XML Design: an FCA Point of View

Viorica Varga
Babes-Bolyai University
Cluj Napoca
400081 str. Kogalniceanu 1
Email: ivarga@cs.ubbcluj.ro

Katalin Tünde Jánosi Rancz
Hungarian University of Transylvania
Tirgu Mures
Email: tsuto@ms.sapientia.ro

Christian Săcărea
Babes-Bolyai University
Cluj Napoca
400081 str. Kogalniceanu 1
Email: csacarea@math.ubbcluj.ro

Katalin Csioban
Babes-Bolyai University
Cluj Napoca
Email: cskatyusa@yahoo.com

***Abstract*—XML (eXtensible Markup Language) documents are the main format for publishing and interchanging data on the Web. Integrity constraints are essential in data design. Functional dependencies are the most important semantic constraints. Functional dependencies satisfied by XML data have been introduced recently. Formal Concept Analysis (FCA) is a mathematical theory of concept hierarchies which is based on Lattice Theory. Data is represented as a two-dimensional context of objects and attributes. FCA discovers dependencies within the data based on the relation among objects and attributes. In this paper we take a first step towards using an FCA approach to study functional dependencies in XML databases. The novelty of our approach is the software, which analyzes an XML document, constructs the Formal Context corresponding to the flat representation of the XML data and finds the implications, which are functional dependencies in XML data.**

I. Introduction

Functional dependencies (FDs) are important in defining redundancies in relational databases [1]. The objective of normalization is to eliminate redundancies from a database or an XML document, eliminate or reduce potential update anomalies. This can be achieved by designing a redundancy-free schema so that redundant data won't take up unnecessary storage, leading to possible update anomalies and inflating data transfer cost.

Designing XML data means to choose an appropriate XML schema, which usually come in the form of DTD (Document Type Definition) or XML Scheme. A large number of classical database subjects have been reexamined in the XML context ([2], [3], [4], [5]) because XML became more and more popular. Discovering XML data redundancies from the data itself becomes necessary and is an integral part of the schema refinement (or re-design) process.

FDs are provided by database designers, but they are used in many areas as data analysis, data integration and data cleaning, and recently investigated with data mining tools [6]. Since XML databases include data which naturally provide redundancies, it is expected that FDs should also play an important role in XML databases. Accordingly, the subject of XML database design received more and more attention and many recent articles have addressed the issue of normalization in XML, while XML functional dependencies (called XFDs), and the related notion of XML normal form have recently become an important research topic.

The first authors who formally defined XML FD and normal form (XNF) were Arenas and Libkin introducing the so-called *tree tuple* approach [2]. In [7] and [3], the authors used a *path-based* approach and built their XML FD notion in a way similar to the XML Key notion proposed in [8].

Yu and Jagadish [9] show, that these XML FD notions are insufficient and propose a Generalized Tree Tuple (GTT) based XML functional dependency and key notion, which include particular redundancies involving set elements. Based on these concepts, the GTT-XNF normal form is presented too.

In later work [10], Arenas and Libkin provided a formal justification for the use of XNF in XML database design, using the classical information theory approach. A measure of the information content of data (independent of updates and queries) is introduced, as an entropy of a suitably chosen probability distribution. A formal definition of a well designed XML schema was given and the fact that XNF is both a necessary and sufficient condition for an XML schema to be well designed was proved.

Vincent et al. in [4] investigated the problem of justifying XML normal forms [3], in the terms of *closest node* XFDs using redundancy elimination. In [4], a normal form for XML documents is proposed and it has been proved to be a necessary and sufficient condition for the elimination of redundancy.

Formal Concept Analysis (FCA) is a mathematical theory on which Conceptual Knowledge Processing and Representation is grounded. Facing the problem of Knowledge Discovery, Processing and Representation in large databases, FCA enables methods of identifying patterns in data, the so-called *concepts*, structuring them in conceptual hierarchies by an order relation, called *subconcept-superconcept relation* and displaying all the relevant knowledge which can be extracted from the data set under analysis [11]. Even more, the internal logic of data can also be displayed, by means of the so-called *implications*, which proved to be the proper framework to describe functional dependencies with FCA tools.

