Katja Moilanen, Timo Niemi,
and Mikko Kuru

An Approach for Designing and Implementing a Visual XML Dataspace System

UNIVERSITY OF TAMPERE
SCHOOL OF INFORMATION SCIENCES
REPORTS IN INFORMATION SCIENCES 16

TAMPERE 2012

UNIVERSITY OF TAMPERE
SCHOOL OF INFORMATION SCIENCES
REPORTS IN INFORMATION SCIENCES 16
DECEMBER 2012

Katja Moilanen, Timo Niemi,
and Mikko Kuru

An Approach for Designing and Implementing a Visual XML Dataspace System

SCHOOL OF INFORMATION SCIENCES
FIN-33014 UNIVERSITY OF TAMPERE

ISBN 978-951-44-9030-9

ISSN-L 1799-8158
ISSN 1799-8158

An Approach for Designing and Implementing a Visual XML Dataspace System

Katja Moilanen, Timo Niemi, and Mikko Kuru
School of Information Sciences (SIS), University of Tampere

Abstract

Dataspace systems constitute a recent data management approach to enabling better cooperation among autonomous and heterogeneous data sources with which the user is initially unfamiliar. A central idea is to gradually increase the user's knowledge about the contents, structures, and semantics of the data sources in the dataspace. Without this knowledge, the user is not able to make sophisticated queries. The dataspace systems proposed so far are usually application-specific. However, this paper introduces an application-independent dataspace system with versatile facilities based on XML sources. Our XML dataspace system provides the user with a uniform visual interface, where various interactions can be combined in declarative and intuitive way. This means, among others, that the user need not master programming techniques.

Keywords: Dataspaces, XML, visual interfaces, query languages.

1. Introduction

Various background assumptions determine the way to satisfy the user's sophisticated information needs. For example, the user's information needs may be directed towards well-integrated databases, whose structures, contents and semantics (s)he masters in detail. In this case, the user can express his/her information needs as a query specified with a query language developed for this purpose. On the other hand, the satisfaction of information needs may concern data sources which were not originally intended to co-operate, which means that their data are heterogeneous. Two possible scenarios can be distinguished. In the first scenario, both the information needs and the data sources, based on which they are satisfied, are well-known in advance. A typical example of this is the enterprise mergers and acquisitions where predictable, long-term information needs are served based on autonomously organized data sources. In this case, the data sources are integrated, for example, by applying data translation (data exchange; see [Fagin et al., 2005; Abiteboul et al., 2002]) and schema mappings (e.g. [Chawathe et al., 1994; Lenzerini, 2002]) techniques.

In the second scenario, the information needs are ad hoc and/or short-term and the data sources are beforehand unknown. An example of this is the need to compare and analyze areas, organizations, individuals, results, and so on based on the data on the Web. In this case, the available data sources are typically independent, which means that the user does not have similar control to their data as in the first scenario. In addition, the application of the data translation and data integration techniques is a costly and time-consuming process, and thus they are not an option if the information needs are short-term. Recently, dataspace systems have been proposed as a low-cost and easy-to-setup solution for enabling cooperation between autonomous and heterogeneous data sources.

A dataspace is a collection of data sources that intend to provide all of the information relevant to a particular user or task, regardless of the format of the underlying data sources or of the systems and interfaces through which they are accessed [Franklin et al., 2005; Halevy et al., 2006]. It is not likely that anyone has intimate familiarity with a dataspace comprising such autonomous and heterogeneous data sources [Howe et al., 2008]. Hence, much of the user's interaction with a dataspace system is of exploratory nature [Dong and Halevy, 2007] meaning that, through it, the user tries to find out how suitable the data sources in the dataspace are from the perspective of his/her information needs. Similarly, unlike traditional data integration systems, dataspace systems represent a co-existence approach and do not require full semantic integration of the available data sources in order to provide useful services. In fact, the central idea behind the dataspace approach is that the integration between the data sources in a dataspace is tightened in a pay-as-you-go fashion as the user's understanding about the dataspace increases over time. This kind of approach results in reduced setup time and costs, both of which have thus far hindered the widespread adoption of data integration systems [Belhajjame et al., 2010]. However, as identified in [Mirza et al., 2010; Halevy et al., 2006], the dataspace technology is still in its infancy and several major issues remain to be solved.

Franklin et al. [2005] and Halevy et al. [2006] listed the desired features that a dataspace system should implement. Among others, a dataspace system should assist the user in identifying and inter-relating the data sources in a dataspace, provide basic query mechanisms over the data sources, and provide a mechanism to trace the origin of the data sources. A dataspace system should also enable the user to focus on satisfying his/her information needs without dealing with disparately managed data. Today, there are only a few dataspace system implementations. However, they all satisfy only a subset of the above requirements. The proposed dataspace systems focus on specific applications, such as, data integration (e.g. [Das Sarma et al., 2009; Vaz Salles et al., 2007]) and searching and querying (e.g. [Howe et al., 2008; Elsayed and Brezany, 2010]), or have been tailored to specific domains, such as, personal (see [Li and Meng, 2009; Dittrich et al., 2007; Cai et al., 2005]) and scientific dataspaces (see [Elsayed and Brezany, 2010]). In this paper, we present a dataspace system, which implements most of the features listed above and is, in addition, application and domain-independent.

[Franklin et al., 2005; Halevy et al. 2006] originally envisioned that the data sources in a dataspace could employ various data models and formats. In practice, the existing dataspace system proposals are based on a specific data model (see [Dittrich and Vaz Salles, 2006; Jeffery et al., 2008; Howe et al., 2008]). They assume either that the available data sources have been organized according to this data model or that the data sources have been converted to obey it. In [Näppilä and Niemi, 2012], we propose that data sources in a dataspace are represented in XML, since XML, being a platform-independent data format, is both the leading markup language for representing structured data and unstructured documents and the de facto standard for data interchange. It is in widespread use, for example, in business-to-business (B2B) applications [Lampathaki et al., 2009] and in the Semantic Web (in the form of RDF documents [W3C RDF, 2004]). In addition, many of other data formats can straightforwardly be converted into XML. For example, all major relational database vendors (IBM, Microsoft, and Oracle) support XML publishing [Rys et al., 2005].

Our goal is to serve the end-users lacking extensive programming skills but who wish to satisfy sophisticated information needs involving several autonomous and heterogeneous XML data sources. We provide them with such a visual XML dataspace system which enables them to satisfy their information needs. As in the context of visual query languages, we aim to utilize the perceptual skills of human beings instead of specification skills typical of a textual approach. However, compared with conventional visual query languages, our situation is more complex. Namely, visual query languages typically assume that the structures, contents and semantics of the underlying data are well-known in advance, whereas in our approach the user is initially unfamiliar with the available XML sources are and the desired information needs can be satisfied by pipelining different operations. Typically, the user triggers an operation, analyzes its result and triggers additional operations. In other words, the final result is produced through several user interactions.

The dataspace system presented in this paper is centered on the visual tree metaphor, which displays the maximal tree structure of an XML document. We formally define the visual tree metaphor in terms of the XML relation representation. The XML relation representation provides unambiguous mapping between a textual XML document and the corresponding XML relation, which allows us to utilize RDBMSs for storing XML data. The formal definition of the XML relation representation and the related constructor algebra can be found in [Niemi and Järvelin, 2006; Niemi et al., 2009]). In [Näppilä et al., 2011] we designed and implemented a declarative and powerful XML query language, RXQL, based on XML relations. For the present paper, we have implemented XML dataspace profiling tools that we formally defined based on XML relations in [Näppilä and Niemi, 2012]. These tools are aimed at assisting the user in assessing the usefulness of the available XML data sources and validating their consistency by means of detecting and resolving the data conflicts in them. The starting point for the visual XML dataspace system presented in this paper is that textual XML data sources have been represented as their corresponding XML relations.

In addition, we show that these and other facilities of our dataspace system can be offered to the user declaratively and intuitively based on the visual interactions. The XML dataspace system presented in this paper is not application-specific and its use does not require programming skills or mastering XML syntax. More specifically, our goals in the present paper are to demonstrate how our XML dataspace system assists the user in

1. finding the potentially relevant XML sources,
2. selecting the relevant XML sources by providing intuition of their structures, contents, and semantics,
3. harmonizing the data sources by detecting and resolving the potential data conflicts, and
4. satisfying sophisticated information needs.

This paper is organized as follows. Section 2 discusses and considers approaches, systems and proposals related to our topic. In Section 3, we introduce some essential notations and our sample XML data sources as well as specify a visual tree metaphor for an XML source. This metaphor is utilized throughout our visual XML dataspace system. In Section 4, the visual primitives used in our system for finding relevant dataspaces are introduced. Section 5 shows how their consistency be evaluated and how the potential data conflicts can be resolved through the proposed system. In Section 6, we show how the user is able to satisfy his/her sophisticated information needs through our system. Section 7 contains discussion, and the conclusions are given in Section 8.

2. Related work

In the dataspace systems proposed so far, the data are typically represented using a data model which has been developed from the view point of the requirements of the underlying application. For example, in a personal dataspace system iMemex introduced in [Dittrich, 2009; Dittrich et al., 2007; Blunchi et al., 2007; Vaz Salles, 2007], all data are converted into its own iDM data model [Dittrich and Vaz Salles, 2006]. Only jSpace supports all data formats, [Elsayed and Brezany, 2010] . In addition, it supports indexing, searching, browsing and querying the data with the SPARQL query language [W3C SPARQL, 2008]. However, the use of SPARQL presupposes that the user masters in detail the underlying pattern matching mechanism, which means that the user needs programming skills [Niemi et al., 2011]. So far, Quarry [Howe et al., 2008] is the only implemented dataspace system that is not intended for any specific domain. Its data sources are converted into resource–property–value triples. Quarry is meant for profiling unfamiliar information sources, but so far only its searching and querying facilities have been implemented. Quarry uses a simple query language comprising a limited number of functions.

An XML dataspace system can be implemented with a computationally complete XML language, such as, XQuery [W3C XQuery, 2010]. XQuery utilizes structures borrowed from procedural programming languages, such as, the notion of the procedural variable, for loops and the if–then–else control structure, which means that the user needs to possess programming skills in order to use XQuery-like languages. On the other hand, a part of the desired functionality (e.g. querying) can be implemented with a textual query language without a computationally complete expressive power, such as, XPath [W3C XPath, 1999]. In fact, most of the textual XML query languages are based on path expressions specified in XPath. However, their usage requires that the user masters at least the pattern matching technique related to the paths. Both XQuery and XPath have been criticized as being too hard for a naive user [O'Keefe and Trotmann, 2004; Cohen et al., 2003]. In our work, we do not expect the user to have extensive programming skills. Therefore a different approach is needed.

In addition, it is characteristic of textual XML query languages that the user has to use XML notation in queries. However, it has been suggested that XML notation is not very user-friendly [Erwig, 2003]. For example, Brabrand et al. [2008] regard XML notation too verbose for the user, and they introduce a system to convert XML syntax into more a readable, non-XML form. Also visual query languages have been proposed to overcome the challenges related to textual XML query languages. Most of them use graphs in representing XML data (e.g., CQLX [Yklef and Alqahtani, 2010], XGL [Flesca, 2002] and XML-GL [Ceri et al., 1999]), whereas some are form-based (e.g., Xing [Erwig, 2003]). However, what they all have in common is that they have a visual user interface and they avoid XML syntax in query specification. Unfortunately, these visual XML query languages do not offer many of the functionalities which are necessary in an XML dataspace. For example, they typically assume that the user is familiar with the XML data sources, which is contrary to the idea of dataspaces.

