STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIX**, Special Issue 2, 2014
ICFCA 2014: 12th International Conference on Formal Concept Analysis, Cluj-Napoca, June 10-13, 2014

TRIADIC APPROACH TO CONCEPTUAL DESIGN OF XML DATA

CHRISTIAN SĂCĂREA AND VIORICA VARGA

Abstract. As XML becomes a popular data representation and exchange format over the web, XML schema design has become an important research area. Discovering XML data redundancies from the data itself becomes necessary and it is an integral part of the schema refinement (or re-design) process. Different authors present the notion of functional dependency in XML data and normal forms for XML data. Yu and Yagadish (2008) give the definition of the Generalized Tree Tuple (GTT) and XNF normal form. They present also a hierarchical and a flat representation of XML data. The hierarchical representation of XML data from the paper of Yu and Yagadish (2008) is used to define a triadic FCA approach for a conceptual model of XML data. The formal tricontext of functional dependencies with respect to a tuple class is given.

1. Introduction and Previous Work

The goal of relational database design is to generate a set of relation schemas that allows us to store information without unnecessary redundancy. The relation scheme obtained by translating the Entity-Relationship model is a good starting point, but we still need to develop new techniques to detect possible redundancies in the preliminary relation scheme. The normal form satisfied by a relation is a measure of the redundancy in the relation. In order to analyze the normal form of a relation we need to detect the functional dependencies that are present in the relation.

XML is a popular data representation and exchange format over the web. XML data design must ensure that there are no unintended redundancies, since these can generate data inflation and transfer costs, as well as unnecessary storage. Hence, the good design of XML schemas is an important issue. Redundancies are captured as functional dependencies in relational databases

Received by the editors: March 25, 2014.
2010 *Mathematics Subject Classification.* 68P15, 03G10.
1998 *CR Categories and Descriptors.* H.2.1 [**D**atabase Management]: Logical design – Scheme and subschema.
Key words and phrases. XML data design, Formal Concept Analysis.

and it is expected that they play a similar role in XML databases, having specific properties and features due to the structure of XML data.

XML functional dependencies (XML FD) have become an important research topic. In 2004, Arenas and Libkin ([1]) adopted a tree tuple-based approach, defining for the first time an XML FD and a normal form. Yu and Jagadish prove in [19] that the previously introduced notions of XML FD are insufficient, and propose a generalized tree tuple-based XML FD.

Formal Concept Analysis offers an algebraic approach to data analysis and knowledge processing. Hence, it lies at hand to use FCA for mapping conceptual designs into XML schemas, since the XML data model is both hierarchical and semistructured. The notion of dependencies between attributes in a many-valued context has been introduced in [4], by Ganter and Wille. J. Hereth investigates in [6] how some basic concepts from database theory translate into the language of Formal Concept Analysis. He defines the power context family resulting from the canonical translation of a relational database. Regarding this power context family, he defines the formal context of functional dependencies. A detailed analysis and complex examples of the formal context of functional dependencies for a relational table are presented in [9]. Determining the implications in this context is investigated in [10], using a specially designed software. These are syntactically the same as functional dependencies in the analyzed table.

Uncovering functional dependencies in XML using FCA has been studied in [13]. AN XML document is read and the formal context corresponding to the flat representation of the XML data is constructed. XML data is converted into a fully unnested relation, a single relational table, and existing FD discovery algorithms are applied directly. The implications are exactly the functional dependencies in the analyzed XML data. This study is continued in [8] and [7]. Here a framework is proposed, which parses the XML document and constructs the formal context corresponding to the flat representation of the XML data. The concept lattice is a useful graphical representation of the analyzed XML document's elements and their hierarchy. Keys and functional dependencies in XML data are determined, as attribute implications in the constructed formal context. Then, the scheme of the XML document is transformed in GTT-XNF using the detected functional dependencies.

In this article the hierarchical representation of XML data from the paper of Yu and Yagadish (2008) is used to define a triadic FCA approach for a conceptual model of XML data. The novelty of the paper is this triadic approach and the proposed formal tricontext of functional dependencies with respect to a tuple class.

2. Mining Functional Dependency in XML Data

XML functional dependency has been defined in different ways, but no generally accepted definition exists. The main problem with defining functional dependency for XML databases is the absence of the definition of a tuple concept for XML. Arenas and Libkin defined tree tuples based upon Document Type Definition (DTD) schema [1]. In [13] an FCA based approach is given to find functional dependency in XML data as using the approach from [1]. In this approach, the XML document is read and then the formal context corresponding to the flat representation of the XML data is constructed. Here we derive the list of implications, these implications are exactly the functional dependencies in the analyzed flat representation of XML data.

