Mapping Data to Queries: Semantics of the IS-A Rule

Martin Hentschel Donald Kossmann Tim Kraska
Systems Group, ETH Zurich
{firstname.lastname}@inf.ethz.ch

Jonas Rutishauser
ETH Zurich
jonasr@ethz.ch

Daniela Florescu
Oracle
dana.florescu@oracle.com

Technical Report, November 2007

Abstract

In many application scenarios, diverse data sources produce XML messages with similar data but varying schemas. In result, data integration systems for XML data are in high demand. The technique presented in this technical report uses mapping rules to integrate diverse XML data. The introduced IS-A RULE enables a polymorphic usage of XML data items. Underlying XQuery programs operate on one main schema only while original data is not transformed, duplicated, or deleted. Techniques are shown that ensure scalability to large numbers of rules and data. Through appending additional rules, the data integration process is extensible in a pay-as-you-go-along fashion.

1 Introduction

Naturally, every company and organization holds its own, very individual stock of data. To exchange and disseminate such data it is widely agreed to use XML. Although due to the individuality of the data, these XML files follow very diverse schemas. Current standardization processes aim to ease the exchange while preserving enough flexibility to ship individual data. For example Health Level 7 (to describe clinical data) [10] and XBRL (to exchange business information) [13] define models to describe the actual data without providing explicit schema specifications. Therefore, integrating external data to be used by local applications still needs a powerful and scalable schema mapping technique.

Requirements A solution is required to query external, highly diverse XML data using the local schema only. Underlying applications reside within their local, common world for ease of use and maintenance. This requires a powerful schema mapping technique that catches and maps every piece of diversity. Since the number of data sources and therefore mappings will quickly get substantial, scalability is highly demanded. Also it should be easy to extend the system to

quickly include new data sources. Therefore, such a solution should be open for definitions of schema mappings or better: semantic rules. A large number of semantic rules must be handled even if parts of the rule set are contradictory.

Proposed Solution To meet these requirements this report introduces the so called Is-a Rule. External data is integrated by specifying mapping rules (i.e. semantic mappings between XML data items). These rules ensure that underlying applications (i.e. XQuery programs) retain to operate on the local schema. Scalability is achieved by evaluating rules lazily and extensibility is possible by adding further rules to the system. Simplified, the Is-a Rule has the form

Source is-a Target.

The *Source* gets mapped to the *Target*. More formally, the *Source* becomes a subset of the *Target*.

Contributions In summary, this report makes the following contributions:

- Examples from various domains are provided that show the power and expressiveness of the Is-a Rule. Advantages over other data integration techniques are shown.
- Syntax and semantics of the Is-a Rule are defined.
- A new data model extending the XQuery Data Model [8] is presented. This new model allows the implementation of Is-a Rules while preserving the whole functionality of XQuery.
- A working system has been implemented extending the SAXON XQuery processor [11]. These extensions are explained as well as approaches to scale to huge numbers of rules.

Current Techniques In the following, related work to integrate XML data is briefly described. A complete overview is given in Section 7.

The first approach to integrate external data is to let the application code access outside sources directly ("multiple queries"). This results in high maintenance effort since all applications have to be rewritten if a source's data format changes or if a new source has to be attached. Therefore, this technique does not scale well in the number of applications as well as the number of sources to be integrated.

The second approach to integrate external data is to transform the data to match the internal schema ("business hub", "transformations"). That way programs operate on the local schema only. This approach does not scale well for large amounts of incoming data since all data is transformed even if not needed by underlying applications. Also original data, which has not been transformed, is lost and unavailable for underlying applications.

The third approach to integrate external data is a data integration system. Such a system maps outside data to a local view. An Answering Queries Using Views (AQUV) approach translates queries operating on the local view to access outer sources. AQUV turns out to be very complex as it has been hardly

studied for XQuery so far. In particular there exists only limited experience for industrial products. Also scalability might be a severe problem if thousands of mapping statements exist.

The remainder of this technical report is organized as follows. The next section presents examples that show the power of the Is-a Rule. Section 3 defines the syntax of Is-a Rules. Section 4 presents a new data model to meet the requirements of *is-a*. Section 5 defines the semantics of Is-a Rules. Section 6 explains the current implementation as well as steps to improve performance. Section 7 reviews related work. Finally, Section 8 concludes this report.

Applying rules to data is further on referred to as the "Rules approach".

2 Examples

The following examples demonstrate the power of the Is-a Rule. These examples follow the classical data integration scenario. There exist N data sources which are queried by one subsequent program. Using the Rules approach, M rules are added to accomplish the integration process.

Another important scenario is "continuous query evaluation". There exists one data source with K subsequent applications. Here again, M rules are added preprocessing the incoming data to suit each subsequent application. An example is a streamed data source such as an RSS or news feed that is processed by different programs (publish & subscribe approach).

2.1 Data Integration

The biggest motivation for this project is data integration. XML documents from different sources most probably follow different schemas. The Is-a Rule is a powerful technique to integrate external data, while current approaches have severe disadvantages.

Given are two XML documents containing information about music (Figure 1). Their schemas differ yet the data is similar.

The task is to query all songs by "Mika". The "multiple queries" approach requires the following XQuery statements, which are combined using a union operator.

```
//music-db/tracks/track[artist eq "Mika"] |
     //jukebox/song[artist eq "Mika"]
```

The main disadvantage is the bad scalability. For every source to be integrated the query has to be rewritten which blows up the size of the query. Other current approaches (such as transformations and AQUV) exhibit disadvantages described in the introduction.