The first authors who presented the problem of finding functional dependencies using FCA were Ganter and Wille [11]. They presented a method to define functional dependencies in multivalued contexts. Different authors have considered in [12] and [13] the use of FCA in order to describe and mine functional dependencies, presenting also efficient algorithms for extracting functional dependencies. Hereth has already described in [14] the relationship between FCA and functional dependencies by the so-called formal context of functional dependencies. Implications in this context describe exactly the functional dependencies.

The paper [15] presents an FCA based approach to detect functional dependencies in a relational database table. The present paper is devoted to extend these concepts to XML data. We apply FCA to uncover functional dependencies in XML. The novelty of this paper relies in the specially developed software, which reads an XML document, constructs the formal context corresponding to the flat representation of the XML data. The corresponding conceptual hierarchy is computed using Conexp [16]. Then, the list of implications is determined, these implications being exactly the functional dependencies in the analyzed XML data.

II. Formal Concept Analysis notions

There is a long philosophical tradition in investigating concepts, ordering them in a certain hierarchy of subconcept-superconcept. Traditionally, a concept is determined by its *extent* and its *intent* (or comprehension). The extent of a concept consists of all objects, individuals or entities which belong to the concept, while the intent consists of all properties which are considered valid for that concept. The hierarchy of concepts is given by the relation of subconcept wrt. certain superconcept, i.e., the extent of a subconcept is part of the extent of the superconcept, while the inverse relation holds for the corresponding intents. Formal Concept Analysis was born from this vigorous philosophical tradition.

As a mathematical theory, Formal Concept Analysis is based on the formalization of the notion of concept and of the medium from where this concept arises, the formal context. Formally speaking a formal context (mathematically defined in the next section) is a triple consisting of two sets and a binary relation between them. Despite of its simplicity, a formal context encodes in its incidence relation some structural information which can be found in the so-called formal concepts, which can be seen as a kind of closed pieces of the information encoded in the considered formal context.

A. Context and Concept

As we have seen before, Formal Concept Analysis is based on a set theoretical model proposing a new paradigm of thinking. A **formal context** $\mathbb{K} := (G, M, I)$ consists of two sets G and M and a binary relation I between G and M. The elements of G are called **objects** (in German *Gegenstände*) and the elements of M are called **attributes** (in German *Merkmale*). The relation I is called the incidence relation of the formal context, and we sometimes write gIm instead of $(g, m) \in I$. If gIm holds, we say that *the object g has the attribute m*.

A small context is usually represented by a cross table, i.e., a rectangular table of crosses and blanks, where the rows are labeled by the objects and the columns are labeled by the attributes. A cross in entry (g, m) indicates gIm.

For a set $A \subseteq G$ of objects we define

$$A' := \{m \in M \mid gIm \text{ for all } g \in A\}$$

the set of all attributes common to the objects in A. Dually, for a set $B \subseteq M$ of attributes we define

$$B' := \{g \in G \mid gIm \text{ for all } m \in B\}$$

the set of all objects which have all attributes in B.

A **formal concept** of the context $\mathbb{K} := (G, M, I)$ is a pair (A, B) where $A \subseteq G$, $B \subseteq M$, $A' = B$, and $B' = A$. We call A the **extent** and B the **intent** of the concept (A, B). The set of all concepts of the context (G, M, I) is denoted by $\mathfrak{B}(G, M, I)$.

B. Implications

Formally, an *implication between attributes* in M, given a formal context (G, M, I), is a pair of subsets A and B of M, which is denoted by $A \to B$. This situation is formally stated in [11]:

Definition 1 *A subset $T \subseteq M$* **respects** *an implication $A \to B$ if $A \subsetneq T$ or $B \subseteq T$. T* **respects** *a set $\mathcal{L}$ of implications if T respects every single implication in $\mathcal{L}$. $A \to B$* **holds** *in a set $\{T_1, T_2, \dots\}$ of subsets if each of the subset T_i respects the implication $A \to B$.*

We say that the implication $A \to B$ **holds** *in a context (G, M, I) if it holds in the system of object intents. In this case we also say that $A \to B$ is* **an implication in the context** *(G, M, I) or, equivalently that withing the context (G, M, I), A is the* **premise** *of B.*

C. Power context families

Power context families proved to be the proper frame to describe both semantics of concept graphs used in Contextual Logic, and functional dependencies for relational databases.