Hartmann et al. [2, 3] define functional dependencies using the concept of tree homomorphism. Wang [16] compared different functional dependency definitions for XML and proposed a new definition of XML FD, which unifies and generalizes the surveyed XML FDs. All these XML FD definitions are based upon path expressions created from DTDs or XML Schema definitions.

Szabó and Benczúr [12] define the functional dependency concept on general regular languages, which is applicable to XML. They consider an XML document as a set of text fragments, each fragment being a string of symbols and the types of these strings are sentences of a regular language.

Yu and Jagadish [19] found that the tree tuples model of Arenas and Libkin [1] cannot handle set elements. They extend the tree tuple model as Generalized Tree Tuple (GTT) by incorporating set element type into the XML FD specification.

In [8] and its extended version [7], we propose a framework to mine FDs from an XML database; it is based on the notions of Generalized Tree Tuple, XML functional dependency and XML key notion as introduced by [19]. The formal context for a tuple class or the whole XML document is constructed from the flat representation of the generalized tree tuple. Non-leaf and leaf level elements (or attributes) and corresponding values are inserted in the formal context, then the concept lattice of the XML data is constructed. The obtained conceptual hierarchy is a useful graphical representation of the analyzed XML document's elements and their hierarchy. The software also finds the keys in the XML document. The set of implications resulted from this concept lattice will be equivalent to the set of functional dependencies that hold in the XML database. If the XML data representation is nested, solving the problem of mining XML FD's using FCA becomes more complicated and involves the use of multicontexts, tricontexts and power tricontexts families.

We start by recalling some basic definitions from [19].

Definition 1. (Schema) *A schema is defined as a set* $S = (E, T, r)$*, where:*

- *E is a finite set of element labels;*
- *T is a finite set of element types, and each $e \in E$ is associated with a $\tau \in T$, written as $(e : \tau)$, τ has the next form:*
  $\tau ::= \texttt{str} \mid \texttt{int} \mid \texttt{float} \mid \texttt{SetOf}\ \tau \mid \texttt{Rcd}[e_1 : \tau_1, \ldots, e_n : \tau_n]$*;*
- *$r \in E$ is the label of the root element, whose associated element type can not be* `SetOf` τ*.*

This definition contains some basic constructs in XML Scheme [15]. The types `str,int` and `float` are system defined *simple types* and `Rcd` indicate *complex scheme elements* (elements with children elements). Keyword `SetOf` is used to indicate *set schema elements* (elements that can have multiple matching data elements sharing the same parent in the data). We will treat attributes and elements in the same way, with a reserved "@" symbol before attributes.

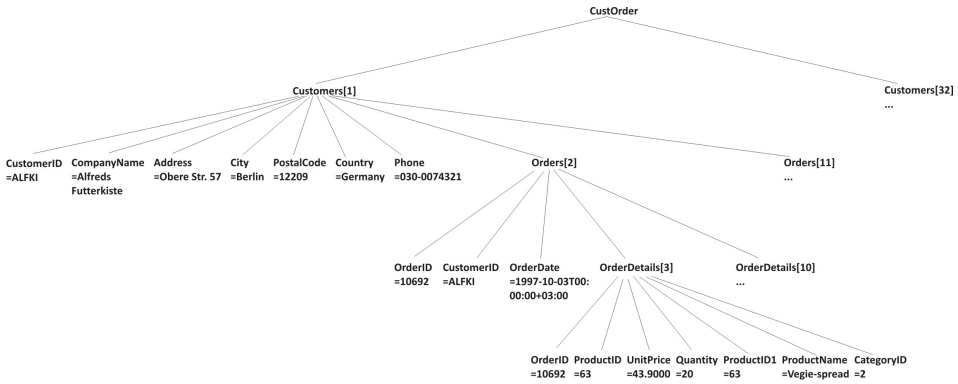

FIGURE 1. Example Tree

Example 1. The scheme $S_{CustOrder}$ of XML document from Figure 1 is:

```
CustOrder:Rcd
   Customers:SetOf Rcd
      CustomerID: str
      CompanyName: str
      Address: str
      City: str
      Country: str
      Phone: str
      Orders: SetOf Rcd
         OrderID: int
         CustomerID: str
```

```
OrderDate: str
OrderDetails: SetOf Rcd
    OrderID: int
    ProductID: int
    UnitPrice: float
    Quantity: float
    ProductName: str
    CategoryID: int
```