Using the Is-a Rule things become a lot easier. For this example the following rule is specified.

music-db/tracks/track is-a jukebox/song

This rule states that the set of items accessed by `music-db/tracks/track` is a subset of `jukebox/song`. Therefore it is now possible to query songs by "Mika" in both documents using one single query:

```
<music-db>
  <tracks>
    <track>
      <title>Relax</title>
      <artist>Mika</artist>
    </track>
    <track>
      <title>Flower</title>
      <artist>Eels</artist>
    </track>
  </tracks>
</music-db>
```

Document 1

```
<jukebox>
  <song>
    <title>Ring Ring</title>
    <artist>Mika</artist>
  </song>
  <song>
    <title>Lift Me Up</title>
    <artist>Moby</artist>
  </song>
</jukebox>
```

Document 2

Figure 1: Documents with similar data but diverse schemas

```
//jukebox/song[artist eq "Mika"]
```

The output is the following:

```
<track>
   <title>Relax</title>
   <artist>Mika</artist>
</track>
<song>
   <title>Ring Ring</title>
   <artist>Mika</artist>
</song>
```

Note that the song "Relax" is queried as *song* but returned as *track*. The Is-a Rule defines subset relationships but does not change or delete the types of data items. The rule states that type track is more specific than type song. Since subtype polymorphism is enabled, it is possible to work with subtype items (*tracks*) wherever supertype items (*songs*) are expected.

Advantages Duplicated code to match different schemas is avoided. Only one subsequent program is needed to query different documents. Therefore only that one program must be maintained and updated. Errors can be reduced.

Transformations are avoided. No data fragments are lost due to transforming only important data. Data that is affected by a rule is still accessible over its original path. Original data is not lost or modified. For example the song "Relax" by "Mika" is both accessible with `/music-db/tracks/track[title eq "Relax"]` or with `/jukebox/song[title eq "Relax"]`.

No new elements are introduced. Therefore the invariant

$$\texttt{count(//song | //track)} = 4 \tag{1}$$

holds before and after the rules are applied to the documents. This is important since here the Is-a Rule just states subset relationships but does not introduce new elements.

And last but not least, the types of data items are maintained. The song "Relax" maintains its type and is returned as track type. The rule states that type track is more specific than the type song.

2.2 Taxonomies

The Is-a Rule can be used to define taxonomies. Since *is-a* states a subset relationship, it is possible to specify ontology-like hierarchies. A taxonomy of quadrilaterals is illustrated by the following graph. Lower forms are special cases of higher forms.

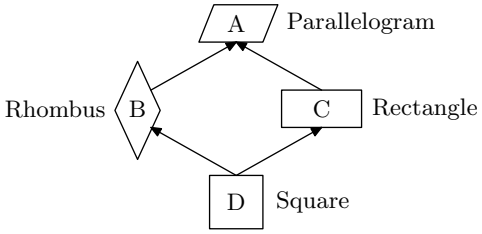


Figure 2: Geometric taxonomy

For example an XML document contains different types of quadrilaterals. The task is to retrieve all parallelograms, not just the elements matching the name "parallelogram". For an XML document shown in Figure 3 on the left, one would add the rules shown on the right. These rules match the taxonomy in Figure 2.

```
<geometry>
 <parallelogram>
  A
 </parallelogram>
 <rhombus>
  B
 </rhombus>
 <rectangle>
  C
 </rectangle>
 <square>
  D
 </square>
</geometry>
```

square is-a rectangle
square is-a rhombus
rectangle is-a
  parallelogram
rhombus is-a
  parallelogram

XML Document **Rules**

Figure 3: Defines rules for a geometric taxonomy

Now, retrieving the names of all parallelograms is possible by the query `//parallelogram/text()`. The output is

```
A B C D
```

since rectangles, rhombuses and squares are also parallelograms.

Advantages Once the rules have been applied, it is much easier to retrieve hierarchical data. The query programmer does not need to know the exact linkage between data items. Error prone code duplication is avoided (e.g. typing `//rhombus | //square` every time instead of just `//rhombus` when searching for all rhombuses).

Multiple rules are transitive (see Section 5.3). For example in Figure 3 a square is considered to be a parallelogram since the given rules state that the set of all squares is transitively a subset of all parallelograms. Therefore programmers of subsequent queries do not have to bother about transitive relationships.

2.3 Complex Integration Rules Using XQuery

Using XQuery it is possible to perform powerful mappings while still working on original data. This results in a lot of freedom to integrate external data. The following example shows how data, consisting of a different schema and different data, is integrated to match the local schema. Figure 4 shows two XML documents.

```
<cars>
 <car>
  <name>VW</name>
  <ps>150</ps>
 </car>
 <car>
  <name>Toyota</name>
  <ps>80</ps>
 </car>
</cars>
```

Document 1

```
<vehicles>
 <vehicle name="Fiat">
  <kw>50</kw>
 </vehicle>
</vehicles>
```

Document 2

Figure 4: Integrate data using XQuery

Now, cars with less than 100 ps should be mapped to vehicles. The schema of a vehicle defines the name to be an attribute and the engine's power to be stored in kilowatts. Rules to integrate cars with less than 100 ps into the vehicle database look like this.

```
cars is-a vehicles
car[ps < 100] as $c is-a
  <vehicle name="{$c/name}">
    <kw>{$c/ps * 0.74}</kw>
  </vehicle>
```

Each selected car is bound to the variable *$c*. *$c* is used by the XQuery element constructor on the right hand side to create the vehicle element. The horse power (ps) is converted into kilowatts by multiplying the horse power by 0.74.

