi3 are necessary to pose queries to the database (Mobi1) using the application (Mobi3). The rule engine will combine

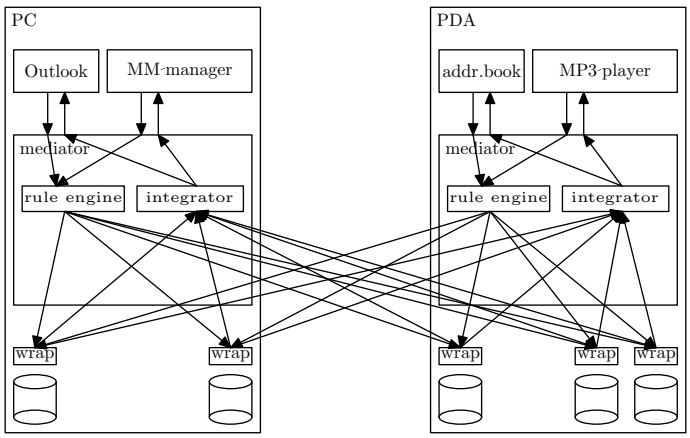


Figure 3: Integration Architecture Instance

these two rule to transform from Mobi1 to Mobi3. Both transformation rules and the combined rule from the rule-engine are shown in Figure 5. Note that in this case, there is no need for a rule to transform from Mobi1 to Mobi2, since the only difference is that in Mobi2 multiple numbers are allowed for each person. An *empty* rule is created, which merely copies the data and changes the name of the toplevel element. The rule to transform from Mobi2 to Mobi3 does have to change the schema. This rule will insert empty elements for *address* and *city*, since these elements are required for the Mobi3 database and are not present in the Mobi1 database. This rule is used by the *rule-engine*. In this case, the *integrator* has no purpose, since there is no need to change data itself.

**versioning** If we add a rule to transform information sources from Mobi 3 to Mobi 1, also versioning is possible. In this case, data is lost, since the Mobi1 schema does not allow for more than one number, addresses and cities to be stored. Also, in this case, the integrator is not used.

**integration** With integration, data from multiple information sources is combined. In this example, the application uses the Mobi 3 schema and has as underlying information sources the Mobi1 database and the WW2 database. The wrapper on top of the WW2 database will transform the data to XML. The schema is similar to the relational schema, where the toplevel element is called WW2, which has a sequence of person nodes. Each of these person nodes has as child nodes with names equal to the attributes of the original relational data. The rule-engine uses the same rules to transform the information source to a Mobi3 source. The query result of both information sources is used as input for the *integrator*.

The integrator determines if an element from the Mobi database are referring to the same person as an element from the WW database. If this is the case, the data is

Person(id, first name, lastname, phone, street, housenumber, city, zipcode)

(a) Schema of WhoWhere 2

```
< !DOCTYPE MOBI1 [
< !ELEMENT MOBI1 (person*) >
< !ELEMENT person (name, number) >
< !ELEMENT name (#PCDATA) >
< !ELEMENT number (#PCDATA) >
]>
```

(b) Schema of Mobi 1

```
<! DOCTYPE MOBI1 [
<! ELEMENT MOBI1 (person*) >
<! ELEMENT person (name, number*) >
<! ELEMENT name (#PCDATA) >
<! ELEMENT number (#PCDATA) >
]>
```

(c) Schema of Mobi 2

```
<! DOCTYPE MOBI1 [
<! ELEMENT MOBI1 (person*) >
<! ELEMENT person (name, number*, address, city)
>
<! ELEMENT name (#PCDATA) >
<! ELEMENT number (#PCDATA) >
<! ELEMENT address (#PCDATA) >
]>
```

(d) Schema of Mobi 3

Figure 4: Schemas of databases

```
<MOBI2> { for $doc in document(...)/MOBI1/person return $doc }
</MOBI2>
```

(a) Transformation rule Mobi1 to Mobi2

```
<MOBI3> { for $doc in document(...)/MOBI2/person return <person>
{$doc/name, $doc/number} <address /> <city /> </person> }
</MOBI3>
```

(b) Transformation rule Mobi2 to Mobi3

```
<MOBI3> { for $doc in (<MOBI2> { for $doc in document(...)/MOBI1/person return $doc } </MOBI2>)/MOBI2/person return <person> {$doc/name, $doc/number} <address /> <city /> </person> } </MOBI3>
```

(c) Rule-engine rule Mobi1 to Mobi3

Figure 5: Transformation rules

combined. That means that data is compared and if different, the probability of each of the possibilities is calculated and associated with the attribute-value. As a result, the output of the integrator is a *probabilistic XML* document. If certain information on a person is only present in one of the information sources, the resulting information source contains the available information without any associated probabilities.