A schema element e_k can be identified through a path expression, $path(e_k) = /e_1/e_2/.../e_k$, where $e_1 = r$, and e_i is associated with type $\tau_i ::= \texttt{Rcd}\ [..., e_{i+1} : \tau_{i+1}, ...]$ for all $i \in [1, k-1]$. A path is *repeatable*, if e_k is a set element. We adopt XPath steps "." (self) and ".." (parent) to form a relative path given an anchor path.

Definition 2. (Data tree) *An XML database is defined to be a rooted labeled tree $T = \langle N, \mathcal{P}, \mathcal{V}, n_r \rangle$, where:*

- *N is a set of labeled data nodes, each $n \in N$ has a label e and a node key that uniquely identifies it in T;*
- *$n_r \in N$ is the root node;*
- *$\mathcal{P}$ is a set of parent-child edges, there is exactly one $p = (n', n)$ in $\mathcal{P}$ for each $n \in N$ (except n_r), where $n' \in N, n \neq n', n'$ is called the parent node, n is called the child node;*
- *$\mathcal{V}$ is a set of value assignments, there is exactly one $v = (n, s)$ in $\mathcal{V}$ for each leaf node $n \in N$, where s is a value of simple type.*

We assign a node key, referred to as @key, to each data node in the data tree in a pre-order traversal. A data element n_k is a descendant of another data element n_1 if there exists a series of data elements n_i, such that $(n_i, n_{i+1}) \in \mathcal{P}$ for all $i \in [1, k-1]$. Data element n_k can be addressed using a path expression, $path(n_k) = /e_1/ \ldots /e_k$, where e_i is the label of n_i for each $i \in [1, k]$, $n_1 = n_r$, and $(n_i, n_{i+1}) \in \mathcal{P}$ for all $i \in [1, k-1]$.

A data element n_k is called *repeatable* if e_k corresponds to a set element in the schema. Element n_k is called a *direct descendant* of element n_a, if n_k is a descendant of n_a, $path(n_k) = \ldots /e_a/e_1/ \ldots /e_{k-1}/e_k$, and e_i is not a set element for any $i \in [1, k-1]$.

In considering data redundancy, it is important to determine the equality between the "values" associated with two data elements, instead of comparing their "identities" which are represented by @key. So, we have:

Definition 3. (Element-value equality) *Two data elements n_1 of $T_1 = \langle N_1, \mathcal{P}_1, \mathcal{V}_1, n_{r1} \rangle$ and n_2 of $T_2 = \langle N_2, \mathcal{P}_2, \mathcal{V}_2, n_{r2} \rangle$ are element-value equal (written as $n_1 =_{ev} n_2$) if and only if:*

- n_1 *and* n_2 *both exist and have the same label;*
- *There exists a set* M*, such that for every pair* $(n'_1, n'_2) \in M$*,* $n'_1 =_{ev} n'_2$*, where* n'_1, n'_2 *are children elements of* n_1, n_2*, respectively. Every child element of* n_1 *or* n_2 *appears in exactly one pair in* M*.*
- $(n_1, s) \in \mathcal{V}_1$ *if and only if* $(n_2, s) \in \mathcal{V}_2$*,where* s *is a simple value.*

Definition 4. (Path-value equality) *Two data element paths* p_1 *on* $T_1 = \langle N_1, \mathcal{P}_1, \mathcal{V}_1, n_{r1}\rangle$ *and* p_2 *on* $T_2 = \langle N_2, \mathcal{P}_2, \mathcal{V}_2, n_{r2}\rangle$ *are path-value equal (written as* $T_1.p_1 =_{pv} T_2.p_2$*) if and only if there is a set* M' *of matching pairs where*

- *For each pair* $m' = (n_1, n_2)$ *in* M'*,* $n_1 \in N_1$*,* $n_2 \in N_2$*,* $path(n_1) = p_1$*,* $path(n_2) = p_2$*, and* $n_1 =_{ev} n_2$*;*
- *All data elements with path* p_1 *in* T_1 *and path* p_2 *in* T_2 *participate in* M'*, and each such data element participates in only one such pair.*

The definition of functional dependency in XML data needs the definition of so called Generalized Tree Tuple.