Now, the query `/vehicles/vehicle` will return

```
<car>
   <name>Toyota</name>
   <ps>80</ps>
</car>
<vehicle name="Fiat">
   <kw>50</kw>
</vehicle>
```

The *Toyota* is returned as *car* because car is the more specific type. Nevertheless it is possible to use the *Toyota* in programs operating only on vehicle types. For example the query `/vehicles/vehicle/kw` returns

```
<kw>59.2</kw>
<kw>50</kw>
```

which shows that the *Toyota* also contains the *kw* element. It is hidden in the *Toyota*'s serialization to match the car type.

Advantages Unlike in transformations, original data items will not be copied and duplicated. Therefore the invariant

$$\texttt{count(//car | //vehicle)} = 3 \tag{2}$$

holds before and after the rules have been applied. Note that `count(//car)` + `count(//vehicle)` evaluates to 4. Here the *Toyota* is counted twice, while duplicate elimination is performed in Invariant 2.

Subtype polymorphism (see Appendix A) is enabled. It is possible to use mapped *cars* wherever *vehicles* are expected. These *cars* actually conform to two types. Therefore, the following invariant still holds after the application of the rules.

$$\texttt{count(//vehicle)} = \texttt{count(//kw)} \tag{3}$$

The Is-a Rule does not delete the typing of data items. The *vehicle*'s type is met by the XQuery expression of right hand side of the rule. However, the original car type is more specific and the typing of a *car* will still refer to its car type. Each *car* will be serialized conforming to this car type.

Opposed to transformations, it is still possible to completely work on the old schema (using the *Toyota* as *car*). Even queries that mix between the old and the new schema can be performed. (Querying the *Toyota* as *car*, but working with it as *vehicle*. Although that might not be good programming style.) That means, original data is still available for future uses not being removed as in transformations.

2.4 Data Integration Using Contexts

The specification of contexts reduces the set of nodes a rule applies to. Thus it is possible to add source specific properties to nodes. The following example shows how contexts are used to only select nodes from one particular document. Given are two XML documents containing books (Figure 5). Each price element should be appended by an attribute stating the source of the price (here Amazon or Barnes & Nobles).

```
<amazon-db>
 <book isbn="224">
  <title>Rules</title>
  <price>7.99</price>
 </book>
</amazon-db>
```

Document 1

```
<bn-db>
 <book isbn="224">
  <title>Rules</title>
  <price>6.99</price>
 </book>
</bn-db>
```

Document 2

Figure 5: The need of context specifications

Contexts are prefaced by the keyword `in`. Adding a *source* attribute to each price can be done by applying the following rules.

price in amazon-db/book is-a <price source="amazon"/>
price in bn-db/book is-a <price source="bn"/>

For each context the price elements are enhanced by the specified attribute. No new price elements are created. Also no new access paths are introduced, because the node name of the returned XQuery element equals the original name "price". Only the new *source* attributes are created with new node IDs. These attributes are then hiddenly added to the price elements. Now it is possible to run the following XQuery.

```
let $prc := //book[@isbn = 224]/price
let $min := fn:min($prc/text())
let $src := $prc[text() = $min]/@source
return <result price="$min" source="$src"/>
```

Which will output

```
<result price="6.99" source="bn"/>
```

Advantages Using contexts allows more freedom to specify which data items are affected by a rule. Using XQuery it is possible to add new properties to an existing data item. The affected item will not be duplicated. For the above example the invariant

$$\texttt{count(//price)} = 2 \tag{4}$$

will hold before and after the rules have been applied.

The new *source* attribute will not show up when serializing the price element to not break its type. However it is possible to retrieve that attribute by explicitly querying its path (e.g. `$prc/@source`). Evaluating `/bn-db/book/price` returns

```
<price>6.99</price>
```

The additional *source* attribute is not included. Therefore as opposed to XQuery Update or transformations the types will not be reset when adding new properties.

2.5 Summary

The examples showed the power of the Is-a Rule. Simple path expressions as target allow very easy integration of XML. XQuery element constructors as target are a powerful way to integrate diverse data. Context definitions allow precise statements about which data items are affected by the rule. Only one subsequent XQuery program operating on the local schema is needed to query all data sources.

As limitation, the Is-a Rule allows mappings from XML elements to XML elements only. Other mappings (e.g. mapping attributes to text nodes) are prohibited to enable clean subtype polymorphism. See Section 5.2 for details of the implementation. Appendix A explains subtype polymorphism in the context of XML.

3 Syntax

The Is-a Rule's syntax is defined as

Definition 3.1.

$$
\begin{aligned}
\mathit{ISARule} &::= \mathit{Source}\ \text{'is-a'}\ \mathit{Target} \\
\mathit{Source} &::= \mathit{SourceNode} \mid \mathit{SourceNode}\ \text{'in'}\ \mathit{PathExpr}^{1} \\
\mathit{SourceNode} &::= \mathit{PathExpr} \mid \mathit{PathExpr}\ \text{'as'}\ \mathit{VarRef}^{1} \\
\mathit{Target} &::= \mathit{TargetNode} \mid \mathit{TargetNode}\ \text{'in'}\ \mathit{PathExpr} \\
\mathit{TargetNode} &::= \mathit{SimplePath} \mid \mathit{DirElemConstructor}^{1} \\
\mathit{SimplePath} &::= \mathit{QName}^{1} \mid \text{'/'}\ \mathit{QName} \mid \\
&\qquad \mathit{SimplePath}\ \text{'/'}\ \mathit{QName}
\end{aligned}
$$

Note that *SimplePath* is a subset of an XQuery path expression.

Since *SourceNode* is an XQuery path expression it selects a set of nodes. Subsequentially this set will be called "source sequence" or "source nodes". Nodes returned by path expressions directly following the 'in' keywords are referred to as "contexts" or "context expressions".