Definition 2 *A* power context family *is a sequence $\overrightarrow{\mathbb{K}} := (\mathbb{K}_0, \mathbb{K}_1, \mathbb{K}_2, \dots)$ of formal contexts $\mathbb{K}_j := (G_j, M_j, I_j)$ with $G_j \subseteq (G_0)^j$ for $j \in \mathbb{N} \setminus \{0\}$. The formal concepts of $\mathbb{K}_j$ with $j \in \mathbb{N} \setminus \{0\}$ are called* relation concepts*, because their extents represent k-ary relations on the object set G_0.*

III. Designing XML Data

We provide in the following a list of definitions. Consider the following pairwise disjoint sets: El of element names, Att of attribute names, Str of possible values of string-valued attributes, $Vert$ of node identifiers. Attribute names starts with the symbol @. The symbols **S** and $\perp$ are reserved.

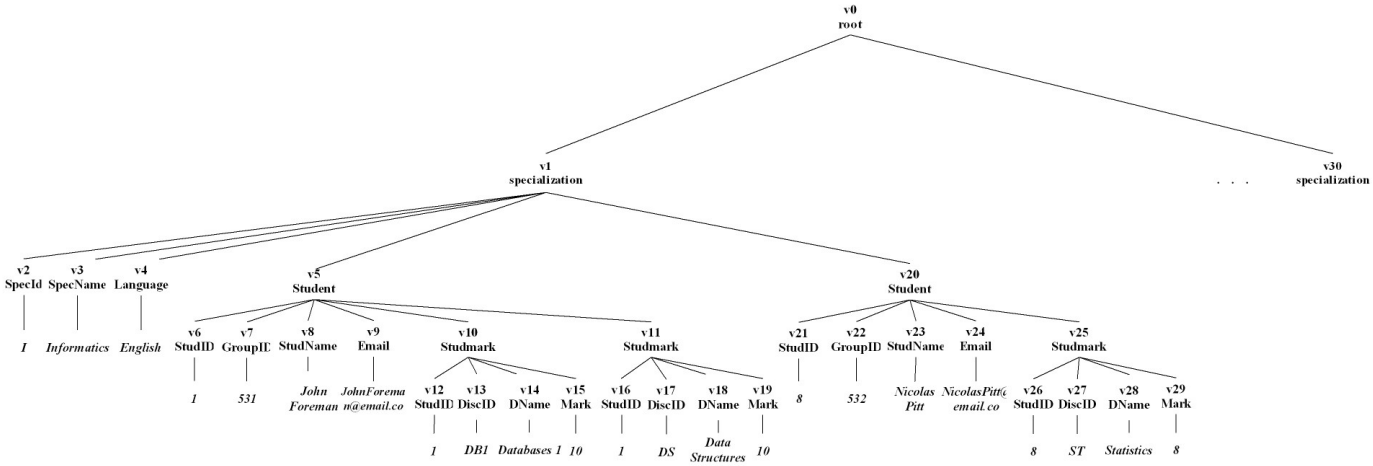

Fig. 1. XML tree example

Definition 3 (DTD) *A DTD (Document Type Definition) is defined to be a tuple* $D := (E, A, P, R, r)$*, where:*

- $E \subseteq El$ *is a finite set of element types.*
- $A \subseteq Att$ *is a finite set of attributes.*
- P *is a mapping from* E *to element type definitions: Given* $\tau \in E$*,* $P(\tau) = \mathbf{S}$ *or* $P(\tau)$ *is a regular expression* α *defined as:*

$$\alpha \ ::= \ \varepsilon \ \mid \ \tau' \ \mid \ \alpha \,|\, \alpha \ \mid \ \alpha, \alpha \ \mid \ \alpha^*$$

  where ε *is the empty sequence,* $\tau' \in E$ *and "|" denote union, "," means concatenation and "*" represents the Kleene closure.*
- R *is a mapping from* E *to the powerset of* A*. If* $@l \in R(\tau)$*, we say that* $@l$ *is defined for* τ*.*
- $r \in E$ *is called the element type of the root. We assume that* r *does not occur in* $P(\tau)$ *for any* $\tau \in E$*.*

The symbol ε represents element type declaration **EMPTY**, while $\mathbf{S}$ represents **#PCDATA**.