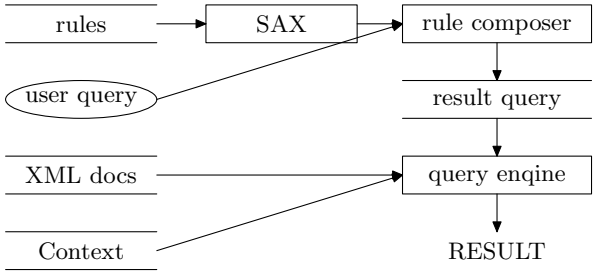


Figure 6: Architecture of rule system

# 3 Transformations

The rule engine, as schematically shown in figure 6, constructs the transition graph from the available schema descriptions. Then, it rewrites the applications query in such a way that it can be executed by the underlying information sources, using the transition graph to rewrite schemas from the information sources into the schema the application uses.

In this section, we will discuss the requirements of the transformation language. We will only consider the requirements directly related to schema transformations.

## 3.1 Requirements

In this section we will give an overview of the requirements on the transformation language. First we will consider transformation from the schema point of view, followed by the data point of view.

### 3.1.1 Schema

The mapping language should be able to

- support one to one, many to one, one to many and many to many mappings
  An element from the original information source can either have a one to one correspondence with an element in the destination information source, or only be partially related. In the latter case, it might be that more information is needed, or just the opposite, that the original element contains more information than required.

- change element names
  Elements in original and destination information source don't necessarily have the same name. This is also evident from the fact that there are not only one to one mappings, but also one to many and many to one mappings possible.

- nest (or chain) mappings
  Due to the fact that only mappings from one schema description to one other

schema description are specified, multiple transformations may be needed to transform from original to destination information source. Nesting, or chaining of mappings is needed to accomplish this total transformation.

- convert selectively
  Not all instances of the original information source may need to be transformed. There may be a restriction on the destination information source. Also not all elements of an instance may need to be transformed, because the destination could have less data per instance.

### 3.1.2 Data

We will briefly discuss data transformation, since it is greatly influenced by the schema transformation. Furthermore, although schema and data transformation are two independent actions, they are performed in a single pass and therefore the language used for schema transformation should also be able to perform data transformation.

- Language should be able to support user defined conversion rules.
- Language should (preferably) be able to perform simple operations, e.g. concatenation, adding, multiplying.

Conversions that need to be supported include

- direct transfer (name → name, or lastname → surname)
- type changes (price (int) → price (float))
- ordinal conversion (price (euro) → price (USD))
- 'simple' combinations (price * tax-level → price_inc_VAT, or concat(firstname, concat(' ', lastname)) → fullname)

## 3.2 Exchange of rules

The integration system will have to work in a peer to peer environment. In order for every device to be able to transform data, transformation rules have to exchanged between device. To minimize data transfer, rules only have to be sent when needed, i.e. if the transformation rule is not already present on the device.

Tranformation rules can be assigned an ID and a checksum can be calculated. Along with the transformation path, the IDs and checksums of the transformation rules needed can be sent to the device. If one (or more) of the transformation rules are not present, they can be obtained.

## 3.3 Conflicting data

When data is integrated, instances of the same real world object can occur in both original information sources. Of each real world object, only one instance can occur in the integrated information source. If the instances are identical in both original sources, there is no problem and one instance can be put into the resulting information source. A problem arises when the information is not identical. In this case, it could be that both instances refer to different real world objects. Consider, for example, the address book case, where we have a person called "John" in the first book and a person called "Jon" in the second. All other data, i.e. address and phone number, is identical. There are two possibilities

- Both "John" and "Jon" refer to the same person
  This could be the case if one (or both) of the names is misspelled. More general, both attribute values refer to the same *property*-value, but a typing error was made. This situation can also occur when updates have been made in one information source, but not in the other. For example, if the phone number of a person changes. In this case, the real world object, i.e. the person, doesn't change, but just one of the attributes changes.