3.1 Context Abbreviation

The above definition allows the keyword 'in' to be absent. It is not necessary to specify the context explicitly. In that case it is implicitly inserted.

Definition 3.2. *Let R be a rule* A in C is-a B in D*. If C is not declared the term "//." is inserted implicitly. If D is not declared the term "." is inserted. If C does not start with "/", it is implicitly prefixed with "//".*

[1] This syntax definition is stated in [4].

For example the rule *vehicle is-a car* is implicitly expanded to *vehicle in //. is-a car in .* before evaluation.

3.2 Namespaces

Namespaces are used in RULES in the same way as in XQuery. To be able to work with namespaces, they have to be declared explicitly. Namespaces are declared using the XQuery syntax. The order in which rules and namespaces appear is not relevant. Namespace declarations are processed first.

The advantage of declaring namespaces explicitly is that prefixes are unchangeably linked to their respective namespace URIs for a rule set. Therefore, prefixes used in rules do not depend on namespace declarations of input documents or the subsequent XQuery program.

4 Data Model Extensions

In order to better understand the effects of and to implement the IS-A RULE a different data model of XML is presented. The here presented model extends the XQuery data model [8] to allow the execution of rules. The major extensions are as follows. Nodes are accessible over various paths and nodes are allowed to have multiple parents.

A current, familiar model of XML is a tree. XML nodes are mapped to graph nodes. The children property is represented by edges between nodes. Figure 6 shows an example of an XML document represented as tree.

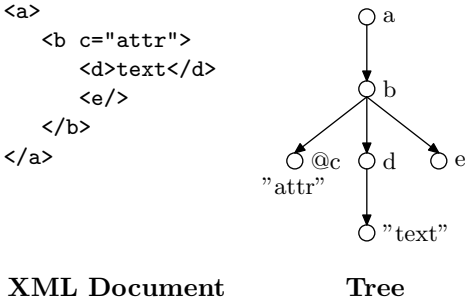
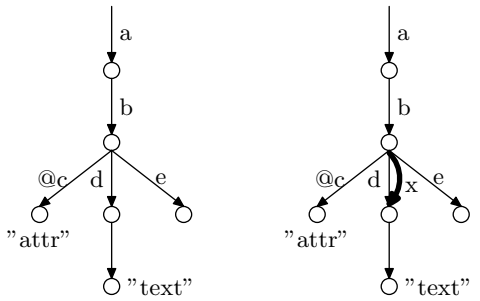


Figure 6: Familiar representation of XML

For the new model the name of a node is moved to its incoming edge. Element selection will work on edge names instead of node names. That means, a node with multiple incoming edges is able to be accessed over different paths. This change of representation enables rules to insert new access paths (i.e. new edges) without removing original ones. The original tree is converted to a graph. Figure 7a shows the new representation for the above XML document. Node names are moved to incoming edges. Content still remains within the nodes themselves (e.g. for text nodes or attributes). For serialization the element's name is set to the name of its original incoming edge.

Evaluating the rule *d is-a x* adds a new edge to the graph (see Figure 7b). Its name (more specific its QName) is "x". The new edge is thickened to highlight



(a) New tree **(b) Additional edge introduced by rule *d is-a x***

Figure 7: New representation of XML

its different role. New edges are respected during query evaluation but are not serialized as result. Element d is now also retrievable by the query `//x`. Query `//x` outputs <d>text</d>. The original name "d" is maintained since rules state subset relationships not transformations.

To not break the schema of element b the new edge x will not be printed when serializing b.

Nodes are distinguished by node identifiers. Nodes with multiple incoming edges, therefore retrievable by different queries, have one single ID. Duplicate elimination is possible. In the above example query `//d | //x` returns element d only once.

5 Semantics

In the following the semantics of the Is-a Rule are defined. As already laid out in Section 3 the rule has two different types. The semantics of the *SimplePath Rule* is defined in Section 5.1. Section 5.2 defines the semantics of the *Element Constructor Rule*. Both types of rules are connected in the way that *A is-a B* has exactly the same semantics as *A is-a <B/>* (see Theorem 2). Section 5.3 defines the semantics if multiple rules are stated.

Preface The result of an XQuery path expression is a sequence of nodes. Disregarding duplicates and ordering such a sequence can be seen as a set. Therefore if a sequence S contains the node x it holds that

$$x \in S.$$

Applying the path expression A to a node x is written as x/A. The result is a sequence. If the node y is within that sequence, it holds that

$$y \in x/A.$$

The here presented definitions and proofs make use of this mix of set theoretical formulas with XQuery path notation.

5.1 SimplePath Rule

A SimplePath rule is a rule which has a *SimplePath* expression on the right hand side.

Definition 5.1. *The rule* A in C is-a B in D*, B's syntax being a SimplePath as in Definition 3.1, is defined as*

$$\forall x, y, z(x \in C, y \in x/A, z \in x/D \Rightarrow y \in z/B).$$

If A is bound to Variable V (i.e. *A as V...*) V has no effect on the evaluation of the rule. This variable cannot be used by expression C, since variables are not allowed in *SimplePath* expressions. Also it cannot be used by expressions B and D, because these expressions do not work on single nodes selected by A. See the following "algorithm" section for further explanation of rule execution.

The forward slash in x/A and x/D means that the path expressions A and D are applied using element x as context (i.e. x being the current node in the dynamic context). The same applies for the expression z/B, B being a *SimplePath* as defined in Definition 3.1.

Subset Relationship If no contexts are specified the Is-a Rule still states a relationship. Here, all source items become a subset of the target.