Definition 4 (DTD path) *Given a DTD* $D = (E, A, P, R, r)$*, a string* $w = w_1, \ldots, w_n$ *is a path in* D *if* $w_1 = r$*,* w_i *is in the alphabet of* $P(w_i)$*, for each* $i \in \{2, n-1\}$*, and* w_n *is in the alphabet of* $P(w_{n-1})$ *or* $w_n = @l$ *for some* $@l \in R(w_{n-1})$*.*

We define *length(w)* as n and $last(w)$ as w_n. The set of all paths in D is denoted by $paths(D)$. We define

$$EPaths(D) = \{p \in paths(D) \mid last(p) \in E\}$$

as the set of all paths in D that ends with an element type (not by an attribute or $\mathbf{S}$).

Definition 5 (XML Tree) *An XML tree* T *is defined to be a tree* $(V, lab, ele, att, root)$*, where*

- $V \subseteq Vert$ *is a finite set of vertices (nodes).*
- $lab : V \rightarrow El$ *assigns a label to each node of the tree.*
- $ele : V \rightarrow Str \cup V^*$ *assigns to each node a string or an ordered set of nodes as its children.*
- $attr\colon V \times Att \rightarrow Str$ *is a partial function. For each* $v \in V$*, the set* $\{@l \in Att \mid att(v, @l)$ *is defined*$\}$ *is required to be finite.*
- $root \in V$ *is called the root of T.*

We assume that there is a parent-child relation on the nodes of a tree: $\{(v, v') \in V \times V \mid v'$ occurs in $ele(V)\}$. For each $v \in V$, the elements $v' \in V$ that occur in $ele(V)$ are called $subelements$ or $children$ of v, and the elements of the set $\{@l \in Att \mid att(v, @l)$ *is defined* $\}$ are called $attributes$ of node v. For each $v \in V$, $lab(V)$ is refered as the $type$ of node v. If v is a node of T and $@l \in Att$ such that $att(v, @l)$ is defined, we will use the notation $v.@l$ instead of $att(v, @l)$.

Definition 6 (XML tree conforms DTD) *Given a DTD* $D = (E, A, P, R, r)$ *and an XML tree* $T = (V, lab, ele, att, root)$*, we say that* T *conforms to* D*, we write* $T \models D$*, if*

- $lab\colon V \rightarrow E$ *is a mapping,*
- *for each* $v \in V$*, if* $P(lab(v)) = \mathbf{S}$*, then* $ele(v) = [s]$ *for some* $s \in Str$*. Otherwise,* $ele(v) = [v_1, \ldots, v_n]$*, where the string* $lab(v_1) \ldots lab(v_n)$ *is in the language defined by* $P(lab(v))$*,*
- $att\colon V \times A \rightarrow Str$ *is a partial function such that for every* $v \in V$ *and* $@l \in A$*,* $att(v, @l)$ *is defined if and only if* $@l \in R(lab(v))$*,*
- $lab(root) = r$*.*

Example 7 *Consider the following DTD, that describes a part of a university database. The XML tree which conforms to this DTD is represented in Figure 1. We will perform an analysis on it, in order to find functional dependencies.*

```
<!ELEMENT root (specialization*)>
<!ELEMENT specialization (SpecID,
        SpecName, Language, Student*)>
```

```
<!ELEMENT Student (StudID, GroupID,
        StudName, Email, Studmark*)>
<!ELEMENT Studmark (StudID, DiscID,
        DName, Mark)>
<!ELEMENT SpecID (#PCDATA)>
<!ELEMENT SpecName (#PCDATA)>
<!ELEMENT Language (#PCDATA)>
<!ELEMENT StudID (#PCDATA)>
<!ELEMENT GroupID (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT StudName (#PCDATA)>
<!ELEMENT DiscID (#PCDATA)>
<!ELEMENT DName (#PCDATA)>
<!ELEMENT Mark (#PCDATA)>
```

The notion of $path$ is used to navigate and query XML trees and also to define constraints for XML data.