Our starting point in this paper is that the available XML data sources do not have DTD [W3C XML, 2008] or XML Schema [W3C XML Schema, 2004] descriptions attached to them. There are several other reasons to discard DTDs and XML Schemas. For example, the XML sources conforming to the same DTD/XML Schema can differ greatly from each other. On the other hand, it is also possible that XML data sources with different DTDs/XML Schemas can look alike. Likewise, the schema-level manipulation of XML data is not always sufficient, because, due to its semi-structured nature, the data on the instance- and schema-level are mixed. This means, among others, that in XML sources the same label can denote both a data item name and a value. Above all, most of real-world XML data sources do not have a DTD or XML Schema [Tagarelli and Greco, 2008]. Due to the above reasons, our approach to XML dataspace systems is schemaless.

3. Visualizing XML data sources

3.1. Notations

In [Niemi and Järvelin, 2006; Niemi et al., 2009], we developed such a notion of the relation that enables any textual XML source to be converted unambiguously to the corresponding XML relation, and vice versa. In the XML dataspace system presented in this paper, the XML sources are stored and represented as XML relations. Although this representation is invisible to the user, the visualization (visible to the user) of XML sources is based on it. Therefore, we next introduce the formalism of this representation to the extent which is sufficient for understanding the definition of the visual metaphor given in Section 4. The exhaustive formalism of the XML relation representation can be found in [Niemi and Järvelin, 2006; Niemi et al., 2009].

Formally, an XML relation is a named ternary relation with the schema $D(C, T, I)$, where D stands for the name of an XML source; C expresses a data item name occurrence (i.e. an element name, an attribute name) or a value of a data item occurrence; T expresses the type of C (the alternatives are 'a', 'e' or 'v' for an attribute name, an element name, and a value, respectively); and I is an index expressing the exact and unambiguous location of C in D. The XML relation D consists of triples constructed in this way. Thus, the expression $(c, t, ind) \in D$ refers to any triple in D. In order to keep our expressions compact we use the character "_" to denote any value of any component (C, T or I) in a triple. For example, the expression $\{ind \mid (\_, \_, ind) \in D\}$ gives all the indices in D. By analyzing the values of T in the context of any triple we may decide whether C represents the schema-level information (in this case $T \in$ {'a', 'e'}) or instance-level information (in this case $T =$ 'v'). In the triples, the I component expresses the exact hierarchical order of the C component in D. In an XML source, an attribute or element name may have several occurrences, which means that an XML relation may contain triples with the identical C and T components but with the different I components. In other words, I is the key attribute of the XML relation D. If the value ($T =$ 'v') of an attribute or element occurrence is a string containing space characters, then every substring separated by a space in this string has its own unambiguous index. This is because each string may have its own semantics.

The indices in the XML relation are Dewey-style structural indices. The count of the index numbers in an index expresses at which hierarchy level the C component is in D. In our indexing scheme, the root element of D is always equipped with the index $\langle 1 \rangle$, its first immediate successor with the index $\langle 1,1 \rangle$, the first immediate successor of the former data item with the index $\langle 1,1,1 \rangle$, and so on. Structural indices allow us to effectively access the ancestors and descendants of a given data item occurrence. All the indices in an XML relation have the same first index number 1, indicating that they all depend on the same root element. Index manipulation plays an important role in processing XML relations. Here, we need only the expression for referring to the last index number and the rest of the index numbers in an index. In the expression index = $\langle part \perp i \rangle$, i refers to the last index number and $part$ is the index which one obtains by removing the last index number from *index*. Thus, for example, the expression $\langle ind \mid j\rangle$ applied to the index ⟨1,2,3,4⟩ means that *ind* = ⟨1,2,3⟩ and *j* = 4. Analogously, if *ind* is the index ⟨1,2,3⟩ and *j* = 4, then the expression $\langle ind \perp j\rangle$ refers to the index ⟨1,2,3,4⟩.

XML documents are often modeled as ordered labeled trees. In the tree model, the leaf nodes represent data items associated with values. For convenience, we call a leaf data item a data item, which has only values but no own substructures. Next, we define the function leaf(*ind*), which is true if the index *ind* is associated with a leaf data item in the XML relation *D*, as follows:

$$\text{leaf}(ind) = \begin{cases} true, & \text{if } (\_, t, ind) \in D \land t \in \{\text{'a'}, \text{'e'}\} \mid \forall (\_, t', \langle ind \perp i\rangle) \in D \mid t' = \text{'v'} \\ false, & \text{otherwise} \end{cases} \qquad (1)$$

For example, in the data source *short_publications_4* (see Appendix A) the function leaf(⟨1,2⟩) returns *false*, because the index ⟨1,2⟩ is attached to the tuple (‘publisher’, ‘e’, ⟨1,2⟩) and its immediate successors in the document hierarchy ((‘name’, ‘a’, ⟨1,2,1⟩) and (‘writer’, ‘e’, ⟨1,2,2⟩)) are not associated with the values. On the other hand, in the context of this same data source, the function leaf(⟨1,2,2,1⟩) is *true*. This is due the fact that the immediate successors of the data item corresponding to the tuple (‘name’, ‘a’, ⟨1,2,2,1⟩) are values (the tuples (‘Charlotte’, ‘v’, ⟨1,2,2,1,1⟩) and (‘Lewis’, ‘v’, ⟨1,2,2,1,2⟩)).

3.2. Visual Tree Metaphor

It would be possible to build an XML dataspace system based on the textual XML documents. However, from the view point of the user this would be undesirable. There are, at least, three basic reasons for this. First, XML is a very redundant data format, which means that XML data sources easily become large. This is due, among others, to the fact that the start and end tags need to be repeated for each occurrence of a specific element. Second, XML elements are often deeply nested which means that the end tag may appear far from the corresponding start tag. In other words, it is hard for the user to get a clear understanding of the underlying XML structure. Third, as noted in [Erwig, 2003], XML itself is not a user-friendly notation. Therefore, the starting point in our XML dataspace system is that it does not require direct interaction with the XML syntax.

An XML source should be visualized to the user so that (s)he can select by simply pointing and clicking the data which (s)he is interested in. Due to the semi-structured nature of XML, the data representing the schema-level (i.e. element and attribute names) are intermingled with the instance-level data (i.e. element and attribute values) in XML sources. The user’s interpretation on an underlying XML source can be supported by visualizing the schema and instance levels separately. The schema-level visualization displays only data item names and the relationships among them in the XML source at hand. Based on this visualization, the user gets a reduced view on both the content and structure of the underlying XML source. In order to get more detailed knowledge on the content, the user can by pointing and clicking get all value occurrences of any data item name in this schema visualization.

From the view point of the user, the tree layout is a compact and intuitive way to visualize hierarchical data, like XML. Therefore, in our system the schema-level of an XML source is visualized as a tree. The visual representation of a tree consists only of two symbols: a node and an edge. In our system, a node is displayed as a rectangle labeled by the name of the corresponding data item. The mapping between the data model and the visual model is called a visual metaphor [Haber et al., 1994]. In our case, the visual metaphor defines how the schema visualization is produced from the XML relation representation of an XML source. Next, we consider this process in detail.

The visual metaphor is quite straightforward for a structured data model, such as the relational model. This is due to fact that a structured data model has a separate and fixed schema to which the instance level conforms. The situation is more complicated in our approach, in which unfamiliar XML sources have no schemata. In our case, the schema-level visualization can be constructed only by finding out the components belonging to the schema level and their internal relationships among the semi-structured XML data. A special challenge for the visualization of XML data is that the occurrences of a data item might be organized differently.

In an XML source those data items, which depend immediately on the data item with a specific name, can vary considerably between their occurrences. In the tree visualization of the schema, we apply the maximum principle: we visualize as the children of a data item all those different data items which may depend immediately on this data item in some of its occurrences. For example, let us assume that a data item with the name *A* has the following three occurrences. In its first occurrence the data items with the names *B*, *C* and *D* depend immediately on *A*, in the second occurrence the data items with the names *D* and *E* depend immediately on *A* and in the third occurrence the data items with the names *B*, *D* and *F* depend immediately on *A*. In this case, the data items with names *B*, *C*, *D*, *E* and *F* are visualized as the children of *A* although there is no occurrence of *A* in which these data items would appear at the same time. However, this kind of visualization gives the user an overview on data items and their relationships in the XML source of interest. Next, we demonstrate how the XML relation representation supports visualization and give a recursive algorithm for the implementation of the visual metaphor. It constructs the XML tree visualization in a top-down fashion from an XML relation representation of a XML source.

INPUT: The XML relation *XRel*.
OUTPUT: The tree visualization of the schema of *XRel*.
ALGORITHM: Find the root *r* as follows $(r, 'e', \langle 1 \rangle) \in XRel$ and draw a rectangle with the name *r*. The visualization starts by finding out all different data item names depending immediately on the root *r* i.e. by calling the procedure $visualize(r, \{\langle 1 \rangle\}, XRel)$.

DEFINITION OF PROCEDURE $visualize(DN, I\text{-}SET, XRel)$:

Step 1:

Construct the set Z that contains all data item indices which depend immediately on different occurrences of DN whose indices are in I-SET.

$$Z = \bigcup_{ind \in I-SET} \left\{ (ind \perp t) \,\middle|\, (\_, t, (ind \perp t)) \in XRel \bigwedge \square t \in \{'a', 'e'\} \right\}$$

Construct the set N that contains all different data item names which are associated with the indices in Z.

$$N = \bigcup_{ind \in Z} \left\{ name \,\middle|\, (name, t, ind) \in XRel \bigwedge \square t \in \{'a', 'e'\} \right\}$$

Construct the set SET1 that contains pairs in which the first argument expresses a data item name which depends immediately on DN in one or more its occurrences expressed in I-SET. If a data item name depending immediately on DN has several occurrences in XRel, their all indices are expressed in the second argument of a pair in SET1.

$$SET1 = \bigcup_{n \in N} \left\{ \left(n, \left\{ ind \,\middle|\, ind \in Z \bigwedge \square (n, \_, ind) \in XRel \right\} \right) \right\}$$

Step 2: Visualize a hierarchical level as follows. Draw the edges from the rectangle with the name *DN* to each first argument (i.e. a data item name) in the pairs of *SET1*. Data item names are visualized as rectangles labeled by these names.

Step 3: Construct the set *SET2* that contains only those data item names with their indices which have substructures. *SET2* is constructed from *SET1* by removing leaf (function is defined in Section 3.1) data item names from *SET1* because they do not have substructures and they have been visualized in Step 2.

$$SET2 = \{(name, Ind\text{-}set1) | (name, Ind\text{-}set) \in SET1 \wedge Ind\text{-}set1 \neq \emptyset\}$$

where *Ind-set1 = select(Ind-set)* and the function *select(Ind-set)* is defined as follows

$$select(Ind\text{-}set) = \{ind | ind \in Ind\text{-}set \wedge \neg leaf(ind)\}$$

Step 4: If *SET2* $= \emptyset$, it means that only leaf data item names depend immediately on *DN*. If *SET2* $\neq \emptyset$, it means that a new hierarchical level depending on *DN* must be constructed

If *SET2* = $\emptyset$, then stop visualization else call for each pair *(ename, Indset)* in *SET2* the procedure *visualize(ename, Indset, XRel).*

The rectangles in the visualization contain also some instance-level information. In the context of every data item name, the number of its occurrences in the XML data source is given. In addition, the probability (frequency) for, how often a data item occurrence with a specific name appears in the context of the occurrences of its parent data item, is expressed. The information related to the labels 'Instances' and 'Frequency' is defined by the formulas *count(DN,name)* and *percent(DN, name)* respectively. In the formulas variables *name*, *ind*, *Ind-set*, *I-SET* have the same meaning as in the procedure *visualize*.

$$count(DN, name) = |Ind\text{-}set| \text{ , when } (name, Ind\text{-}set) \in SET1$$

$$percent(DN, name) = 100 \times \frac{|\{ind \mid ind \in I\text{-}SET \wedge (name, Ind\text{-}set) \in SET1 : \exists (ind \perp t) \in Ind\text{-}set\}|}{|I\text{-}SET|}$$

In these formulas it is worth noting that *SET1* depends on *DN* and its indices *(I-SET)*. It is possible that in an XML source the parents of the data items with the same name have different names or even if they have the same name, they can appear on different levels. For example, in Figure 4.1 the parents of the data item **name** can be the data items **publisher**, **writer**, **article**, **book** and **artist**. Therefore it is necessary to give as an argument the parent of the data item of interest. The number produced by applying the function *percent(DN, name)* is the percentage of those parent data item occurrences which have at least one data item occurrence with the name **name** as a child. The information produced by *count* and *percent* assists the user in estimating how often a specific data item occurs in the underlying XML source and how regular the structure of the underlying XML source is.