- "John" and "Jon" are different persons
  In this case, the real world objects represented by the data are different, but the attribute values are similar. This could be the case, if the persons are, for example, room mates.

In the "John"-"Jon" case, only one of the attribute values was slighly different, but there could be many more differences in both instances, far more difficult to solve. One, easy solution to this problem is to store each instance, which is different as a separate real world object. However, this would flood the information source with information and would require the user to frequently delete incorrect instances. Another approach is to store uncertain, or probabilistic data. We will use the possible world approach, which we proposed in [dKvK04].

Consider two address books, both containing only one person. The first address book contains a person "John" with telephone number 1111 and the second address book contains a person "John" with telephone number 2222. In Figure 7 the integrated probabilistic XML document [vKdKA05] is shown. An XML node $\bigtriangledown$ indicates a probability node. This node indicates that its child nodes are mutually exclusive. Child nodes of a probability node are always $\circ$ nodes. These nodes are called possibility nodes and indicate that child nodes occur with the associated probability. The child nodes of a possibility node are dependent, i.e. they either all occur, or none of them occur. Child nodes of possibility nodes are normal XML nodes ($\bullet$).

## 3.4 Patterns

Any schema transformation can be described in terms of patterns. We can distinguish several patterns in schema transformation. All patterns are caused by the fact that schema design is a human action, with freedom of choice. Modelling an address book

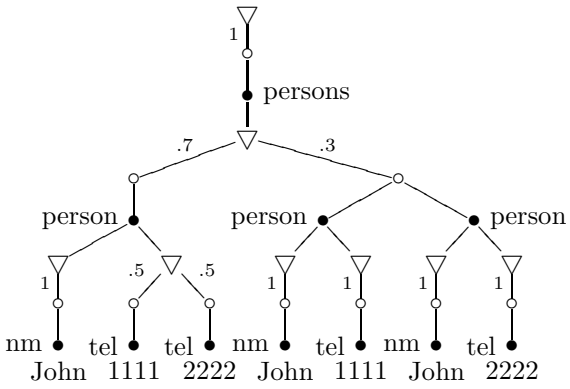


Figure 7: Example probabilistic XML tree.

in XML, for example can result in a schema where each person has a name, address and phone number, but it can also be that a person has a name and personal information, consisting of an address and phone number. We will list these patterns here. Note that this list is not exhaustive.

- Type changes (int $\mapsto$ float)
  In one information source, an element, e.g. price, may have type integer, while in another information source this element has type float.

- Domain changes (string XXXX $\mapsto$ string XXXXX)
  The domain of elements can change, while the type itself remains the same. For example, a zipcode in one information source can consist of 4 digits, while in another information source it consists of 5 digits.

- Relation changes (child-parent $\mapsto$ parent-child, parent-child $\mapsto$ siblings)
  Relations among elements can be different from one information source to another. This pattern in shown earlier in this section, where personal information in one information source was stored as following siblings of the name element, while in another address book this information was stored as child elements of a separate "personal information" node, which in turn is a following sibling of the name element.

- Schema information to data and vice versa
  Information contained within the schema in one information source can be contained in the data within another information source. For example, one addressbook contains a sequence of elements *colleague* and a sequence of elements *friends*, while another addressbook contains one sequence of elements *person* with an attribute *kind* with either value *colleague* or *friend*. In both cases persons in the addressbook are either a colleague or friend. In the first addressbook

this information is contained within the schema, while in the second addressbook this information is contained in the data.

## 3.5 Scenarios

In this section we will show some scenario's and the result of those scenario's when using our framework. Within these scenarios, the patterns of schema transformation can be recognized.

**Scenario 1** *A zipcode consists of a 4 digit number. Due to the insufficient number of available zipcodes, all codes are converted to a 5 digit zipcode. Existing zipcodes are prefixed with a 1.*

This scenario shows a change in the value of an element. These changes can be simple, like in the scenario above, or complex. This would be the case if not all zipcodes where prefi xed with the same number, but if areas were reassigned a zipcode.