Definition 8 (Tree Path) *Let $T = (V, lab, ele, att, root)$ be a tree, a path in T is a string $w = w_1 \dots w_n$, where $w_1 \dots w_{n-1} \in El$ and $w_n \in El \cup Att \cup \{\mathbf{S}\}$, such that there are vertices $v_1, \dots, v_{n-1}$ in V with labels $w_1, \dots, w_{n-1}$, such that:*

- *v_{i+1} is a child of $v_i, i \in \{1, n-2\}$,*
- *if $w_n \in El$ then v_{n-1} has a child v_n labeled with w_n. If $w_n = @l$ is an attribute in Att, then $att(v_{n-1}, @l)$ is defined. If $w_n = \mathbf{S}$, then v_{n-1} has a child in Str.*

The set of all paths in a tree T that start from the root is denoted by $paths(T)$. Given two nodes x and y in T such that y is a descendant of x, we say that $w_1 \dots w_n$ is a path from x to y if in the above definition we have $x = w_1$ and $y = w_n$.

Definition 9 (Path prefix) *Given two paths $p = w_1 \dots w_k$ and $p' = w'_1 \dots w'_h$, p is a prefix of p' if and only if $k \leq h$ and $w_i = w'_i$ for all $i \in \{1, \dots, k\}$.*

There are different definitions in order to express the satisfaction of a functional dependency by an XML tree. Most of them define a functional dependency as an expression of the form $p_1, \dots, p_n \rightarrow q$, where $p_1, \dots, p_n$,q are path expressions.

Arenas and Libkin [2] use a relational representation of XML documents to define the satisfaction of *functional dependencies*. This relational representation is based on the notion of *tree tuples*. Given an XML tree T that conforms to a DTD D, a tree tuple is intuitively a subtree of T with the same root that contains at most one occurrence of every path. Then satisfaction is defined in the usual way: if two tree tuples in a tree agree on all the paths $p_1, \dots, p_n$, then they must agree on q. A tree tuple may not be defined on some paths, as tree tuples have at most one occurrence of every path and may have zero occurrences. Let $\perp$ represent such missing values.

Definition 10 (Tree tuple) *Given a DTD $D = (E, A, P, R, r)$ and an XML tree $T = (V, lab, ele, att, root)$, such that T conforms to D, ($T \models D$) a tree tuple t in T is formally defined as a function from $paths(D)$ to $Vert \cup Str \cup \{\perp\}$ such that if for an element path q with $last(q) = a$ we have $t(q) \neq \perp$, then*

- *$t(q) \in V$ and $lab(t(q)) = a$,*
- *if path q' is a prefix of path q, then $t(q') \neq \perp$ and $t(q')$ lies on the path from the root to $t(q)$ in T,*
- *if $@l$ is defined for $t(q)$ and its value is $s \in Str$, then $t(q.@l) = s$.*

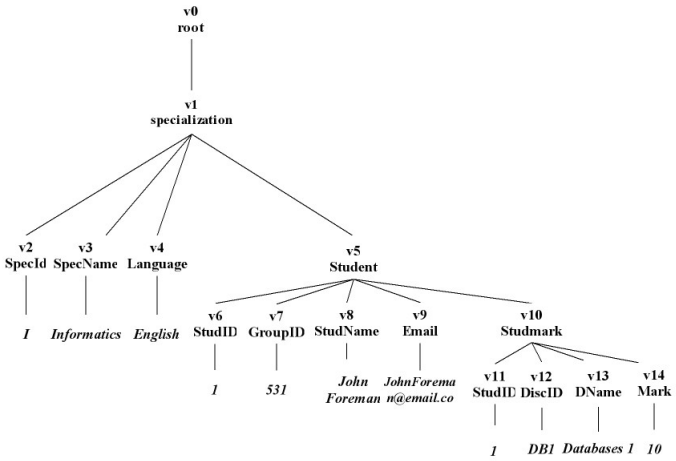

Fig. 2. A tree tuple

A tree tuple t in T is maximal if there is no other tree tuple t' in T that is obtained by only replacing some null values in t with values from $V \cup Str$. The set of maximal tree tuples in T is denoted by $tuples_D(T)$. An example of a tree tuple from the tree that conforms DTD from Example 7 is presented in Figure 2.