Theorem 1. *The rule A is-a B states that all data items selected by $//A$ are a subset of $//B$ (i.e. the subset relation $//A \subseteq //B$ holds).*

Proof.

$$\begin{aligned}
& A \text{ is-a } B \\
\Leftrightarrow\ & A \text{ in } //. \text{ is-a } B \text{ in } . && \text{(Def. 3.2)} \\
\Leftrightarrow\ & \forall x, y, z(x \in //., y \in x/A, z \in x/. \Rightarrow \\
& \quad y \in z/B) && \text{(Def. 5.1)} \\
\Leftrightarrow\ & \forall x, y(x \in //., y \in x/A \Rightarrow y \in x/B) && (x = z) \\
\Rightarrow\ & \forall y(y \in //A \Rightarrow y \in //B) \\
\Leftrightarrow\ & //A \subseteq //B && \text{(Def. } \subseteq\text{)}
\end{aligned}$$

□

5.1.1 Algorithm / Context Selection

A sample algorithm is provided to show how the presented data model is used to implement the above definition. It further clarifies how source and context nodes are selected. Also the order in which the single expressions of a rule are executed is shown.

A rule (*A in C is-a B in D*) selects source nodes along with their respective context nodes. Then additional edges are introduced between source and context nodes. The algorithm to apply the stated definitions to the proposed data model is as follows. A rule selects tuples consisting of context nodes and source nodes. Each tuple t contains a set of context nodes C_t and a set of source nodes N_t. All tuples are contained in set T.

$$\begin{aligned}
T &= \textit{set of all tuples } t \\
t &= < C_t, N_t >
\end{aligned}$$

Step 1 Path expression C is evaluated first. According to Definition 3.2 if C is not declared, C is set to "//.". C is evaluated using the document as context. Note that C gets prefixed with "//" if it does not start with "/" (see Definition 3.2). Each so selected node creates its own tuple.

$$\forall c \in eval_{doc}(C) : < \{c\}, \emptyset > \in T$$

Step 2 Expression A is evaluated based on each c. Each tuple's N_t is set to the result of the evaluation of A.

$$\forall t \in T : \ eval_{c_t}(A) \rightarrow N_t \qquad \text{with } t = < \{c_t\}, N_t >$$

After these two steps a node c may be context to several source nodes (that is, N_t may contain several nodes). Although each source node n has exactly one context node (each C_t contains exactly one node).

Step 3 If path expression D is declared, D is applied to each c_t creating a new set C_t. Therefore it is possible that afterwards a source node n has several context nodes. If D is not declared, it is implicitly set to ".". In that case C_t remains unchanged.

$$\forall t \in T : \ eval_{c_t}(D) \rightarrow C_t \qquad \text{with } t = < \{c_t\}, N_t >$$

Now C_t as well as N_t may consist of several nodes.

Step 4 Edges are now introduced from each context node to each source node within a tuple.

$$\forall t \in T, c \in C_t, n \in N_t : \ addEdge(c, n)$$

The name of each new edge is defined by expression B.

5.1.2 Example

This example will show the different affects of context definitions. Given is the following XML document with the generated node IDs shown on the left.

```
1   <a>
2      <a>
3         <b>
4            <x/>
           </b>
5          <b>
6            <x/>
           </b>
       </a>
7      <b/>
8      <b>
9         <x/>
       </b>
    </a>
```

The node with node ID 0 is created internally. Figure 8 shows the internal data representation. This example illustrates that the following similar looking rules indeed behave differently.

$$Rule_1 = x \ in \ a/b \ is\text{-}a \ f$$
$$Rule_2 = a/b/x \ is\text{-}a \ f \ in \ a/b$$

The first rule restricts the context nodes to be within the set given by `a/b`. The second rule is more open. It states that all elements in `a/b/x` will be accessible as "f" after stepping into `a/b`.

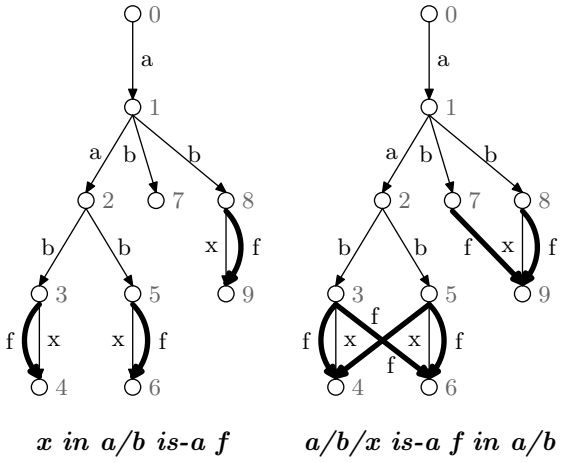


Figure 8: New edges introduced by rules 1 and 2

Applying the described algorithm yields to the following results.

Step 1 Evaluating expression C (i.e. `//a/b` and `//.` respectively):

$$T_1 = \{< \{3\}, \emptyset >, < \{5\}, \emptyset >, < \{7\}, \emptyset >, < \{8\}, \emptyset >\}$$
$$T_2 = \{< \{0\}, \emptyset >, < \{1\}, \emptyset >, \ldots, < \{9\}, \emptyset >\}$$

Step 2 Evaluating expression A (i.e. `x` and `a/b/x` resp.) based on each c_t. (Tuples that contain empty source node sets are omitted.)

$$T_1 = \{< \{3\}, \{4\} >, < \{5\}, \{6\} >, < \{8\}, \{9\} >\}$$
$$T_2 = \{< \{0\}, \{9\} >, < \{1\}, \{4, 6\} >\}$$

Step 3 Evaluating expression D (i.e. `.` and `a/b` resp.) based on each c_t. This results in the final context and source node sets.

$$T_1 = \{< \{3\}, \{4\} >, < \{5\}, \{6\} >, < \{8\}, \{9\} >\}$$
$$T_2 = \{< \{7, 8\}, \{9\} >, < \{3, 5\}, \{4, 6\} >\}$$

Step 4 Introducing new edges as shown in Figure 8. The edge names are defined by expression B.