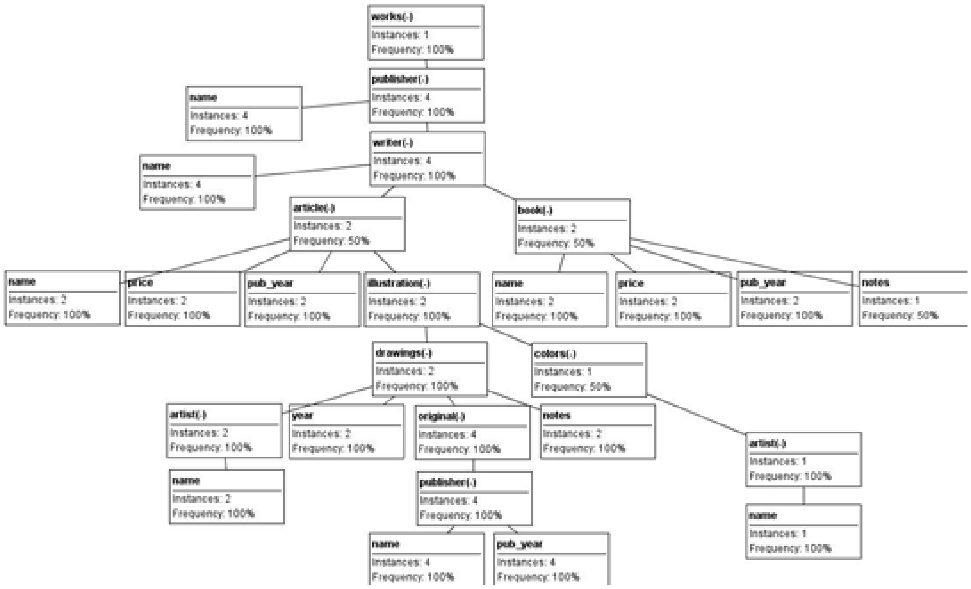


Figure 3.1 The visualization of the data source short_publications_4.xml

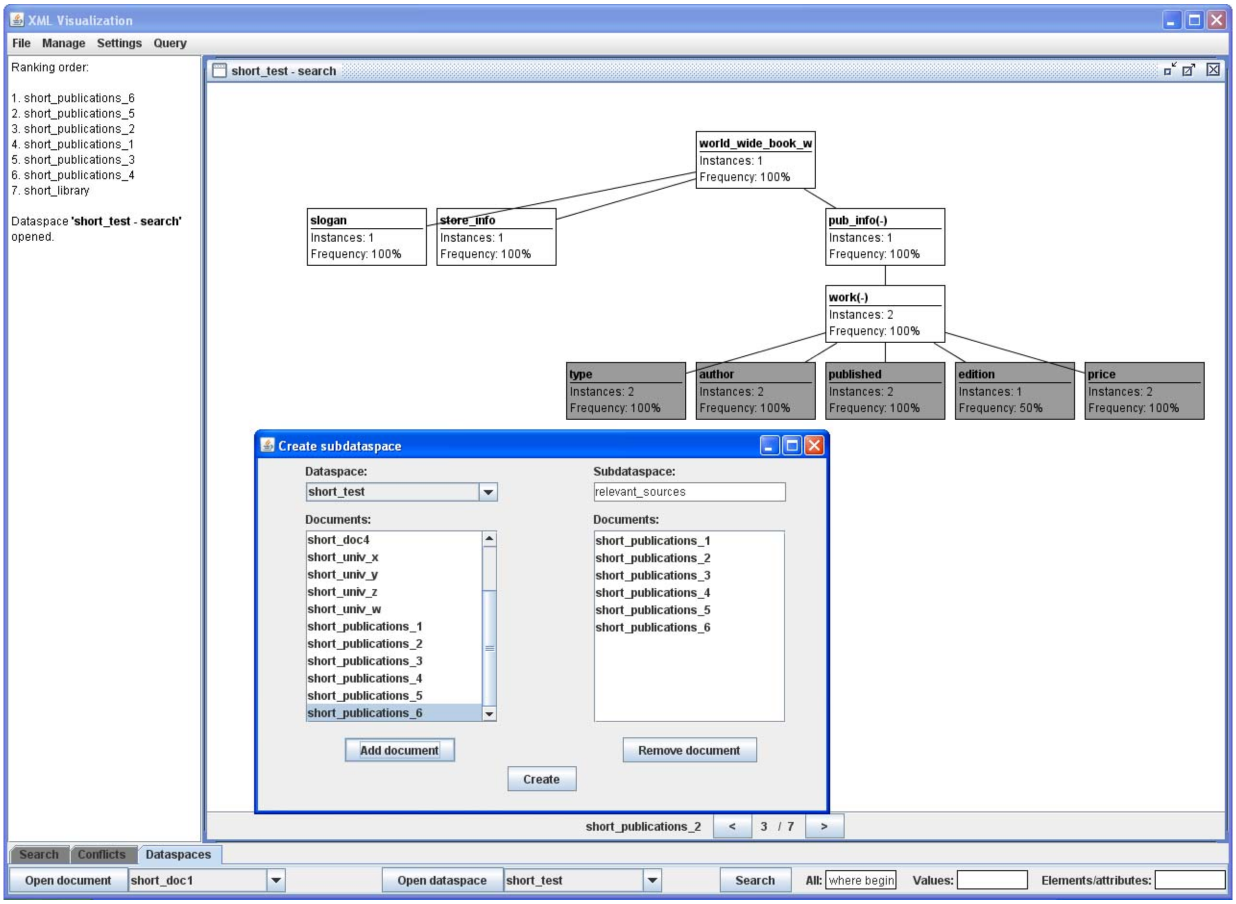
4. The visual primitives for finding the relevant XML sources from a dataspace

In our XML dataspace system, the user interacts with XML data sources only through their visualizations. The use of our dataspace system is a typical pipeline session. The user repeats several times the sequence: select some available operation tailored to a specific purpose, analyze its result and based on this analysis select some other operation. In our system the operations and their results are displayed visually. In our system the preparation to satisfy a sophisticated information need among the initially unfamiliar XML data sources typically proceeds through the following main phases: (1) use keyword search to find the most appropriate XML data sources from the underlying dataspace; (2) assess the usefulness of XML data sources produced in Phase 1 and select the relevant ones; (3) find out possible data conflicts among relevant XML data sources; (4) customize data of relevant XML data sources, if necessary; and (5) express the sophisticated information need. In next sections these phases are described in detail.

4.1. The visual front view

To be able to satisfy his/her information needs, the user must first find relevant XML sources from a dataspace. In this section, we introduce the visual primitives that assist the user in finding them. The visual front view (see Figure 4.1) in our dataspace system consists of a main window divided into three parts. The left side of the window has been reserved for the messages of the system. Every time the user triggers an operation, the system notifies with a message whether the operation was successful or not. The right side of the window is used to visualize the XML sources of the underlying dataspace. Three tabs, labeled “Search”, “Conflicts” and “Dataspaces,” are in the bottom of the main window. (The functionality behind the “Search” and “Conflicts” tabs will be explained in Section 5.) In the beginning of a user session, the tab “Dataspaces” is active. In this tab, the user selects the dataspace of interest from the right-hand-side drop-down list by clicking the "Open Dataspace" button, which results that all the XML sources belonging to the chosen dataspace are displayed in a new browsable window. The user can browse between the XML data sources by clicking the "<" and ">" buttons. The number of the available XML sources and the ordinal of the active XML source are shown between the buttons. If the user wants to view only some specific XML sources, (s)he can open them one by one by selecting their names from the left-hand-side drop-down list and clicking the button “Open document”.

The XML sources are displayed based on the visual metaphor presented in Section 3.. First only the root of an XML source is shown. By right-clicking the name of the root and selecting the menu item “Total Visualization”, the visualization of the whole XML structure is displayed. Another option is to expand or reduce the tree view by one level at time by clicking the plus or minus sign in the end of the data item name. If the user is interested in the values related to the occurrences of the chosen data item, (s)he right-clicks the data item name, selects the menu item “Values”, and the values are displayed in a pop-up window. From this visualization the user is able to by one glance see which data items are contentual and which are structural. Contentual data items (data items with values only) are displayed with grey filling, whereas structural data items (data items whose immediate successors are attributes and/or elements only) are displayed with white filling. The data items which are both contentual and structural are displayed with light grey filling.



Ranking order:

1. short_publications_6
2. short_publications_5
3. short_publications_2
4. short_publications_1
5. short_publications_3
6. short_publications_4
7. short_library

Dataspace **'short_test - search'** opened.

world_wide_book_w
Instances: 1
Frequency: 100%

slogan
Instances: 1
Frequency: 100%

store_info
Instances: 1
Frequency: 100%

pub_info(-)
Instances: 1
Frequency: 100%

work(-)
Instances: 2
Frequency: 100%

type
Instances: 2
Frequency: 100%

author
Instances: 2
Frequency: 100%

published
Instances: 2
Frequency: 100%

edition
Instances: 1
Frequency: 50%

price
Instances: 2
Frequency: 100%

Dataspace: short_test

Documents:
short_doc4
short_univ_x
short_univ_y
short_univ_z
short_univ_w
short_publications_1
short_publications_2
short_publications_3
short_publications_4
short_publications_5
short_publications_6

Subdataspace: relevant_sources

Documents:
short_publications_1
short_publications_2
short_publications_3
short_publications_4
short_publications_5
short_publications_6

short_publications_2 3 / 7

Open document: short_doc1
Open dataspace: short_test
All: where begin

Figure 4.1. The picture of the visual user interface: menus, tabs, search results, visualization of a data source and creation of a subdataspace

The user is able to find out the potentially relevant data sources through keyword search. The visual primitives for keyword search are also found in the “Dataspaces” tab. The “Search” button is associated with three text field, “All”, “Values” and “Data Item Names”. The keywords typed in the first text field are search both among data item names and among values, the keywords in the second field among values, and the keywords in the third field among data item names. Our keyword search is case insensitive, and the character “%” can be used for truncation both in the beginning and in the end of the keyword. Our keyword search has been implemented based on language modeling and Jelinek-Mercer smoothing. The search results are given in two different ways. The first one is the ordered list of XML data source names displayed in the left side of the main window. The second one is the browsable window (similar to the one explained above) in the right side of the main window. It contains a virtual collection of the potentially relevant XML sources. The XML sources are in the same ranked order as the list in the left side.