Definition 11 (XFD) *A functional dependency (FD) over a DTD D is an expression $\{q_1, \dots, q_n\} \rightarrow q$, where $n \geq 1$ and $q, q_1, \dots, q_n \in paths(D)$. An XML tree T that conforms to D satisfies an FD $\{q_1, \dots, q_n\} \rightarrow q$, written as $T \models \{q_1, \dots, q_n\} \rightarrow q$, if for any two tree tuples $t_1, t_2 \in tuples_D(T)$, whenever $t_1(q_i) = t_2(q_i) \neq \perp$ for all $i \in \{1, n\}$, then $t_1(q) = t_2(q)$.*

IV. Detecting XML Functional Dependencies using FCA

In order to mine functional dependencies we extend the notions introduced in [14] to XML data.

Tuple-based XML FD notion proposed in the above section suggests a natural technique for XFD discovery. We can convert the XML data into a fully unnested relation, a single relational table, and apply existing FD discovery algorithms directly. Taking this into consideration we use the definition introduced by Hereth in [14], which makes the translation from the relational table into a power context family, in order to define the formal context of functional dependencies. (See more details in [15]).

Definition 12 *Let* $\overrightarrow{\mathbb{K}}$ *be a power context family, and let* $m \in M_k$ *be an attribute of the* k*-th context. Then, the formal context of functional dependencies of* m *with regard to* $\overrightarrow{\mathbb{K}}$ *is defined as* $FD\left(m, \overrightarrow{\mathbb{K}}\right) := \left(m^{I_k} \times m^{I_k}, \{\ 1, 2, ..., k\},\ J\right)$ *with* $((g,h),i) \in J :\Leftrightarrow \pi_i(g) = \pi_i(h)$ *with* $g, h \in m^{I_k}$ *and* $i \in \{\ 1, 2, ..., k\}$.

Our approach can be described in five steps. The output of each step provides the input of the next step. The examples we provide refer XML data, which conforms DTD from Example 7. The attributes are not listed with the whole path, due to space considerations.

STEP1: We read an XML document, which contains at the beginning the schema of the data. Then, by parsing the document, we create the so called *tree tuples*, defined in [2]. Each tree tuple has the same structure and has the same number of elements. We use the *flat representation* which converts the XML data into a *flat table*. The flat table is built up by tree tuples. Each row in the table corresponds to a tree tuple in the XML tree. As shown in Figure 3, the flat representation converts the XML data which conforms DTD from Example 7 into a single relation of flat tuples.

STEP2: In the second step, we need to produce an appropriate context, in order to apply FCA. The *formal context of the functional dependencies* for the XML document has to be constructed.

The relevant information for FCA – objects, attributes and the incidence relation – is generated as follows: the objects are considered to be the tree tuple pairs, actually the tuple pairs of the flat table, while the attributes are the leaves (actually not the leaves themselves, but the nodes one level above the leaves) of the tree tuple. The incidence relation of the context shows which attributes of this tuple pairs have the same value. The formal context of the functional dependencies for XML tree which conforms DTD from Example 7 can be seen in Figure 4.

The flat representation is not an advantageous one since a lot of attributes are repeated for several times, but our goal is not to find a way of storing XML documents efficiently, but rather to find functional dependencies in their data. The analyzed XML document may have a large number of tree tuples. Creating the tree tuple pairs, our context table may have a very large number of rows. Therefore, we filter the tuple pairs and we leave out those pairs in which there are no common attributes, by an operation called clarifying the context, which does not alter the conceptual hierarchy. The application we wrote creates the corresponding tuple pairs which are written out in an output file. This file will be the input for the next step.

STEP3: Once the Formal Context of Functional Dependencies is created, we run the Concept Explorer (ConExp) [16] engine. Conexp is a tool which builds the concepts and their hierarchy.

STEP4: In this step we analyze the concept lattice we have obtained. Figure 5 depicts the *Concept Lattice* of the concrete context of FDs from Figure 4 for our example.