5.2 Element Constructor Rule

An element constructor rule is a rule which has an XQuery element constructor as target. Using an XQuery element constructor instead of a SimplePath expression leads to more freedom to map data to another schema.

The example in Section 2.3 shows how the Is-a Rule states subset relationships using XQuery (*car as $c is-a <vehicle> <kw> {$c/ps * 0.74} </kw> </vehicle>*). Cars containing a *ps* element should be a subset of vehicles. A vehicle however contains *kw* elements. Therefore if cars are a subset of vehicles, then we must allow a car to be used as it was a vehicle. That means a car must be accessible through the path expression `//vehicle` (like stated in Definition 5.1). Furthermore it has to contain a *kw* element, since otherwise a later validation to the vehicle schema will fail.

Definition 5.2. *Let B be an XQuery expression returning an XML element* `<E>...</E>`*. Let this E contain the child nodes N_1 to N_n. The rule* A as V in C is-a B in D *is defined as*

$$\forall x, y, z(x \in C, y \in x/A, z \in x/D \Rightarrow y \in z/E)$$

where the nodes N_1 to N_n are added to each y. N_1 to N_n are hidden when node y is serialized. The variable V binds each y and may be used by the XQuery expression B.

Theorem 2. *A is-a <B/ > is identical to A is-a B.*

Proof.

$$\begin{aligned}
& A \text{ is-a } <B/> \\
\Leftrightarrow\ & A \text{ in } //. \text{ is-a } <B/> \text{ in } . && \text{(Def. 3.2)} \\
\Leftrightarrow\ & \forall x, y, z(x \in //., y \in x/A, z \in x/. \Rightarrow \\
& \quad y \in z/B) && \text{(Def. 5.2)} \\
\Leftrightarrow\ & A \text{ is-a } B && \text{(See [2])}
\end{aligned}$$

□

Looking back at the above example of mapping cars to vehicles, the rule's effect is visualized in Figure 9. An additional path called "vehicle" is inserted as well as a hidden new child *kw*. This matches exactly the semantics of Definition 5.2.

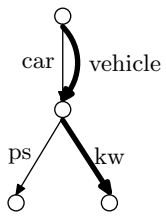


Figure 9: Vehicle as supertype of car

[2]Follows because of Definition 5.1 and that no child nodes exist in element <B/ >.

Advantages The biggest advantage of the *Element Constructor Rule* is that it is possible to exploit the power of XQuery and the XQuery function library in order to define any kind of mapping, including very complex mappings as demonstrated in Section 2.3. It is a powerful way to integrate data, while not transforming, duplicating or deleting original data items. This rule enables the integration of highly diverse data.

Only the child nodes N_1 to N_n are created as new nodes with new node identifiers. The original node y is not duplicated, preserving data consistency. In the example from Section 2.3 the evaluation of `count(//car | //vehicle)` remains constant. No duplication of original elements occurs.

Since the nodes N_1 to N_n are hidden in a serialization, the type of the original node y is not spoiled. It is not necessary to reset an item's type. For example it is possible to use a *car* as it was a *vehicle*, but *vehicle* specific children are not printed when the *car* is output.

Restrictions As already stated in Section 2.5 there are restrictions on what types of XML nodes can be mapped. It is only allowed to map XML elements to XML elements. The way the returned element is "embedded" into the original node accounts for that restriction. For example the rule *@a is-a <x><y/></x>* would make the attribute a accessible as element x and would furthermore add the child y to a. This has to be forbidden.

5.3 Multiple Rules

Multiple rules are evaluated in any order until a fixpoint is found, that is until no more changes to the presented data model occur. A rule may also use access paths introduced by other rules, former runs of other rules, or former runs of itself.

Theorem 3. *Multiple rules introduce transitive subset relationships. That means, if the rules A is-a B and B is-a C are stated, then it follows that A is-a C is also met.*

Proof.

$$
\begin{aligned}
& A \text{ is-a } B,\ B \text{ is-a } C \\
\Leftrightarrow\ & \forall x_1, y_1 (x_1 \in //., y_1 \in x_1/A \Rightarrow y_1 \in x_1/B), \\
& \forall x_2, y_2 (x_2 \in //., y_2 \in x_2/B \Rightarrow y_2 \in x_2/C) && \text{(Def. 5.1)} \\
\Leftrightarrow\ & \forall x_1, y_1, y_2 (x_1 \in //., y_1 \in x_1/A \Rightarrow \\
& \quad y_1 \in x_1/B, y_2 \in x_1/B \Rightarrow y_2 \in x_1/C) && (x_1 = x_2) \\
\Rightarrow\ & \forall x_1, y_1 (x_1 \in //., y_1 \in x_1/A \Rightarrow y_1 \in x_1/C) && \text{(See [3])} \\
\Leftrightarrow\ & A \text{ is-a } C && \text{(Def. 5.1)}
\end{aligned}
$$

□

Advantage The order in which rules are stated is irrelevant. Extensibility to integrate new data sources is highly supported, because new rules just need to be appended to the end of an existing rule set.

[3] Follows because of the transitivity of the implication ($\Rightarrow$).

5.4 Circular Dependencies

Circular dependencies are very likely to occur within large rule sets. Therefore, these dependencies must be handled by the evaluation process.