Definition 5. (Generalized tree tuple) *A generalized tree tuple of data tree* $T = \langle N, \mathcal{P}, \mathcal{V}, n_r\rangle$*, with regard to a particular data element* n_p *(called pivot node), is a tree* $t^T_{n_p} = \langle N^t, \mathcal{P}^t, \mathcal{V}^t, n_r\rangle$*, where:*

- $N^t \subseteq N$ *is the set of nodes,* $n_p \in N^t$ *;*
- $\mathcal{P}^t \subseteq \mathcal{P}$ *is the set of parent-child edges;*
- $\mathcal{V}^t \subseteq \mathcal{V}$ *is the set of value assignments;*
- n_r *is the same root node in both* $t^T_{n_p}$ *and* T *;*
- $n \in N^t$ *if and only if: 1)* n *is a descendant or ancestor of* n_p *in* T*, or 2)* n *is a non-repeatable direct descendant of an ancestor of* n_p *in* T *;*
- $(n_1, n_2) \in \mathcal{P}^t$ *if and only if* $n_1 \in N^t$ *,* $n_2 \in N^t, (n_1, n_2) \in \mathcal{P}$*;*
- $(n, s) \in \mathcal{V}^t$ *if and only if* $n \in N^t$*,* $(n, s) \in \mathcal{V}$*.*

A generalized tree tuple is a data tree projected from the original data tree. It has an extra parameter called a pivot node. In contrast with the notion of a tree tuple defined in [1], which separate sibling nodes with the same path at all hierarchy levels, the generalized tree tuple separate sibling nodes with the same path above the pivot node. An example of a generalized tree tuple is given in Figure 2. Based on the pivot node, generalized tree tuples can be categorized into tuple classes:

Definition 6. (Tuple class) *A tuple class* C^T_p *of the data tree* T *is the set of all generalized tree tuples* t^T_n*, where* $path(n) = p$*. Path* p *is called the pivot path.*

Definition 7. (XML FD) *An XML FD is a triple* $\langle C_p, LHS, RHS\rangle$*, (LHS for Left Hand Side part of FD and RH for Right Hand Side) written as* $LHS \rightarrow$

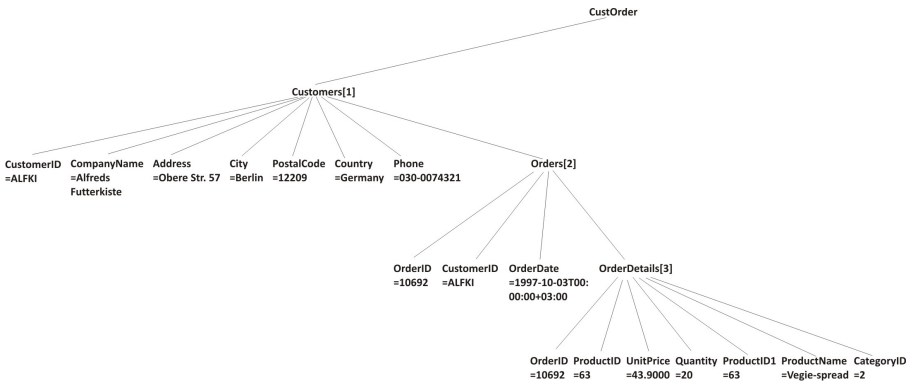

FIGURE 2. Example tree tuple

RHS w.r.t. C_p, where C_p denotes a tuple class, LHS is a set of paths (P_{li} , $i = [1, n]$) relative to p, and RHS is a single path (P_r) relative to p.

An XML FD holds on a data tree T (or T satisfies an XML FD) if and only if for any two generalized tree tuples $t_1, t_2 \in C_p$

- $\exists i \in [1, n]$, $t_1.P_{li} = \perp$ or $t_2.P_{li} = \perp$, or

- If $\forall i \in [1, n]$, $t_1.P_{li} =_{pv} t_2.P_{li}$, then $t_1.P_r \neq \perp, t_2.P_r \neq \perp, t_1.P_r =_{pv} t_2.P_r$.

A null value, $\perp$, results from a path that matches no node in the tuple, and $=_{pv}$ is the path-value equality defined in Definition 4.