In the top of the main window there are four menus: “File”, “Dataspaces”, “Settings” and “Query”. The “File” menu contains three menu items, “Add Data Source”, “Open Data Source” and “Save Data Source” (their functionality is self-exploratory). Also the “Dataspaces” menu has three menu items, “Create Subdataspace”, “New Dataspace” and “Remove Dataspace”. “Create Subdataspace” is used to make a copy of selected data sources in order to avoid changing the original sources. The selection of this option generates a pop-up window. In this window, the user can create a new subdataspace from the data sources of the existing dataspaces. In the left side of the window, the user selects the data sources by choosing first the dataspace from the “Dataspace” drop-down list, then selecting the name of the XML source from the list beneath and finally clicking the “Add document” button. The name of the new subdataspace is typed in the text field in the right side of the window. The subdataspace is created by clicking the “Create” button in the bottom of the window. The created subdataspace can be opened from the “Dataspaces” tab of the main window. The menu item “New Dataspace” is used to create a new dataspace from XML sources which do not belong to any existing dataspace. This is similar to the creation of a subdataspace. The removal of a dataspace is done through the “Remove Dataspace”. Its selection generates a pop-up window with a drop-down list containing the names of the existing dataspaces. The selected dataspace is removed by clicking the “Remove” button. The “Settings” menu contains various options for adjusting the system.

4.2. Finding relevant sample XML sources

In our example information need, the user wants to compare the average price of the book “How to begin” written by Charlotte Lewis in different countries. Therefore, (s)he is interested in XML sources that include the price of this book. In this example case, the dataspace consists of numerous XML sources. The user understands that only part of them might be interesting to her. By applying the keyword search with keywords “Charlotte”, “Lewis, “how”, “begin” into a dataspace we assume, that the seven XML sources in Appendix A are returned for the user. Only one of them (see Figure 3.1(g)) is not relevant from the viewpoint of the user’s information needs – the XML source does not contain enough information of this book. (see Section 5.2).

Let us assume that the user wants to compare the average prices of Charlotte Lewis's book “How to begin” in different countries. Now, the first task is to find all such XML sources that contain information about this book. The user begins with by selecting the dataspace where (s)he thinks the promising XML sources can be found. Next, the user does the keyword search into the selected dataspace with keywords *Charlotte*, *Lewis*, *how*, *begin* (the "All" text field above). The top-ranked results of this search consist of all our seven sample XML data sources.

Even if a specific XML data source belongs to the result of the above keyword search, it is not guaranteed that it would be appropriate to the user. Therefore, the user has to take a closer look at each of the data sources to assess their relevance. By examining the total visualization of each XML data source, which displays the whole structure of the XML source, the user can observe, for example, that in *short_publications_2.xml* the data item **work** has both value and structural contents. When the user reviews the values of this data item, (s)he notices that they express article and book titles. On the other hand, *short_publications_3.xml* contains a data item with the name **rrp** whose semantics may not be clear to the user. By examining its values, the user is able to conclude that they represent currency and that **rrp** stands for “recommended retail price”. When the user explores also the remaining result XML sources in detail, it becomes obvious that the top-six result XML returned by the keyword search are relevant, whereas the last data source is not relevant because it contains no information about the aforementioned book or its price.

While exploring the relevant data sources, the user may notice that they contain similar information, which is however represented in different ways. This means that these XML sources need to be harmonized and customized in order to use them to satisfy the user's information needs. Since the information needs that are satisfied through dataspace systems are typically ad hoc and short-term, it may not be desirable to customize the original XML data sources. Therefore in our case, the user creates a subdataspace "relevant_sources". The next section introduces visual primitives in terms of which the user is able to ensure that the XML data sources in the subdataspace "relevant_sources" are consistent. This is done by detecting and resolving the possible data conflicts in them.

5. Detection and resolution of XML data conflicts

The relevant XML data sources may contain several forms of heterogeneity, which manifest themselves as data conflicts. The user needs to take them into account before or while querying these XML sources. Our system includes tools for detecting the following essential data conflict types in XML sources, which were originally introduced in [Niemi et al., 2009]:

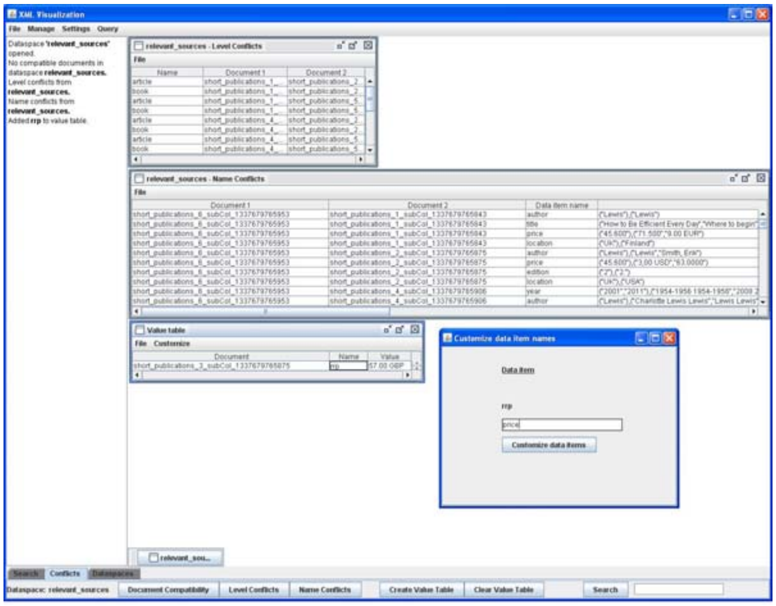
1. Document conflict (document-to-document conflict) between two XML sources occurs if the same information content has been structured differently.
2. Name conflict (data item-to-data item conflict) means that the data items with the same intended meaning are named differently or data items with different meaning are named identically among XML sources.
3. In level conflict (data item-to-value conflict) the same piece of information is represented in some XML source as a data item name and in some other XML source as a value.
4. Value conflict (value-to-value conflict) occurs when semantically equivalent values have not been presented uniformly.

The expressive power of these tools is specified formally in [Näppilä and Niemi, 2012]. In this section, we first introduce the visual primitives meant for assisting the user in detecting of the potential data conflicts. Most of the conflicts can be resolved by customizing XML sources. Next, we introduce how the XML sources are customized. Finally, we use these tools in the context of our relevant sample data sources to demonstrate how the data conflicts are detected and resolved.

5.1. Visual primitives for finding data conflicts and customizing XML sources

The tab “Conflicts” in our visual interface contains the tools for assisting in the detection of data conflicts. The tab contains the following buttons “Document Compatibility”, “Level Conflict”, “Name Conflict”, “Create Value Table”, “Clear Value Table”, and “Search”, which is associated with the text field. The clicking of the first four buttons produces a table (called result table here on) which indicate the potential data conflicts. The last button is used for searching for a given string in the result table. As a result (s)he gets the table of those rows which contain the searched string. Every result table contains a “File” menu whose menu item “Save” can be used to save these tables.

The conflict tools are typically used in the order which they appear in the “Conflicts” tab. The usage of the first three tools is analogous. If the conflict type selected by the user does not occur in the given dataspace, then the user is notified. The table produced by applying the document compatibility tool consists of two columns, "Document1" and "Document2", which contain the names of the XML sources. A row in this table denotes that the two XML sources are compatible. If two XML sources do not appear in the result table, this means that there is a document conflict between them.



Dataspace **'relevant_sources'** opened.
No compatible documents in dataspace **relevant_sources.**
Level conflicts from **relevant_sources.**
Name conflicts from **relevant_sources.**
Added **rrp** to value table.

Name	Document 1	Document 2
article	short_publications_1_...	short_publications_2...
book	short_publications_1_...	short_publications_2...
article	short_publications_1_...	short_publications_5...
book	short_publications_1_...	short_publications_5...
article	short_publications_4_...	short_publications_2...
book	short_publications_4_...	short_publications_2...
article	short_publications_4_...	short_publications_5...
book	short_publications_4_...	short_publications_5...

Document 1	Document 2	Data item name	
short_publications_6_subCol_1337679765953	short_publications_1_subCol_1337679765843	author	("Lewis"),("Lewis")
short_publications_6_subCol_1337679765953	short_publications_1_subCol_1337679765843	title	("How to Be Efficient Every Day","Where to begin"
short_publications_6_subCol_1337679765953	short_publications_1_subCol_1337679765843	price	("45.600"),("71.500","9.00 EUR")
short_publications_6_subCol_1337679765953	short_publications_1_subCol_1337679765843	location	("UK"),("Finland")
short_publications_6_subCol_1337679765953	short_publications_2_subCol_1337679765875	author	("Lewis"),("Lewis","Smith, Erik")
short_publications_6_subCol_1337679765953	short_publications_2_subCol_1337679765875	price	("45.600"),("3,00 USD","63.0000")
short_publications_6_subCol_1337679765953	short_publications_2_subCol_1337679765875	edition	("2"),("2.")
short_publications_6_subCol_1337679765953	short_publications_2_subCol_1337679765875	location	("UK"),("USA")
short_publications_6_subCol_1337679765953	short_publications_4_subCol_1337679765906	year	("2001","2011"),("1954-1958 1954-1958","2008 2
short_publications_6_subCol_1337679765953	short_publications_4_subCol_1337679765906	author	("Lewis"),("Charlotte Lewis Lewis","Lewis Lewis"

Document	Name	Value
short_publications_3_subCol_1337679765875	rrp	57.00 GBP

price

Figure 5.1. Conflict tab, data conflict tables and data item name customization

The table produced by “Level Conflicts” contains three columns, “Name,” “Document1” and “Document2.” Its row denotes that in the first XML source (“Document1”) the string in “Name” is represented as a data item name, whereas it is a value in the second XML source (“Document2”). There are four columns (“Document1”, “Document2”, “Data Item Name”, “Values”) in the result table produced by “Name Conflicts”. Its row denotes that both the XML sources “Document1” and “Document2” contain the data item name given in “Data item name”. The fourth column, “Values,” contains all the values related to this data item given as separate sets so that the values related to the first XML source are represented first within parentheses followed by the values of the second source in their own parentheses. By examining the result table organized in this way, the user is able to conclude that two XML sources contain data items with the same name but with different meaning. Usually this causes the need to customize the data item names for separating their semantics. By analyzing the values, the user may also notice that there are values which have the same intended meaning but they are represented differently. This causes a value conflict. The value conflict may be due to different spelling or measure. For example, the name of a person might have been spelled differently or different currency is used. These values have to be customized in the case they are needed in satisfying the user's information needs.

The customization of the data item names and values starts in a similar fashion. First, the user opens the total visualization of the XML source, where (s)he right-clicks the data item names (or data item names values of which) needed to be customized. In the case that the occurrences related to one data item name or its values need customization, the user selects the menu item “Customize” from the menu, and the “Value Table” window is generated. If more than one data item name (or their values) needs customization, the user selects the menu item “To Value Table”. In the case of more customizations are needed, the user repeats the selection until all the required data items have been dealt. Then (s)he clicks the button “Create Value Table” in the “Conflicts” tab, and the “Value Table” window is generated. The “Value Table” window contains two menus, “File” and “Customize”, and a table that has three columns: "Document" (denoting the names of the XML sources), "Name" (data item names) and "Values". The menu item “Save” in the “File” menu gives a possibility to save the data of the table.

If the user wants to populate the “Value Table” only with the relevant data item occurrences, (s)he has to search for subtrees whose the root this data item is. To do this, the user right-clicks the data item name and selects the menu item ”Search” from the generated menu. The selected data item is shown with a frame labeled “Search”, and the ”Search” tab in the bottom of the main window is activated. The user types the keywords into the text fields of the “Search” tab and triggers the search by clicking the “Search” button. The search results open up in a new browsable window. The user is able to insert a data item occurrence into the value table in the same way as explained above.