Circular dependencies emerge as follows. New nodes generated by a rule (e.g. children of an element constructor or intermediate steps of a SimplePath) may serve as input for other rules. Therefore these two rules depend on each other which will lead to infinite execution (for eager evaluation of rules). A rule may also depend on itself, if new nodes as output match the rule's input.

Given the following XML document as example.

```
<cars>
  <car ps="90">VW</car>
  <car ps="95">Ford</car>
  <car ps="170">Audi</car>
</cars>
```

The following rules depend on each other and will force an infinite loop for the above document.

car is-a slow/vehicle
slow is-a car

A new node *slow* is introduced by the first rule. This node is then made accessible as *car* by the second rule, letting the first rule create another new node *slow*. A circular dependency exists. Another example is the following rule.

car is-a <vehicle><car/></vehicle>

This rule makes a *car* accessible as *vehicle*. Also it adds a new child to each *car* that is called "car". These new *car* nodes serve as input, the rule is executed again. Here also a circular dependency exists.

One approach to deal with circular dependencies is simply to forbid them. For eager evaluation (see Section 6.1) this is necessary because of endless loops. Therefore, eager evaluation is not satisfactory. Detection of such dependencies is achieved by finding cycles in the dependency graph of rules.

To deal with circular dependencies lazy evaluation (Section 6.2) is needed. Lazy evaluation outputs results even if circular dependencies exist. Here, rules are only evaluated for data items explicitly asked for by the subsequent query.

6 Implementation

The here presented Rules approach has been completely implemented and is fully working. A demo application, which can be freely used for try outs is available online at

`http://fifthelement.inf.ethz.ch:8080/rules`.

It demonstrates the complete functioning of the here presented concepts. The current implementation extends the SAXON XQuery processor. Therefore, programs can use the full power and expressiveness of XQuery (with no limitations) while stated Is-a Rules allow complex data integration.

The next section describes eager evaluation of RULES. Section 6.2 describes how to evaluate rules lazily. Lazy evaluation is important, since it scales up to huge amounts of input data and large numbers of rules. Also circular dependencies (see previous section) can be ignored.

6.1 Eager Evaluation

Eager evaluation of RULES consists of multiple steps, which are executed sequentially. At first all XML input files are read into main memory. Internally each XML document is stored as tree using the described data model from Section 4. Afterwards all rules are applied eagerly using the algorithm from Section 5.1.1. New edges and possibly new nodes are introduced. The so manipulated tree (actually a graph now) is queried to return the final result. A chart of this workflow is shown in Figure 10.

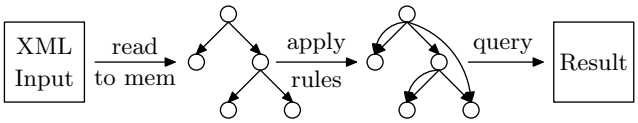


Figure 10: Eager evaluation

Querying the here presented data model is done using the Saxon XQuery engine [11] as back end. Saxon evaluates XQuery programs by heavily making use of iterators. These iterators traverse the internal tree of XML nodes step by step. For example a child iterator returns all children of a certain node, a descendant iterator returns all children, grandchildren and so on. To adopt query execution to the new data model these iterators have been rewritten to traverse the new graph along its *edges*. Node filtering is done through edge names instead of node names. Saxon's evaluation process now uses these new iterators through "wrappers" to evaluate XQuery programs as before.

Disadvantages The disadvantages of this procedure are obvious. Blocking operations delay the generation of early results. Also, all rules are evaluated for all available data. Rules may therefore also be evaluated for data not be requested by the subsequent program. Such unneeded calculations waste time.

Eager evaluation cannot be applied if circular dependencies (Section 5.4) exist. Because circular dependencies will most likely occur for large numbers of rules, eager evaluation is impractical for growing rule sets.

6.2 Lazy Evaluation

The implementation's performance is one of the most crucial factors for the RULES framework to be used as data integration tool. Therefore, performance improvements are needed such as evaluating rules lazily.

Lazy evaluation of RULES means that rules are evaluated while the subsequent program is already executed. Rule application merges with the evaluation of the XQuery program. That enables streaming execution and allows first results to be output earlier. The blocking step of eagerly evaluating rules to all data is therefore removed. The new workflow is shown in Figure 11.

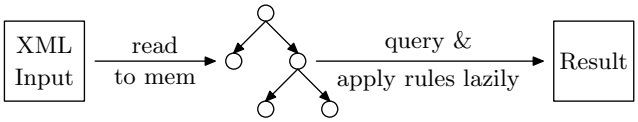


Figure 11: Lazy evaluation

The main idea of lazy evaluation is the following. Evaluation of the XQuery program starts with the original XML tree. During evaluation iterators pass by edges. These iterators trigger events while passing by an edge. Such an event is noticed by all rules. Each rule may then add new edges to the tree. New edges are always added behind the original ones to also get passed by the iterator in the same run.

A rule installs new edges to the data model if its right hand side matches a given XML structure. Also a rule now receives events of passed by edges, which can be compared to an event based XML parser. That equals the use case of event based parsing and XML filtering. Therefore a finite state machine (FSM) may be used by a rule to efficiently find matchings as in [1]. To further gain performance, it is also possible to combine source expressions of a whole rule set to create a single FSM. This approach, called "YFilter", is described in [6].

Advantages Rules are only applied when iterators pass by the respective edges. Useless applications of rules to parts of the input, which may never by queried is avoided. Therefore this technique speeds up the whole execution because the number of rule applications is decreased.

Also first results are output earlier than in the previous workflow from Section 6.1. Although the blocking operation of reading all input XML to memory still exists.