**Scenario 2** *Besides housenumbers, additional information to indicate an appartement is necessary in newly built appartement blocks. Housenumbers that already exist remain the same, but are converted from integer to a string representation. Appartements obtain a number like "31-2B" indicating the appartement is located in block 31, second level, appartement B.*

Granularity changes are usually caused by changes in type. Consider an attribute sales with type float. This attribute indicates the total sales of a specifi c product. The integration result, however, has an attribute sales with an enum type and possibilities good, average, poor. As a consequence, once good, average and poor area defi ned, changing from source to destination is possible. The inverse of this operation is not possible, since information was expressed at a different, i.e. higher, level of granularity.

**Scenario 3** *Instead of having different elements for firstname and surname, the entire name is kept in one element, i.e. firstname and surname have to be concatenated.*

In this scenario elements are combined to form one resulting element. Other examples of combining are prices and VAT percentages as separate elements combined into one price element.

**Scenario 4** *A phonenumber consists of an areacode, a dash, followed by the local phonenumber. The areacode en local phonenumber are split into two seperate elements.*

This scenario shows the opposite action of the previous scenario. One element is split into multiple elements.

The previous scenarios were all about information integration. In case of versioning and evolution, the same situations occur.

**Example** Consider two information sources [LYJ04]:

```
<bibliography>
        <bib>
          <year> 1999 </year>
          <book><title> XML </title><author> Bob </author></book>
          <article><title> XML </title><author> Joe </author><author> Mary </author></article>
        </bib>
        <bib>
          <year> 2000 </year>
          <book><title> Database </title><author> Codd </author></book>
          <article><title> C++ </title><author> John <author></article>
        </bib>
</bibliography>
```

and

```
<bibliography>
        <bib>
          <book><year> 1999 </year><title> XML </title><author> Bob</author></book>
          <book><year> 2000 </year><title> Database </title><author> Codd </author></book>
        </bib>
        <bib>
          <article><year> 1999 </year><title> XML </title><author> Joe </author><author> Mary </author></article>
          <article><year> 2000 </year><title> C++ </title><author> John <author></article>
        </bib>
</bibliography>
```

Schema conversion is necessary to map source A onto B or vice versa.In fact, in this case, the structure of the schema needs to be changed. The query needed to perform the conversion is not trivial.

| | SQL | OQL | XPath | XSLT | XQuery | ℂDuce |
|---|---|---|---|---|---|---|
| selection | √ | √ | √ | √ | √ | √ |
| projection | √ | √ | − | √ | √ | √ |
| schema transformation | √ | √ | − | √ | √ | √ |
| data transformation | √ | √ | − | √ | √ | √ |
| schema check* | √ | √ | √ | √ | √ | √ |
| inputs/output | √ | √ | − | − | √ | √ |

* A check can only be made if the resulting schema is a subset of the specifi ed schema

Table 2: Aspects supported by query languages

# 4 Current languages

In this section we will discuss several existing languages used for querying and data transformation. We will compare these languages to show if and how they can be used in our data integration framework. In comparing the existing languages we will use the following aspects:

- Determine schema of output
  Can the schema of the output be determined by examining the transition query.
- Multiple inputs, one output
  Can multiple inputs be used and combined into one output.
- Selection
  How can instances from the information source be selected.
- Projection
  How can elements from a schema be selected.
- Structural transformation
  How can the schema of the result be changed, compared to the schema of the source.
- Data transformation
  How can the data in the result be changed, compared to the data in the original information source.

The languages we will consider in this report are SQL, OQL, XPath, XSLT, XQuery and ℂDuce [BCM05]. In Table 4 we have summarized the aspects supported by the languages.
From Table 4 it is apparent, that XPath and XSLT do not support all aspects. For the other languages, SQL/OQL and XQuery we will show how transformation can be accomplished, using the following example

```
<!abook[
<!ELEMENT abook (person*)>
<!ELEMENT person (fname, lname, phone, address, category)>
```

```
...
]>
```

where fname, lname, phone and address are textnodes. In case of a relational query language, we assume abook to be a relational, person a row and all other nodes to be attributes.