While customizing the data item names, the user selects from “Value Table” those data item names that (s)he wants to be represented identically. After that, the user selects the “Data Items” menu item from the “Customize” menu. This opens up new a pop-up window, “Customize Data Item Names”, where the selected data item names are listed. Under them is a text field in which the user can type a new data item name which replaces the selected names. Customization is triggered by clicking the “Customize data items” button below the text field. It results that all the listed data item occurrences are renamed in the “Value table” and in the selected XML sources. If the user wants to customize values representing currency, (s)he selects the appropriate values and the menu item “Currency” from the “Customize” menu. This opens up the “Customize Currencies” pop-up window where the selected (currency) values are in the left and the text fields in the right side of this window. In the bottom there are two buttons, “Customize Values” and “Customize by Example”. The user types the desired exchange rate for each value into the text field next to it. If the exchange rate is the same for all the values, the user may type the exchange rate only into the last text field. The actual customization is triggered by clicking the “Customize Value” button. With the functionality behind the “Customize by Example” button, the user-selected values act as an example for converting currencies through the entire dataspace. We have currently implemented only currency conversion, but also other kinds of customizations could be easily augmented to our system. If the user needs to customize string values, (s)he selects first the strings that she wants to be represented identically. After that (s)he selects the menu item “Values” from the menu “Customization” and this generates the “Customize values” window that is similar in its appearance and usage with the “Customize data item names” window.

5.2. Detection of the data conflicts among relevant sample XML sources and their resolution

Although the user is aware of the contents, structures and semantics related to the XML sources in the subdataspace “relevant_sources,” their manipulation presupposes that the user is able to detect and resolve the possible data conflicts among them. For this, (s)he needs the visual primitives introduced in the previous section.

Based on the visualizations of the XML sources the user is able to observe that they are all structured differently. By employing the "Document Compatibility" tool (s)he becomes assured that no two XML sources are compatible. By clicking the “Level conflict” button the user gets candidates for this conflict type in the “relevant_sources” subdataspace. From the result, the user observes, for example, that that the author name *Lewis* is a data item name in *short_publications_3.xml*, whereas it is a value in *short_publications_6.xml*. Further, when the user reviews their visualizations (s)he is able to conclude that in these two XML sources *Lewis* refers to the name “Charlotte Lewis” in the context of the book “How to begin”. From the total visualization of *short_publications_3.xml*, the user notices that the data item occurrence title is a successor of the data item occurrence *Lewis* and its value is “How to begin”. Likewise, in short_publications_6.xml the user espies two *last_name* data items. Be reviewing the values related to them, (s)he notices that the value of the leftmost *last_name* data item is *Lewis* and at the same level there is a title data item whose value is “How to begin”. Pivoting data between the schema and instance levels is made through the visual query interface to be introduced in the next section.

In the context of “relevant_sources”, the result of the “Name Conflicts” tool (see Figure 6.1) indicates that there are 23 shared names among the available sources. The user notices that data item name *price* appears in several XML sources. In order to be sure that the occurrences of this data item have the same semantics, the user has to explore the values of this data item in different XML sources. The user searches for the string "price" in the result table, and gets a reduced table as a result. By reviewing this reduced table, the user is able to conclude that all the price data items in *short_publications_1.xml*, *short_publications_2.xml*, *short_publications_4.xml*, *short_publications_5.xml* and *short_publications_6.xml* have the same meaning. However, based on his/her previous analysis, the user knows that in *short_publications_3.xml* the *price* is expressed as the data item *rrp*. The user resolves this name conflict by renaming the data item name *rrp* to *price*.

The examination of the “Name Conflicts” table reveals that there are also value conflicts among the values of price data items. Namely, different currency labels (USD, EUR, GBP or no label at all) are used in them and the numeric values are represented in two different formats (comma vs. period). In order to be able calculate the averages, the user needs to convert these values into same unit (let us assume that the user selects USD). The customization of values where the currency unit was expressed explicitly was explained above. However, in the case the currency is not expressed explicitly the user needs to search hints for it in the XML source at hand. For example, in *short_publications_5.xml* the name of its root data item (*USA_online_works*) suggests that the appropriate unit is USD.

The user notices from the original “Name conflicts” result table that data item name apparently has different semantics in *short_publications_1.xml* and *short_publications_4.xml*. Namely, in *short_publications_1.xml name* refers to the title of a literary work (book or article), whereas in *short_publications_4.xml* it has three different meanings: the title of a literary work, the name of an author and the name of a publisher. The user needs to customize all the available XML sources so that the data item names which refer to the title of the book of interest are *title*.[1] Also the relevant author names should be represented identically in all the XML sources. The author names in “relevant_sources” are expressed variously (*author*, *writer*, *name*, and *lastname–firstname*). Let us assume that the user decides that they all are represented as author. By examining the spelling variants of the author names the user concludes that these values have also to be customized and replaces all the variants of “Charlotte Lewis” with “Lewis”.

In the “Name Conflicts” result table, there is also a name conflict between *short_publications_1.xml* and *short_publications_6.xml* concerning the data item country (values “Finland” and “UK,” respectively). Now, the user is interested in how the location of stores is

[1] It is worth noting that in this case it is sufficient to customize only those parts of the available data sources which contain relevant information. As a consequence, the customization is not materialized since it would result in that the original XML data sources become partially modified. For this reason, the dataspace approach is different from the traditional data integration.

expressed in the other XML sources as well and finds out that in *short_publications_2.xml* it is within the *store_info* data item ("123 North Pole Avenue, New York"), in *short_publications_3.xml* within contact ("London, +1234567899") and in *short_publications_4.xml* within location (value "Finland"). On the other hand, in *short_publications_5.xml* the name of the root data item (*USA_online_works*) suggests that the location is the USA. It becomes obvious that to be able to calculate the average prices in different countries, the user needs to customize both the data item names (*location*) and the values concerning the locations of the stores (*Finland*, *UK*, *USA*).

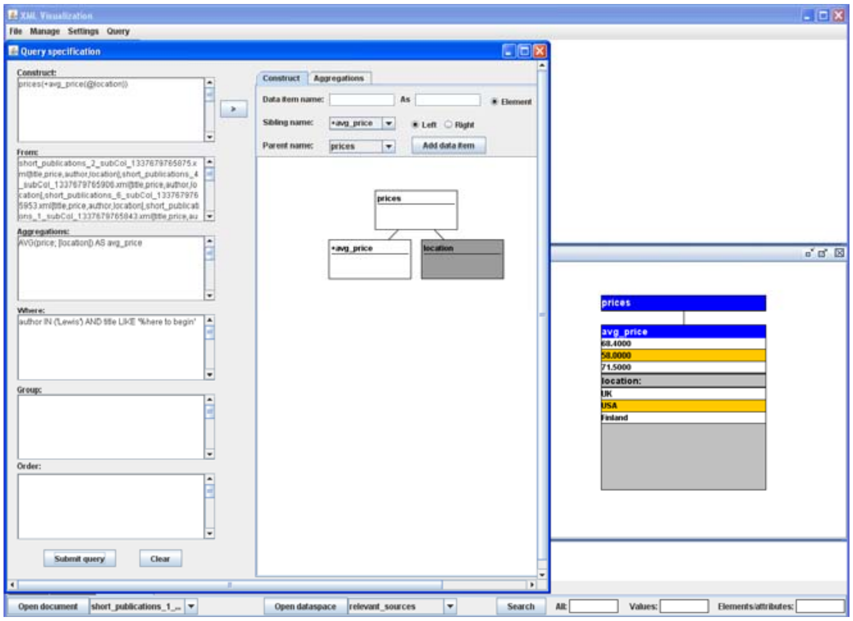
Through the application of the above tools, the user becomes aware of all the data conflicts that occur among the relevant XML data sources. The majority of these data conflicts have been resolved through customization. In the next section, we describe how the user's sophisticated information needs is finally satisfied by the query interface in our dataspace system. In this context, we also show how the level conflicts can be resolved.

6. Satisfying the user's sophisticated information needs

In our dataspace system, the user satisfies his/her information needs through the visual query interface based on the RXQL query language. RXQL, introduced in [Näppilä et al., 2011], is specifically designed for extracting, selecting, restructuring, harmonizing and aggregating heterogeneous XML data. RXQL is a declarative query language, which means that the user does not need to express how the result of the query is produced but need only to specify the data item names and their relationships in the result. In this section, we first present the visual primitives in terms of which the user expresses his/her information need and then the information needs related to our running example are specified and evaluated.

6.1. The primitives of the visual query interface

The query formulation is started by opening the "Query Specification" window (see Figure 7.1) through the "Create Query" menu item in the "Query" menu. The window is divided into two parts. On the left side there are six text boxes ("CONSTRUCT", "FROM", "AGGREGATIONS", "WHERE", "GROUP" and "ORDER") and on the right side two tabs ("Construct" and "Aggregations"). The text boxes are reserved for the specific query parts used in query specification. Only the first two parts are mandatory. The typical filling order of the boxes is top-down.



Construct: prices(+avg_price(@location))

From: short_publications_2_subCol_1337679765875.xml@title,price,author,location|,short_publications_4_subCol_1337679765906.xml@title,price,author,location|,short_publications_6_subCol_1337679765953.xml@title,price,author,location|,short_publications_1_subCol_1337679765843.xml@title,price,au

Aggregations: AVG(price; [location]) AS avg_price

Where: author IN ('Lewis') AND title LIKE '%here to begin'

Group:

Order:

Data item name: As: ● Element

Sibling name: +avg_price ● Left ○ Right

Parent name: prices

prices
+avg_price
location

prices

avg_price: 68.4000, 58.0000, 71.5000

location: UK, USA, Finland

Open document: short_publications_1_...

Open dataspace: relevant_sources

All: Values: Elements/attributes:

Figure 6.1. The RXQL query window (on the left) and the result document (on the right)

In the “Construct” tab, the user is able to specify the content and structure of the desired result document visually. The process begins by typing the name of the data item into the “Data Item Name” text field. If the data item name is used in the “FROM” part and the user wants to use a different name for it in the result, (s)he types the new name into the “AS” text field. In case a data item is repetitive, the user just types the ‘+’ sign before its name. The type of the data item is indicated through radio buttons. Structurally, the location of the data item in the result document is specified from the “Sibling Name” or “Parent Name” drop-down lists, which include all the data item names that the user has already given as result data items this far. From the “Parent Name” list, the user can select the name of the parent data item. If the user specifies the location of the new data item with respect to a sibling, (s)he first needs to right-click a data item name of the result visualization and selects from the “Sibling name” list the name of its nearest sibling. The user is also able to express whether the result data item is placed to the left or right side of the selected sibling. Data item can be removed from this visualization by clicking the scroll wheel of the mouse. The user may also specify the content and structure of the desired result document by typing the corresponding RXQL expression directly into the "CONSTRUCT" text box.

In the “FROM” text box, the user specifies the XML sources and their data items which are used in the query. The expressions in the “FROM” text box are constructed through the visualizations of each XML data source. The user opens the visualization of an XML source, right-clicks the needed data item name and selects from the generated menu the submenu “From” with six menu items “Basic”, “Into”, “Into (this)”, “Value() as”, “As” and “Create”. Through the menu item “Basic”, the user expresses that this data item name can be used in the result. The menu items “Into”, “Into (this)” and “Value() as” are used for resolving level conflicts. The “Into” menu item generates a pop-up window with two text fields. The data item name given in the leftmost field is treated as the value of the data item name given in the rightmost field. The difference between “Into” and “Into (this)” is that with “Into” the names of all such data items, which are children of the selected data item, are set as default text into the leftmost text field, whereas with “Into (this)” only the name of the selected data item is set as default text. With “Value() as”, the values of the data items, which were manipulated with “Into” and “Into (this)”, are moved as the values of the data item whose name is given in the rightmost text field. The menu item “As” is for renaming of data items. By clicking this menu item, a pop-up window with two text fields opens up. In the left-hand field the data item name to be replaced is shown, and the user types the new name in the right-hand field. When the user has specified all the needed data item names, then (s)he selects the “Create” menu item and the corresponding “FROM” part is echoed in the visual query interface.