Circular dependencies (Section 5.4) and therefore endless loops when evaluating large rule sets are avoided. Therefore, lazy evaluation is again favorable for large numbers of rules.

7 Related Work

Data integration has always been a major issue in information systems, most work revolves around schema integration of relational data. Integration of XML data, especially using mappings to express semantical relationships between data items has not been examined in depth so far.

Schema Mapping [2] points out that XML schema mapping can be divided into four categories as *one-to-one* and *many-to-one* mappings, *aggregations* and *context sensitive mappings*. All of these can be handled by the Rules approach. It is summarized that current techniques use notations of mappings, which will be used to create "transformer" code. Such transformations make original data unavailable limiting the freedom of the subsequent query. The Rules approach only introduces new access paths maintaining original data. This is especially important for mappings that aggregate data, since original data items are lost after the transformation.

Answering Queries Using Views Answering Queries Using Views (AQUV) as surveyed in [9] is closely related to the RULES approach. Both techniques declare semantic mappings. In AQUV the query is transformed to meet several schemas, while the data model remains unchanged. Using the IS-A RULE does not transform the subsequent query but works on the new, here presented data model. It is believed that both approaches are equally expressive, but this has not been proven yet. It has not been shown that AQUV scales up to large numbers of mappings. Scalability for the RULES approach is achieved by lazy evaluation. AQUV has not been implemented for XQuery so far. An application demonstrating the functioning of the here presented RULES concept is available online[4].

Dataspace Management Systems iTrails [12] as part of iMemex [3] uses similar rules (called trails) to add integration semantics to dataspaces. Here, rules semantically enrich data to improve search results. These rules are added in a pay-as-you-go fashion. In contrast to the here presented RULES framework, iTrails does query rewriting to evaluate rules whereas the IS-A RULE does not transform the subsequent query but operates on the presented data model. Also iTrails lives within the iMemex data model [7] designed to describe a dataspace. The here presented work focuses on XML data exclusively.

Semantic Web (Ontologies) The Resource Description Framework (RDF) [5] adds semantics to web data items. RDF allows the expression of relationships between resources through tuples. Such a tuple consists of a subject and an object which are linked together by a predicate. Therefore, an RDF tuple may be seen as a rule expressing semantical relationships between data items. Similarly, RDF builds up ontologies as the IS-A RULE may represent taxonomies (see Section 2.2). In difference, RDF reasons about URIs not XML elements.

8 Conclusions

The IS-A RULE states subset relationships between XML elements. A rule selects source items and maps these to the rule's target expression. Thus it is possible to map XML data to another schema. Multiple rules are transitive. The resulting data may then be queried by a single XQuery program. Data integration without transformations and without using multiple similar XQuery programs is possible.

A new data model has been presented to allow the stated definitions to be implemented. For the familiar XML representation as tree, node names are moved from graph nodes to their respective incoming edges. New edges are introduced by rules converting this tree to a graph. Therefore, data items are accessible through several paths and different names. Internally, XQuery axis iterators select elements based on the names of edges. No duplication of elements is needed. Types are preserved by introducing new edges in a hidden way. Hidden edges are visible to axis iterators but are not serialized. Subtype polymorphism is enabled, such that subtype data items can be used wherever supertype data items are expected.

[4] http://fifthelement.inf.ethz.ch:8080/rules

A restriction is that elements can be mapped to other elements only. For example it is not possible to map attributes or text nodes to an element, or to map contents of an element to an attribute. Other types of rules need to be applied.

For the future such other rules mapping all types of XML nodes are needed. Improvements in performance of the current implementation have to be accomplished using lazy evaluation, indexing, and parallelism.

References

[1] M. Altinel and M. J. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *VLDB*, pages 53–64, 2000.

[2] S. Anand and E. Wilde. Mapping XML instances. In *WWW (Special interest tracks and posters)*, pages 888–889, 2005.

[3] L. Blunschi, J.-P. Dittrich, O. R. Girard, S. K. Karakashian, and M. A. V. Salles. A Dataspace Odyssey: The iMeMex Personal Dataspace Management System (Demo). In *CIDR*, pages 114–119, 2007.

[4] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0. http://www.w3.org/TR/xquery.

[5] D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/RDF.

[6] Y. Diao, P. M. Fischer, M. J. Franklin, and R. To. YFilter: Efficient and Scalable Filtering of XML Documents. In *ICDE*, pages 341–, 2002.

[7] J.-P. Dittrich and M. A. V. Salles. iDM: A Unified and Versatile Data Model for Personal Dataspace Management. In *VLDB*, pages 367–378, 2006.

[8] M. Fernández, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 Data Model. http://www.w3.org/TR/xpath-datamodel.

[9] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[10] Health Level Seven Inc. HealthLevel 7. http://hl7.org.

[11] M. Kay. Saxon XSLT and XQuery Processor. http://saxon.sourceforge.net.

[12] M. A. V. Salles, J.-P. Dittrich, S. K. Karakashian, O. R. Girard, and L. Blunschi. iTrails: Pay-as-you-go Information Integration in Dataspaces. In *VLDB*, pages 663–674, 2007.

[13] xbrl.org. XBRL eXtensible Business Reporting Language. http://www.xbrl.org.

A Subtype Polymorphism

The concept of subtype polymorphism is borrowed from the software engineering field. Subtype polymorphism allows subtype objects to be used wherever supertype objects are expected. Here, "objects" are XML elements, also referred to as data items. For the here presented RULES approach subtype polymorphism means, source data items can be used wherever target data items are expected. The programmer of an IS-A RULE therefore has to ensure that source items conform to the target's type.