We assume telephone numbers to consist of an areacode (3 digits) and the actual phonenumber (4 digits) separated by dash (-). A name consists of a first name (fname) and a last name (lname), both consisting of just one word, separated by a space. In this case, the address book is intended to be used at work and contains the offices of colleaques.

The schema of the desired information source is as follows

```
<!abook[
<!ELEMENT abook (person*)>
<!ELEMENT person (name, arecode, phone, office)>
...
]>
```

but we only want to include those colleagues having a category greater than 1. The category itself is not needed in the resulting information source.

**SQL** is a declarative query language, used in relational database systems. The result returned by an SQL statement, is a relation, or table, containing the information described by the given statement. Although, SQL is standardized [SQL92], most database management systems have introduced their own version of SQL. A general SQL statement consists of the following parts

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
```

in the given order. The SELECT statement selects attributes from a relation to be in the result (projection). The FROM statement is used to indicate which tables are used. The WHERE clause specifies restrictions on the rows selected (selection). The other statements are not important to the schema transformation. Due to the number of different SQL *dialects*, where necessary, we will use the MySQL SQL dialect.

The schema transformation from the address book example, can be achieved by using the following SQL statement.

```
SELECT CONCAT(fname, ' ', lname) AS name,
       SUBSTRING(phone FROM 1 FOR POSITION('-' IN phone)-1)
           AS areacode,
       SUBSTRING(phone FROM POSITION('-' IN phone)+1)
           AS phone,
```

```
        address AS office
FROM abook
WHERE category > 1;
```

**OQL** or Object Query Language is an extension of SQL. However, its underlying datamodel is Object Definition Language (ODL), which is capable of representing objects. Some integration projects [CGMH+94, MAG+97, BBC+00] have used extensions to ODL as a datamodel as well as associated query languages [AQM+97].

**XQuery** is a full-fledged programming and query language [XQu]. It is a superset of XPath, but has a powerful additional construct, called the FLWOR expression.FLWOR is an acronym for the statements used in the construct, which are For, Let, While, Order by and Return.