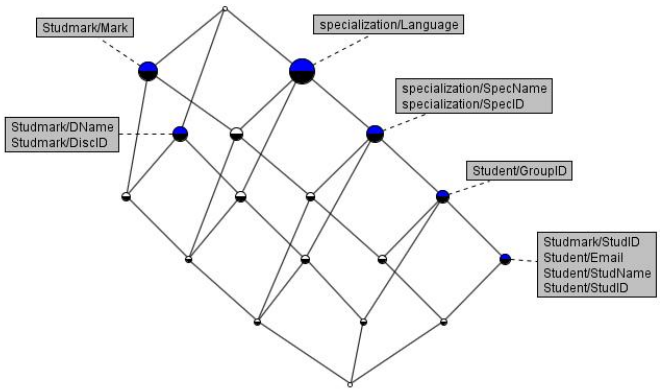

Fig. 5. Concept lattice of the FD context

An edge connects two concepts if one implies the other directly. The lattice has an interesting property: for every two concepts we choose, either one implies the other, or there exists a third concept which implies both concepts and also a fourth concept which is implicated by both concepts. Each link connecting two nodes represents the subconcept-superconcept relation between them.

We can observe that attributes listed in a vertex are related. Information about students, like `StudName`, `Email`, `StudID` are in one vertex of the lattice. The analyst can see, that StudId appears two times, one is: `specialization/students/StudID` and the second: `specialization/students/StudMark/StudID`. It is a redundancy, which can be seen from the lattice. XML data has a hierarhical structure. A lattice represent superconcept-subconcept relation, which can be applied in designing the hierarhical structure of the tree. In our example data, the type of the relations are: between languages and specializations is 1:n, between specialization and groups is 1:n, between groups and students is 1:n. These attributes can be seen one as subconcept of the other. The relation between students and marks is m:n, so there is no subconcept relation. This subconcept relation can help the XML designer to construct the XML tree hierarhical structure.

STEP5: In this step we examine the candidate concepts resulting from the previous steps and uses them to explore the dependencies. Finally, from the resulted context we generate the list of all functional dependencies. The implications in this lattice corresponds to functional dependencies in XML, as can be seen in the following: the concept labeled `students/StudID` is a subconcept of the concept labeled `students/GroupID`, in other words the concept labeled `students/StudID` implies the concept labeled `students/GroupID`. This means that in every tuple pair where the `StudID` field has the same value, the GroupID is the same. Hence, we have obtained the following implication, which is functional dependency:

$$specialization/students/StudID \rightarrow$$

SpecID	SpecName	Language	Student/ StudID	Student/ GroupID	Student/ StudName	Student/ Email	Studmark/ StudID	Studmark/ DiscID	Studmark/ DName	Studmark/ Mark
I	Informatics	English	I	531	John Foreman	JohnForeman@email.co	1	DB1	Databases 1	10
I	Informatics	English	I	531	John Foreman	JohnForeman@email.co	1	DS	Data Structures	9
I	Informatics	English	8	532	Nicolas Pitt	NicolasPitt@email.co	8	ST	Statistics	8