If the user wants to use the primitives of the “Aggregations” tab for filling the “AGGREGATIONS” text box, the “FROM” text box needs to be filled first with the visual primitives. This is due to the fact that the data item names used in “AGGREGATIONS” box must be found in the XML sources specified in the “FROM” box. On the left side of the “Aggregations” tab are drop-down lists “Select Function” and “Select Data Item” and the text field “Name”. From the “Select Function” list the user selects the function to be used for aggregating the values selected from the “Select Data Item” list. The name of the result of the aggregation is typed into the “Name” text field. The grouping factors are selected from the list by typing the ordinal of the grouping factor (data item name) into the text field before its name. The last three query parts “WHERE”, “GROUP” and “ORDER” are analogous to the corresponding constructs in SQL, and their contents are typed into the respective text boxes.

When the RXQL query is formulated through the visual interface, the user executes the query by clicking the button “Submit query”. The user has three options to view the result. The first one is the default tree view of the result document. In this option, the names of the data items with substructures are displayed by blue background color and white text, whereas data items with only values are displayed by grey background and black text. The values are displayed with black text on white background. In this visualization, the user may locate the associated values by pointing the value with mouse. This changes the background color of the values that belong together from white to yellow. The second option to view the result is the tree visualization described in Section 5. This visualization is available through the menu item “Normal Visualization” from the “Query” menu. The third option is the textual XML document that the user first needs to save with the “Save Document” menu item (available in the “File” menu) and then to open with a program capable of showing XML. This option is useful when query results are required as textual XML data.

6.2. Sophisticated query to satisfy the sample information need

As stated in Section 4.2, the user's information needs is to find out the average prices of Charlotte Lewis's book "How to begin" in different economic areas. As explained above, the user has created the subdataspace "relevant_sources" and through customization resolved all the data conflicts, except for the level conflict, from its XML sources. Therefore, the query specification with our visual query interface is more straightforward than in the textual RXQL where the user is also responsible for resolving the data conflicts.

The user specifies the content and structure of the result document with the visual primitives introduced above. In our sample case, the content and structure of the result document is simple. Its root element is *prices* and has one child element *avg_price* with the attribute *location*. The user formulates the "FROM" part as follows. First, (s)he opens all the relevant XML sources and selects the data items of interest (*location*, *author*, *title*, *price*) in them through the menu item "Basic" of the "From" submenu. The user needs to pay a special attention only to the data item *Lewis* in *short_publications_3.xml* and to *USA* in *short_publications_5.xml,* because they contain level conflicts.

The data item names *Lewis* and *USA* need to be transposed to values of *author* and *location*, respectively through the "Into" menu item. Next, the user formulates the "AGGREGATIONS" through the "Aggregations" tab. (S)he selects the function "AVG" and the data item *price* and specifies that the grouping factor is the data item *location*. As the XML sources contain also information about other publications than the ones the user is interested in, the user needs to restrict the result. This is done through the "WHERE" text box where the user specifies that (s)he is only interested in such publications where the author name is "Lewis" (the expression "author IN ('Lewis')) and the title is "How to begin" (LIKE ('How to begin')). Now, the query is complete and can be executed. The result of this query is depicted in Figure 6.1. From this result, the user is able to observe that the lowest average price are found in the USA (58.00) and the highest in Finland (71.50).

7. Discussion

There are several textual and visual query languages with which it is possible to satisfy sophisticated information needs. However their usage presupposes that the user knows which data sources (s)he needs in satisfying his/her information need. In addition the user has to know the information contents, structures and semantics of the data sources of interest. Also the data sources are supposed to be well-integrated. In this kind of environment the user has to specify the navigation to connect the needed data.

Nowadays, there are more and more such sophisticated information needs which do not satisfy the above background assumptions. In other words, in satisfying his/her information needs the user does not have knowledge about:

1. Which data sources (s)he should manipulate
2. What kind of information content, semantics and structure is related to a specific data source i.e. the user is initially unfamiliar with the data sources
3. What kinds of data conflicts exist in relevant data sources and how these factors can be taken into account for or removed.

It is obvious that the expressive power of conventional textual or visual query languages is not sufficient to handle this kind of situation. In other words a new approach with versatile facilities is needed. The recent data management trend called dataspacing has been proposed for this purpose.

The dataspace systems developed so far are tailored to specific application domains. In our work, we concentrated on developing a general-purposed dataspace system. Our dataspace system is based on XML and it is general enough to handle any XML based application domain. XML was chosen as the data format of our system because of it is in itself a popular data format and in addition the other data formats can be converted easily into it.

Typically the users of dataspaces are often initially unfamiliar with its data sources. The main purpose of dataspace systems is to offer such functionalities in terms of which the user is able to satisfy his/her information needs although (s)he does not know the underlying data sources. The dataspace system should provide the user with the possibility to gradually increase his/her knowledge on data sources in order to make it possible to specify sophisticated queries.. The functionalities of our dataspace system may be divided into four groups:

1. The functionalities that assist the user to find relevant XML sources from available XML sources.
2. The functionalities that help the user to understand the contents, structures and semantics of the relevant XML sources.
3. The functionalities that assist the user to detect and handle (e.g. by customizing) the heterogeneity factors among the autonomous and heterogeneous XML sources.
4. The query facilities in terms of which the user is able to to extract, select, rename, restructure, order, group and aggregate XML sources for satisfying sophisticated information needs.

In this paper, we propose a visual interface offering all the functionalities listed above. To the best of our knowledge, this is the first visual XML dataspace system with these functionalities. The main starting point in developing our system was to develop a dataspace system the use of which does not require programming skills or knowledge of XML syntax.

Our system is founded on the notion of XML relation [Niemi and Järvelin, 2006; Niemi et al., 2009]. The basic functionalities of handling the XML relation have been implemented in programming language C and added as functions into PostGreSQL data base. The XML sources are also stored in it. The visual interface has been implemented with Java and NetBeans. In implementing the fourth group of functionalities, we also utilized the processing mechanism of our previous work, RXQL [Näppilä et al., 2011].

We have plans to expand our visual XML dataspace system and continue our research. The XML dataspace may include such semantic relationships among the data of XML sources that the user is not aware of them although (s)he would be interested in them. These relationships are usually called ‘Semantic Associations’. There are two kinds of semantic associations called basic and derived associations. The basic semantic association expresses explicitly that two entities are directly associated with each other for example person A is a friend of person B. The derived association means that two entities are connected with each other through several basic associations. In addition to the example above person C is a friend of person B and the connection between A and C is unknown. Still in this case person A and person C are connected to each other with derived association: person A can be connected to person C (and vice versa) through person B (A → B and B → C, so A → C). The derived semantic associations can be constructed automatically provided the basic associations are known. In the future, our XML dataspace system will assist the user in finding and establishing basic semantic associations among XML sources. This makes it possible to automatically detect the derived semantic associations e.g. with our query languages [Niemi and Jämsen, 2007a; Niemi and Jämsen, 2007b].

8. Conclusions

In this paper we propose a novel visual and general-purpose XML dataspace system. Its use does not require that the user knows XML syntax or possesses programming skills. Our visual XML dataspace system contains visual primitives that assist the user to find the relevant XML sources from the underlying dataspace, to learn about the contents, semantics and structures of such unfamiliar XML sources, to detect and remove data conflicts among the sources and to formulate sophisticated queries. In our approach the tree visualization of XML data plays a central role when the user gradually increases his/her knowledge about the underlying XML sources. A visual metaphor for this visualization is defined in this paper. In this paper we demonstrated how the user can satisfy his/her a sophisticated information needs among highly heterogeneous XML sources through our system.

Acknowledgements

This work was supported by the Academy of Finland under the project no. 140315. The authors wish to thank Professor Kalervo Järvelin for his useful comments and Mr. Mikko Kokkonen for his contribution in the implementation of some parts of the functionality behind our system.

References

[Abiteboul et al., 2002] Serge Abiteboul, Sophie Cluet, Tova Milo. Correspondence and translation for heterogeneous data. *Theoretical Computer Science* **275** (1-2), 2002: 179-213.

[Blunschi et al., 2007] Lukas Blunschi, Jens-Peter Dittrich, Olivier René Girard, Shant Kirakos Karakashian, and Marcos Antonio Vaz Salles. A dataspace odyssey: The iMeMex personal dataspace management system (demo). In: *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research*, 2007:114-119.

[Brabrand et al., 2008] Claus Brabrand, Anders Møller, and Michael I. Schwarzbach: Dual syntax for XML languages. Information Systems, **33** (4-5), 2008 : 385-406.

[Cai et al., 2005] Yuhan Cai, Xin Luna Dong, Alon Halevy, Jing Michelle Liu, and Jayant Madhavan. Personal information management with SEMEX. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, 2005: 921-923.

[Ceri et al., 1999] Stefano Ceri, Sara Comai, Piero Fraternali, Stefano Paraboschi, Letizia Tanca, and Ernesto Damiani: XML-GL: A graphical language for querying and restructuring XML documents. In: *Atti del Settimo Convegno Nazionale Sistemi Evoluti per Basi di Dati*, 1999: 151-165:

[Chawathe et al., 1994] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom: The TSIMMIS project: Integration of heterogeneous information sources. In: *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, 1994: 7-18.

[Cohen et al., 2003] Sara Cohen, Jonathan Mamou, Yaron Kanza and Yehoshua Sagiv: XSEarch: A Semantic Search Engine for XML. VLDB, 2003: 45-56.

[Das Sarma et al., 2009] Anish Das Sarma, Xin Luna Dong, and Alon Y. Halevy: Data modeling in dataspace support platforms. In: *Conceptual Modeling: Foundations and Applications – Essays in Honor of John Mylopoulos*. LNCS **5600**, Springer, 2009: 122-138.

[Dittrich, 2009] Jens Dittrich: The iMeMex dataspace management system: Architecture, concepts, and lessons learned. In: *Proceedings of the 26th British National Conference on Databases*. LNCS **5588**, Springer, 2009: 7.

[Dittrich et al., 2007] Jens-Peter Dittrich, Lukas Blunschi, Markus Färber, Olivier René Girard, Shant Kirakos Karakashian, and Marcos Antonio Vaz Salles: From personal desktops to personal dataspaces: A report on building the iMeMex personal dataspace management system. In: *Datenbanksysteme in Business, Technologie und Web*, 2007: 292-308.

[Dittrich and Vaz Salles, 2006] Jens-Peter Dittrich and Marcos Antonio Vaz Salles. iDM: A unified and versatile data model for personal dataspace management. In: *Proceedings of the 32nd International Conference on Very Large Data Bases*. ACM Press, 2006: 367-378.

[Elsayed and Brezany, 2010] Ibrahim Elsayed and Peter Brezany: Towards large-scale scientific dataspaces for e-science applications. Database systems for advanced applications. LNCS **6193**, Springer, 2010 : 69-80.

[Erwig, 2003] Martin Erwig: Xing: a visual XML query language. *Journal of Visual Languages and Computing* **14** (1), 2003: 5-45.

[Fagin et al., 2005] Ronald Fagina, Phokion G. Kolaitisb, Renée J. Millerc, and Lucian Popaa: *Data exchange: semantics and query answering*. Theoretical Computer Science 336 (1): 89-124.

[Flesca et al., 2002] Sergio Flesca, Filippo Furfaro, and Sergio Greco: XGL: a graphical query language for XML. In: *Proceedings of the International Database Engineering & Applications Symposium*. IEEE Computer Society, 2002: 86-95.

[Florescu and Kossmann, 1999] Daniela Florescu and Donald Kossmann : Storing and querying XML data using an RDBMS, *IEEE Data Engineering Bulletin*, **22** (3) 1999 : 27-34.

[Franklin et al., 2005] Michael Franklin, Alon Halevy, and David Maier: From databases to dataspaces. *SIGMOD Record* **34** (4), 2005: 27-31.

[Haber et al., 1994] Haber E.M., Ioannidis Y.E., and Livny M. Foundations of visual metaphors for schema display. J. Intelligent Inf. Syst., 3(3/4):263–298, 1994.