```
<abook>
{
for $doc in document(...)/abook/person[./category > 1]
return
  <person>
    <name>{concat($doc/fname, ' ', $doc/lname)}</name>
    <areacode>{substring($doc/phone, 1, 3)}</areacode>
    <phone>{substring($doc/phone, 5, 4)}</phone>
    <office>{$doc/address/text()}</office>
  </person>
}
</abook>
```

## 4.1 Comparison

The language used for our integration system, preferably is an existing language, with possibly some added features. This increases the possible use of the integration system. The language used, should be powerful enough to accommodate for the requirements specified in section 3.1. If possible, the language should support user defined functions, so additional features can be included easily.

All languages, except for XPath, support projection, where one or more elements or attributes is not used in the output. With XPath it is only possible to select an entire (sub)tree and therefore, individual leaves are automatically included in the result, if their parent is included.

Selection is possible with every language. The method with which selection is specified differs from language to language, but every language supports at least some form of selection. In XPath, and therefore also XQuery, predicates are used to select (sub)trees. In SQL and OQL (the relational oriented languages) and XQuery, a WHERE clause determines whether an element is included in the result or not. In XSLT and XQuery, an if-statement can be used to selectively include elements.

Schema conversion is possible in languages, except for XPath. Nesting several transformations, however, is not possible in every language. Although some implementations of SQL/OQL support this feature, it is not supported by every DBMS. Of the other languages, only XQuery fully supports nesting.

All languages support some form of data transformations, but due to stored procedures and the possibility of user defi ned functions resp., SQL/OQL and XQuery are better suited for this purpose.

In short, Although SQL and OQL in some cases meet all the requirements, it is not the case for all implementations. XQuery in this case seems better equipped to handle both schema and data integration. Also, since our mediator internally works with XML, an XML oriented query language is the logical choice.

# 5 Issues and future work

- *Extend the Rule system to (semi-)automatically create conversion rules*
  At the moment, every rule has to be specifi ed manually. Even the smallest change in the schema, makes a change in the rule necessary. The rule engine should be able to notice small changes, e.g. due to a new version of the same application,in the schemas of underlying information and, with the help from external sources, e.g. The "Oracle", be able to create new rules, based on the already available ones.

- *Extend the Rule system to (semi-)automatically discover semantically similar elements*
  Semantically similar elements don't have to have the same name. Even worse is the case where two elements have the same name, but are semantically different, e.g. one element *number* which holds phone numbers and another element *number*, holding social security numbers. The rule engine should be able to discover if elements are semantically similar, or not.

- *Design the "Oracle"*
  The "Oracle" should, given two (or more) elements, decide if these elements are similar and to what degree. To accomplish this task, the "Oracle" can use all available internal and external information, such as schema information, ontology information, but also other attribute values.

- *Design the Probabilistic Integrator*
  When data from the information sources is returned, it still needs to be integrated. Information sources may only have partial information on a particular real world object and this has to be combined with other partial information in order to present all stored information on that particular object. When combining information, there may be doubt on whether or not the information is similar. Using the "Oracle", a probability can be assigned to indicate the level of certainty.

- *Ever-growing data set in uncertain setting*
  As we have shown in [vKdKA05], integration of probabilistic data tends to blow up the information source. Making the rule engine more intelligent will probably reduce the size of the resulting information source, but this will not be enough. Other possibilities are to involve the user and to *forget* elements with small possibilities.

# 6 Conclusion

We designed a framework for rule based information integration. First we formally defined transformation rules. Based on this definition we also defined the concepts evolution and versioning formally. The integration process is divided into two phases, the schema integration phase and the data integration phase. We recognized patterns occurring in schema transformation and showed that conflicts in data during integration can be solved using uncertain, or probabilistic data.

We have shown several existing querying languages and we identified aspects needed to support integration. Based on the aspects supported by the query languages and the fact that our integration system works with XML, we have chosen XQuery as the transformation language.

# References


[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[BBC+00] Domenico Beneventano, Sonia Bergamaschi, Silvana Castano, Alberto Corni, R. Guidetti, G. Malvezzi, Michele Melchiori, and Maurizio Vincini. Information integration: The momis project demonstration. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 611–614. Morgan Kaufmann, 2000.

[BCM05] Véronique Benzaken, Giuseppe Castagna, and Cédric Miachon. A full pattern-based paradigm for xml query processing. In Manuel V. Hermenegildo and Daniel Cabeza, editors, *PADL*, volume 3350 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2005.

[CGMH+94] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.

[dK04] Ander de Keijzer. Information integration - the process of integration, evolution and versioning. Technical report, University of Twente, 2004.

[dKvK04] Ander de Keijzer and Maurice van Keulen. A possible world approach to uncertain relational data. In *Supporting Imprecision and Uncertainty in Flexible Databases*, 2004.

[Doa02] A. Doan. *Learning to map between structured representations of Data*. PhD thesis, University of Washington, 2002.

[HM93] Joachim Hammer and Dennis McLeod. An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous database systems. *Journal for Intelligent and Cooperative Information Systems*, 2(1):51–83, 1993.

[LYJ04] Yunyao Li, Cong Yu, and H. V. Jagadish. Schema-free xquery. In *VLDB*, pages 72–83, 2004.

[MAG+97] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, and Jennifer Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, 1997.

[MBDH02] Jayant Madhavan, Philip A. Bernstein, Pedro Domingos, and Alon Y. Halevy. Representing and reasoning about mappings between domain models. In *AAAI/IAAI*, pages 80–86, 2002.

[MH03] Jayant Madhavan and Alon Y. Halevy. Composing mappings among data sources. In *VLDB*, pages 572–583, 2003.

[MHH+01] R.J. Miller, M.A. Hernandez, L.M. Haas, L.-L. Yan, C.T.H. Ho, R. Fagin, and L. Popa. The clio project: Managing heterogeneity. *ACM SIGMOD Record*, 30(1):78–83, March 2001.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, December 2001.

[SQL92] SQL. Ansi x3.135-1992, 1992.

[vKdKA05] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *Proceedings of the International Conference on Data Engineering, ICDE 2005*, 2005.

[VMP03] Yannis Velegrakis, Renée J. Miller, and Lucian Popa. Mapping adaptation under evolving schemas. In *VLDB*, pages 584–595, 2003.

[XQu] XQuery. http://www.w3.org/xml/query.