Fig. 3. Flat table representation of example XML data

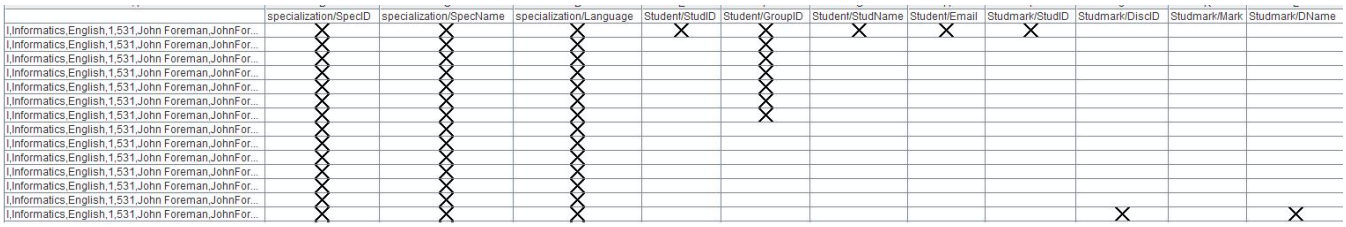

Fig. 4. Formal context of functional dependencies

1 < 531 > specialization/SpecID ==> specialization/SpecName specialization/Language;
2 < 531 > specialization/SpecName ==> specialization/SpecID specialization/Language;
3 < 314 > Studmark/DName ==> Studmark/DiscID;
4 < 314 > Studmark/DiscID ==> Studmark/DName;
5 < 216 > Student/GroupID ==> specialization/SpecID specialization/SpecName specialization/Language;
6 < 57 > Student/StudID ==> specialization/SpecID specialization/SpecName specialization/Language Student/GroupID Student/StudName Student/Email Studmark/StudID;
7 < 57 > Student/StudName ==> specialization/SpecID specialization/SpecName specialization/Language Student/StudID Student/GroupID Student/Email Studmark/StudID;
8 < 57 > Student/Email ==> specialization/SpecID specialization/SpecName specialization/Language Student/StudID Student/GroupID Student/StudName Studmark/StudID;
9 < 57 > Studmark/StudID ==> specialization/SpecID specialization/SpecName specialization/Language Student/StudID Student/GroupID Student/StudName Student/Email;

Fig. 6. Functional depedencies in XML document

$$specialization/students/GroupID$$

In the same manner, by using our approach, we can generate from the conceptual hierarchy the complete list of all functional dependencies, see Figure 6.

V. Conclusion and Future Work

Formal Concept Analysis (FCA) has been widely applied in many fields. In this paper we have proved that FCA offers the possibility of mining functional dependencies in XML data. As future work, we propose to suggest a correct XML scheme of the XML document, in which we found the functional dependecies by the software.

References

[1] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*. McGraw-Hill, Fifth Edition, 2005.

[2] M. Arenas and L. Libkin, "A normal form for xml documents," *ACM TODS*, vol. 29, no. 1, pp. 195–232, 2004.

[3] M. W. Vincent, J. Liu, and C. Liu, "Strong functional dependencies and their application to normal forms in xml," *ACM TODS*, vol. 29, no. 3, pp. 445–462, 2004.

[4] M. W. Vincent, J. Liu, and M. Mohania, "On the equivalence between fds in xml and fds in relations," *Acta Informatica*, vol. 44, no. 3-4, pp. 207–247, 2007.

[5] M. Arenas, W. Fan, and L. Libkin, "On the complexity of verifying consistency of xml specifications," *SIAM J. Comput*, vol. 38, no. 3, pp. 841–880, 2008.

[6] N. Novelli and R. Cicchetti, "Functional and embedded dependency inference: a data mining point of view," *Information Systems*, vol. 26, no. 7, pp. 477–506, 2001.

[7] M. Lee, T. Ling, and W. Low, "Designing functional dependencies for xml," in *Proceedings of the EDBT Conference*, 2002, pp. 124–141.

[8] P. Buneman, S. Davidson, W. Fan, , C. Hara, and W.-C. Tan, "Keys for xml," in *Proceedings of the WWW, Hong Kong*, 2001, pp. 201–210.

[9] C. Yu and H. V. Jagadish, "Xml schema refinement through redundancy detection and normalization," *VLDB*, vol. 17, no. 2, pp. 203–223, 2008.

[10] M. Arenas and L. Libkin, "An information-theoretic approach to normal forms for relational and xml data," *JACM*, vol. 52, no. 2, pp. 246–283, 2005.

[11] B. Ganter and R. Wille, *Formal Concept Analysis. Mathematical Foundations*. 1999, Springer.

[12] S. Lopes, J.-M. Petit, and L. Lakhal, "Functional and approximate dependency mining: database and fca points of view," in *Special issue of Journal of Experimental and Theoretical Artificial Intelligence (JETAI) on Concept Lattices for KDD*. Taylor and Francis, 2002, vol. 14, no. 2-3, pp. 93–114.

[13] ——, "Efficient discovery of functional dependencies and armstrong relations," *Advances in Database Technology EDBT*, vol. 1777, pp. 350–364, 2000.

[14] J. Hereth, "Relational scaling and databases," in *Proceedings of the 10th International Conference on Conceptual Structures: Integration and Interfaces*, ser. LNCS, vol. 2393. Springer Verlag, 2002, pp. 62–76.

[15] K. J. Rancz, V. Varga, and J. Puskas, "A software tool for data analysis based on formal concept analysis," *Studia Univ. Babeş-Bolyai, Informatica*, vol. 53, no. 2, pp. 67–78, 2008.

[16] A. S. Yevtushenko, "System of data analysis "concept explorer"," in *Proceedings of the 7th National Conference on Artificial Intelligence KII*, 2000, pp. 127–134.