[Halevy et al., 2006] Alon Halevy, Michael Franklin, and David Maier: Principles of dataspace systems. In: *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM Press, 2006: 1-9.

[Howe et al., 2008] Bill Howe, David Meier, Nicolas Rayner, and James Rucker: Quarrying dataspaces: Schemaless profiling of unfamiliar information sources. In: *Proceedings of the 24th International Conference on Data Engineering Workshops*. IEEE Computer Society, 2008: 270-277.

[Halverson et al., 2004] Alan Halverson, Vanja Josifovski, Guy Lohman, Hamid Pirahesh, and Mathias Mörchel: ROX: relational over XML. In: Nascimento, M.A., Özsu, M.T., Kossmann, D., Miller, R.J., Blakeley, J.A., Schiefer, K.B. (Eds), Proceedings of the 30th International Conference on Very Large Data Bases, Toronto, Canada, August 31-September 3, Morgan Kaufmann, San Francisco, CA, 2004: 264-75.

[Jagadish et al., 2002] H. V. Jagadish, Shurug Al-Khalifa, Adriane Chapman, Laks V. S. Lakshmanan, Andrew Nierman, Stelios Paparizos, Jignesh M. Patel, Divesh Srivastava, Nuwee Wiwatwattana, Yuqing Wu, Cong Yu: TIMBER: A native XML database. VLDB J. (VLDB) **11**(4), 2002: 274-291.

[Jeffery et al., 2008] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy: Pay-as-you-go user feedback for dataspace systems. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, 2008: 847-860.

[Jelinek and Slavik, 2004] Josef Jelinek and Pavel Slavik: XML Visualization Using Tree Rewriting. In: *Proceedings of the 20th spring conference on Computer graphics*, 2004 : 65-72.

[O'Keefe and Trotmann, 2004] Robert A. O'Keefe and Andrew Trotman: The simplest query language that could possibly work. In: *Proceedings of the 2nd Workshop of the INitiative for the Evaluation of XML Retrieval (INEX)*, 2004: 117-124.

[Lampathaki et al., 2009] Fenareti Lampathaki, Spiros Mouzakitis, George Gionis, Yannis Charalabidis, and Dimitris Askounis: Business to business interoperability: A current review of XML data integration standards. *Computer Standards and Interfaces* **31** (6), 2009: 1045-1055.

[Lenzerini, 2002] Maurizio Lenzerini: Data integration: A theoretical perspective. In: *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM Press, 2002: 233-246.

[Li and Meng, 2009] Yukun Li, Xiaofeng Meng: Supporting context-based query in personal DataSpace. In: *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. ACM Press, 2009: 1437-1440.

[Lo et al., 2005] Anthony Lo, Reda Alhajja and Ken Barker: VIREX: visual relational to XML conversion tool. *Journal of Visual Languages and Computing*. **17** (1), 2006 : 25–45.

[Lu et al., 2006] Eric Jui-Lin Lu, Bo-Chan Wu and Po-Yun Chuang. An empirical study of XML data management in business information systems. *Journal of Systems and Software*, **79** (7) 2006 :. 984-1000

[Min et al., 2008] Jun-Ki Min, Chun-Hee Lee and Chin-Wan Chung: XTRON: An XML data management system using relational databases. Information and software technology. 50 (5) 2008 : 462-479.

[Mirza et al., 2010] Hamid Turab Mirza, Ling Chen, Genchai Chen: Practicability of dataspace systems. *International Journal of Digital Content Technology and its Applications* **4** (3), 2010: 233-243.

[Niemi et al., 2009] Timo Niemi, Turkka Näppilä, Kalervo Järvelin: A relational data harmonization approach to XML. *Journal of Information Science*, **35** (5), 2009: 571-601.

[Niemi et al., 2011] Timo Niemi, Marko Junkkari and Kalervo Järvelin: Concept-based query language approach to enterprise information systems. To appear in *Enterprise Information Systems* (41 pages).

[Niemi and Järvelin, 2006] Timo Niemi and Kalervo Järvelin: Another look at XML. Department of Computer Sciences, University of Tampere, Tampere, Working Paper No. A-2006-1.

[Näppilä et al., 2011] Turkka Näppilä, Katja Moilanen and Timo Niemi: A query language for selecting, harmonizing, and aggregating heterogenous XML data. *International Journal of Web Information Systems,* **7** (1), 2011: 62-99.

[Näppilä and Niemi, 2012] Turkka Näppilä and Timo Niemi: An approach for developing a schemaless XML dataspace profiling system. *Journal of Information Science*, **38** (3), 2012: 234-257.

[Pal et al., 2004] Shankar Pal, Istvan Cseri, Oliver Seeliger, Gideon Schaller, Leo Giakoumakis, and Vasili Zolotov, Indexing XML data stored in a relational database. In: Nascimento, M.A., Özsu, M.T., Kossmann, D., Miller, R.J., Blakeley, J.A., Schiefer, K.B. (Eds), Proceedings of the 30th International Conference on Very Large Data Bases, Toronto, Canada, August 31-September 3, Morgan Kaufmann, San Francisco, CA, 2004 : 1134-45.

[Rys et al., 2005] Michael Rys, Donald D. Chamberlin, and Daniela Florescu: XML and relational database management systems: The inside story. *In: Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, 2005: 945-947

[Shanmugasundaram et al., 1999] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational databases for querying XML documents: limitations and opportunities, in Atkinson, M.P., Orlowska, M.E., Valduriez, P., Zdonik, S.B., Brodie, M.L. (Eds),Proceedings of 25th International Conference on Very Large Data Bases, Edinburgh, Scotland, September 7-10, Morgan Kaufman, San Francisco, CA, 1999 : 302-14.

[Tagarelli and Greco, 2008] Andrea Tagarelli and Sergio Greco: Semantic clustering of XML documents. *ACM Trans. Inf. Syst. (TOIS)* **28**(1), 2010 : 1-56.

[Vaz Salles et al., 2007] Marcos Antonio Vaz Salles, Jens-Peter Dittrich, Shant Kirakos Karakashian, Olivier René Girard, and Lukas Blunschi: iTrails: Pay-as-you-go information integration in dataspaces. In: *Proceedings of the 33rd International Conference on Very Large Data Bases*. ACM Press, 2007: 663-674.

[W3C RDF, 2004] *Resource Description Framework (RDF).* Available at http://www.w3.org/RDF/ (2004; accessed February, 2012).

[W3C SPARQL, 2008] *SPARQL Query Language for RDF*. Available at http://www.w3.org/TR/rdf-sparql-query/ (2008; accessed February 2012).

[W3C XML, 2008] *Extensible Markup Language (XML) 1.0 (Fifth Edition).* Available at http://www.w3.org/TR/REC-xml/ (2008; accessed February, 2012)

[W3C XML Schema, 2004] *XML Schema Part 0: Primer Second Edition.* Available at http://www.w3.org/TR/xmlschema-0/ (2004; accessed February, 2012).

[W3C XPath, 1999] *XML Path Language (XPath): Version 1.0*. Available at http://www.w3.org/TR/xpath/ (1999; accessed February, 2012).

[W3C XQuery, 2010] *XQuery 1.0: An XML Query Language (Second Edition)*. Available at http://www.w3.org/TR/xquery/ (2009; accessed February, 2012).

[Xing et al., 2008] Guangming Xing, Zhonghang Xia and Huanjing Wang : Xdiff+: a visualization system for XML documents and Schemata. In: *Proceedings of the 46th Annual Southeast Regional Conference*, Auburn, Alabama, March 28-29, 2008 : 40-45.

[Yklef and Alqahtani, 2010] Mourad Ykhlef and Sarra Alqahtani. Querying and restructuring XML data by graphical query language. *International Journal of Web Information Systems* **6** (3), 2010: 230-247.

Appendix A

Our relevant XML sources contain information about literary works: books and journal articles. They contain at least basic information of these works: the type of the work, author, title, price and year of the publication. These XML sources are assumed to be autonomous and heterogeneous. This is because data items containing the information of interest are labeled and structured differently in our XML sources. We shall introduce our visual tools of our XML dataspace system in the context of these XML sources. The XML relation representation of the sample XML data source *short_publications_4.xml* is given in Appendix B.

```
<online_db store="Euro books" postadd="Book Street 1, 00000 Helsinki" country="Finland">
   <book>
      <writer>Charlotte Lewis</writer>
      <name>How to begin</name>
      <price>55.00 EUR</price>
      <y_Of_p>2001</y_Of_p>
      <notes>The award-winning book!</notes>
   </book>
   <article>
      <writer>Erik Smith</writer>
      <name>My story about cats</name>
      <price>9.00 EUR</price>
      <y_Of_p>2010</y_Of_p>
      <drawings>Becky Hammer</drawings>
      <remark>paperback</remark>
   </article>
</online_db>
```

(a) short_publications_1.xml

```
<world_wide_book_web_store>
   <slogan>Order anytime and anywhere! We deliver!</slogan>
   <store_info>123 North pole Avenue, New York</store_info>
   <pub_info>
      <work type="article" author="Smith, Erik" published="2002" edition="2." price="3,00
USD">
      How to become a good vet
      </work>
      <work type="book" author="Lewis, Charlotte" published="2001" price="63,00 USD">
         How to begin
      </work>
   </pub_info>
</world_wide_book_web_store>
```

(b) short_publications_2.xml

```
<publications_db contact="London, +1234567899">
   <publications year="2001">
      <Lewis>
         <Book>
            <title>How to begin</title>
            <rrp>57.00 GBP</rrp>
         </Book>
      </Lewis>
   </publications>
   <publications year="2011">
      <Smith>
         <Book>
            <title>My story as a vet and novelist</title>
            <aka>Erik the vet</aka>
            <rrp>25.00 GBP</rrp>
         </Book>
      </Smith>
   </publications>
</publications_db>
```

(c) short_publications_3.xml

```
<USA_online_works>
   <work type="article" author="Smith, Erik" published="2001" price="3,00">
      <title>How to become a good vet</title>
   </work>
   <work type="book" author="Lewis, Charlotte" published="2001" price="53,00">
      <title>How to begin</title>
      <description>This famous book has got various prizes.</description>
   </work>
</USA_online_works>
```

(e) short_publications_5.xml

```
<for_sale_publications>
   <publisher pub="Work inc." year="2001" country="UK">
      <Books>
<author last_name="Lewis" first_name="Charlotte">
            <title>How to begin</title>
            <price>28.50</price>
         </author>
      </Books>
   </publisher>
   <publisher pub="Frog" year="2011" edition="2" country="Finland">
      <Articles>
         <author last_name="Lewis" first_name="Charlotte">
            <title>How to Be Efficient Every Day</title>
            <normalPrice>9.00</normalPrice>
            <special>New customers can have this article for free!</special>
         </author>
      </Articles>
   </publisher>
</for_sale_publications>
```

(f) short_publications_6.xml

```
<library_books>
   <author name="Lewis, Charlotte">
      <title>How to be efficient every day</title>
      <pubYear>2010</pubYear>
      <location>335</location>
   </author>
   <author name="King, George">
      <title>The spring comes</title>
      <pubYear>1982</pubYear>
      <location>335</location>
   </author>
   <author name="King, George">
      <title>One year after</title>
      <pubYear>1969</pubYear>
      <location>335</location>
   </author>
</library_books>
```

(g) short_library.xml

Figure A1. Sample XML sources

```
<works_www location="Finland">
  <publisher name="Lion books">
    <writer name="Erik Smith">
      <article name="My story about cats">
        <price>EUR 15.00</price>
        <pub_year>2010</pub_year>
        <illustration>
          <drawings>
            <artist name="Becky Hammer"></artist>
            <year>1954-1956</year>
            <original>
              <publisher name="Artist works">
                <pub_year>1954</pub_year>
              </publisher>
            </original>
            <original>
              <publisher name="Drawings inc">
                <pub_year>1955</pub_year>
              </publisher>
            </original>
            <original>
              <publisher name="Artists inc">
                <pub_year>1956</pub_year>
              </publisher>
            </original>
            <notes>pencil drawings</notes>
          </drawings>
          <colors>
            <artist name="Lisa Bell"></artist>
          </colors>
        </illustration>
      </article>
    </writer>
  </publisher>
```

(Continued)

```
  <publisher name="The best books">
    <writer name="Charlotte Lewis">
      <book name="How to begin">
        <price>EUR 55.00</price>
        <pub_year>2001</pub_year>
        <notes>The book of the year winner</notes>
      </book>
    </writer>
  </publisher>
  <publisher name="Mars books">
    <writer name="Erik Smith">
      <book name="My story as a vet and novelist">
        <price>25.00 EUR</price>
        <pub_year>2011</pub_year>
      </book>
    </writer>
  </publisher>
  <publisher name="Frog">
    <writer name="Charlotte Lewis">
      <article name="How to be efficient every day">
        <price>EUR 9.00</price>
        <pub_year>2009</pub_year>
        <illustration>
        <drawings>
          <artist name="Charles Lewis"></artist>
          <year>2008</year>
          <original>
            <publisher name="Chidrens drawings">
              <pub_year>2006</pub_year>
            </publisher>
          </original>
          <notes>watercolour</notes>
        </drawings>
        </illustration>
      </article>
    </writer>
  </publisher>
</works_www>
```

(d) short_publications_4.xml

Figure A1. (Continued)

Appendix B

Table B1. The XML relation representation of *short_publications_4.xml*

Tuple
(works_www, 'e', ⟨1⟩)
(location, 'a', ⟨1,1⟩)
(Finland, 'v', ⟨1,1,1⟩)
(publisher, 'e', ⟨1,2⟩)
(name, 'a', ⟨1,2,1⟩)
(Lion, 'v', ⟨1,2,1,1⟩)
(books, 'v', ⟨1,2,1,2⟩)
(writer, 'e', ⟨1,2,2⟩)
(name, 'a', ⟨1,2,2,1⟩)
(Erik, 'v', ⟨1,2,2,1,1⟩)
(Smith, 'v', ⟨1,2,2,1,2⟩)
(article, 'e', ⟨1,2,2,2⟩)
(name, 'a', ⟨1,2,2,2,1⟩)
(My, 'v', ⟨1,2,2,2,1,1⟩)
(story, 'v', ⟨1,2,2,2,1,2⟩)
(about, 'v', ⟨1,2,2,2,1,3⟩)
(cats, 'v', ⟨1,2,2,2,1,4⟩)
(price, 'e', ⟨1,2,2,2,2⟩)
(EUR, 'v', ⟨1,2,2,2,2,1⟩)
(15.00, 'v', ⟨1,2,2,2,2,2⟩)
(pub_year, 'e', ⟨1,2,2,2,3⟩)
(2010, 'v', ⟨1,2,2,2,3,1⟩)
(illustration, 'e', ⟨1,2,2,2,4⟩)
(drawings, 'e', ⟨1,2,2,2,4,1⟩)
(artist, 'e', ⟨1,2,2,2,4,1,1⟩)
(name, 'a', ⟨1,2,2,2,4,1,1,1⟩)
(Becky, 'v', ⟨1,2,2,2,4,1,1,1,1⟩)
(Hammer, 'v', ⟨1,2,2,2,4,1,1,1,2⟩)
(year, 'e', ⟨1,2,2,2,4,1,2⟩)
(1954-1956, 'v', ⟨1,2,2,2,4,1,2,1⟩)
(original, 'e', ⟨1,2,2,2,4,1,3⟩)
(publisher, 'e', ⟨1,2,2,2,4,1,3,1⟩)
(name, 'a', ⟨1,2,2,2,4,1,3,1,1⟩)
(Artist, 'v', ⟨1,2,2,2,4,1,3,1,1,1⟩)
(works, 'v', ⟨1,2,2,2,4,1,3,1,1,2⟩)
(pub_year, 'e', ⟨1,2,2,2,4,1,3,1,2⟩)
(1954, 'v', ⟨1,2,2,2,4,1,3,1,2,1⟩)
(original, 'e', ⟨1,2,2,2,4,1,4⟩)
(publisher, 'e', ⟨1,2,2,2,4,1,4,1⟩)
(inc, 'v', ⟨1,2,2,2,4,1,4,1,1,2⟩)
(pub_year, 'e', ⟨1,2,2,2,4,1,4,1,2⟩)
(1955, 'v', ⟨1,2,2,2,4,1,4,1,2,1⟩)
(original, 'e', ⟨1,2,2,2,4,1,5⟩)
(publisher, 'e', ⟨1,2,2,2,4,1,5,1⟩)
(name, 'a', ⟨1,2,2,2,4,1,5,1,1⟩)
(Artists, 'v', ⟨1,2,2,2,4,1,5,1,1,1⟩)
(inc, 'v', ⟨1,2,2,2,4,1,5,1,1,2⟩)
(pub_year, 'e', ⟨1,2,2,2,4,1,5,1,2⟩)
(1956, 'v', ⟨1,2,2,2,4,1,5,1,2,1⟩)
(notes, 'e', ⟨1,2,2,2,4,1,6⟩)
(pencil, 'v', ⟨1,2,2,2,4,1,6,1⟩)
(drawings, 'v', ⟨1,2,2,2,4,1,6,2⟩)
(colors, 'e', ⟨1,2,2,2,4,2⟩)
(artist, 'e', ⟨1,2,2,2,4,2,1⟩)
(name, 'a', ⟨1,2,2,2,4,2,1,1⟩)
(Lisa, 'v', ⟨1,2,2,2,4,2,1,1,1⟩)
(Bell, 'v', ⟨1,2,2,2,4,2,1,1,2⟩)
(publisher, 'e', ⟨1,3⟩)
(name, 'a', ⟨1,3,1⟩)
(The, 'v', ⟨1,3,1,1⟩)
(best, 'v', ⟨1,3,1,2⟩)
(books, 'v', ⟨1,3,1,3⟩)
(writer, 'e', ⟨1,3,2⟩)
(name, 'a', ⟨1,3,2,1⟩)
(Charlotte, 'v', ⟨1,3,2,1,1⟩)
(Lewis, 'v', ⟨1,3,2,1,2⟩)
(book, 'e', ⟨1,3,2,2⟩)
(name, 'a', ⟨1,3,2,2,1⟩)
(How, 'v', ⟨1,3,2,2,1,1⟩)
(to, 'v', ⟨1,3,2,2,1,2⟩)
(begin, 'v', ⟨1,3,2,2,1,3⟩)
(price, 'e', ⟨1,3,2,2,2⟩)
(EUR, 'v', ⟨1,3,2,2,2,1⟩)
(55.00, 'v', ⟨1,3,2,2,2,2⟩)
(pub_year, 'e', ⟨1,3,2,2,3⟩)
(2001, 'v', ⟨1,3,2,2,3,1⟩)
(notes, 'e', ⟨1,3,2,2,4⟩)
(The, 'v', ⟨1,3,2,2,4,1⟩)

Table B1. *(Continued)*

Tuple
(name, 'a', ⟨1,2,2,2,4,1,4,1,1⟩)
(Drawings, 'v', ⟨1,2,2,2,4,1,4,1,1,1⟩)
(the, 'v', ⟨1,3,2,2,4,4⟩)
(year, 'v', ⟨1,3,2,2,4,5⟩)
(winner, 'v', ⟨1,3,2,2,4,6⟩)
(publisher, 'e', ⟨1,4⟩)
(name, 'a', ⟨1,4,1⟩)
(Mars, 'v', ⟨1,4,1,1⟩)
(books, 'v', ⟨1,4,1,2⟩)
(writer, 'e', ⟨1,4,2⟩)
(name, 'a', ⟨1,4,2,1⟩)
(Erik, 'v', ⟨1,4,2,1,1⟩)
(Smith, 'v', ⟨1,4,2,1,2⟩)
(book, 'e', <1,4,2,2⟩)
(name, 'a', ⟨1,4,2,2,1⟩)
(My, 'v', ⟨1,4,2,2,1,1⟩)
(story, 'v', ⟨1,4,2,2,1,2⟩)
(as, 'v', ⟨1,4,2,2,1,3⟩)
(a, 'v', ⟨1,4,2,2,1,4⟩)
(vet, 'v', ⟨1,4,2,2,1,5⟩)
(and, 'v', ⟨1,4,2,2,1,6⟩)
(novelist, 'v', ⟨1,4,2,2,1,7⟩)
(price, 'e', ⟨1,4,2,2,2⟩)
(25.00, 'v', ⟨1,4,2,2,2,1⟩)
(EUR, 'v', ⟨1,4,2,2,2,2⟩)
(pub_year, 'e', ⟨1,4,2,2,3⟩)
(2011, 'v', ⟨1,4,2,2,3,1⟩)
(publisher, 'e', ⟨1,5⟩)
(name, 'a', ⟨1,5,1⟩)
(Frog, 'v', ⟨1,5,1,1⟩)
(writer, 'e', ⟨1,5,2⟩)
(name, 'a', ⟨1,5,2,1⟩)
(Charlotte, 'v', ⟨1,5,2,1,1⟩)
(Lewis, 'v', ⟨1,5,2,1,2⟩)
(article, 'e', ⟨1,5,2,2⟩)
(name, 'a', ⟨1,5,2,2,1⟩)
(How, 'v', ⟨1,5,2,2,1,1⟩)
(to, 'v', ⟨1,5,2,2,1,2⟩)
(be, 'v', ⟨1,5,2,2,1,3⟩)
(efficient, 'v', ⟨1,5,2,2,1,4⟩)
(every, 'v', ⟨1,5,2,2,1,5⟩)
(day, 'v', ⟨1,5,2,2,1,6⟩)
(price, 'e', ⟨1,5,2,2,2⟩)
(EUR, 'v', ⟨1,5,2,2,2,1⟩)
(9.00, 'v', ⟨1,5,2,2,2,2⟩)
(pub_year, 'e', ⟨1,5,2,2,3⟩)
(book, 'v', ⟨1,3,2,2,4,2⟩)
(of, 'v', ⟨1,3,2,2,4,3⟩)
(2009, 'v', ⟨1,5,2,2,3,1⟩)
(illustration, 'e', ⟨1,5,2,2,4⟩)
(drawings, 'e', ⟨1,5,2,2,4,1⟩)
(artist, 'e', ⟨1,5,2,2,4,1,1⟩)
(name, 'a', ⟨1,5,2,2,4,1,1,1⟩)
(Charles, 'v', ⟨1,5,2,2,4,1,1,1,1⟩)
(Lewis, 'v', ⟨1,5,2,2,4,1,1,1,2⟩)
(year, 'e', ⟨1,5,2,2,4,1,2⟩)
(2008, 'v', ⟨1,5,2,2,4,1,2,1⟩)
(original, 'e', ⟨1,5,2,2,4,1,3⟩)
(publisher, 'e', ⟨1,5,2,2,4,1,3,1⟩)
(name, 'a', ⟨1,5,2,2,4,1,3,1,1⟩)
(Chidrens, 'v', ⟨1,5,2,2,4,1,3,1,1,1⟩)
(drawings, 'v', ⟨1,5,2,2,4,1,3,1,1,2⟩)
(pub_year, 'e', ⟨1,5,2,2,4,1,3,1,2⟩)
(2006, 'v', ⟨1,5,2,2,4,1,3,1,2,1⟩)
(notes, 'e', ⟨1,5,2,2,4,1,4⟩)
(watercolour, 'v', ⟨1,5,2,2,4,1,4,1⟩)