Example 2. (XML FD) In our running example whenever two products agree on `ProductID` values, they have the same `ProductName`. This can be formulated as follows:
`{./ProductID}` $\rightarrow$ `./ProductName` w.r.t $C_{OrderDetails}$
Another example is:
`{./ProductID}` $\rightarrow$ `./CategoryID` w.r.t $C_{OrderDetails}$

In our approach we find the XML keys of a given XML document, so we need the next definition:

Definition 8. (XML key) *An XML Key of a data tree T is a pair $\langle C_p, LHS \rangle$, where T satisfies the XML FD $\langle C_p, LHS, ./@key \rangle$.*

Example 3. We have the XML FD: $\langle C_{Orders}, ./OrderID, ./@key \rangle$, which implies that $\langle C_{Orders}, ./OrderID \rangle$ is an XML key.

Tuple classes with repeatable pivot paths are called *essential tuple classes.*

Definition 9. (Interesting XML FD) *An XML FD $\langle C_p, LHS, RHS \rangle$ is interesting if it satisfies the following conditions:*

- *$RHS \notin LHS$;*

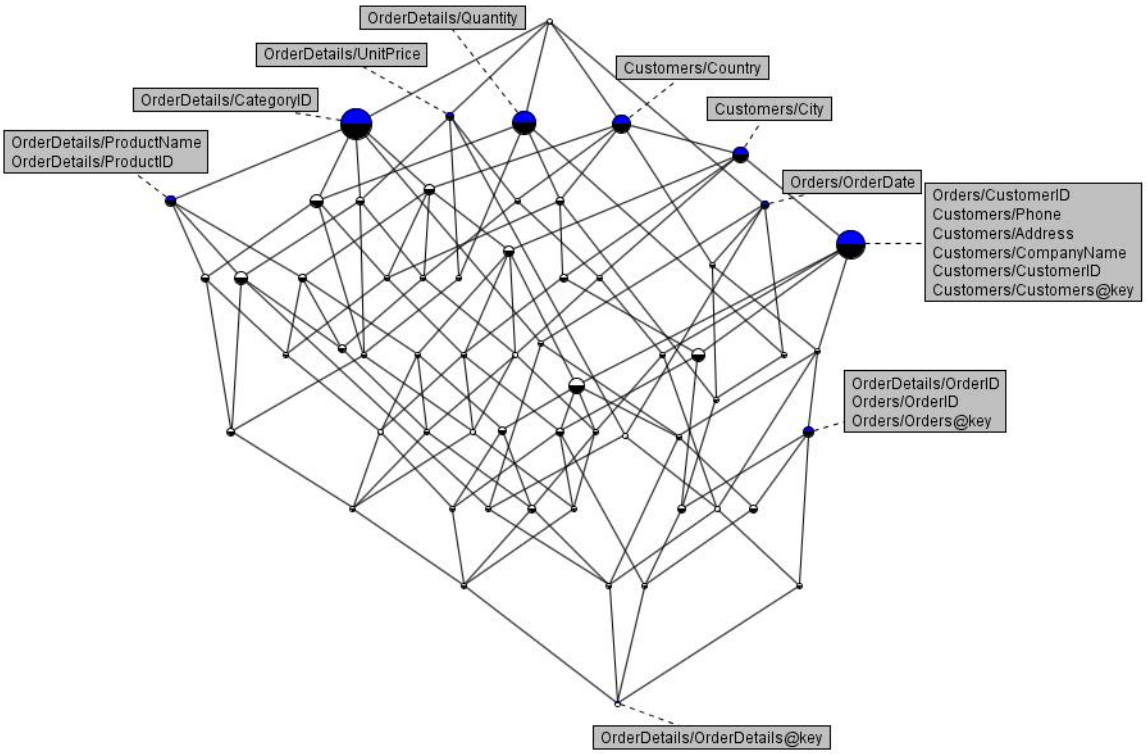

FIGURE 3. Concept Lattice of functional dependencies' Formal Context for tuple class $C_{Customers}$

- C_p *is an essential tuple class;*
- *RHS matches to descendent(s) of the pivot node.*

Definition 10. (XML data redundancy) *A data tree T contains a redundancy if and only if T satisfies an interesting XML FD $\langle C_p, LHS, RHS\rangle$, but does not satisfy the XML Key $\langle C_p, LHS\rangle$.*

As pointed out by [19], the hierarchical representation of XML data avoids many redundancies compared with the flat representation of XML data. Hence, we can separate individual relations like $R_{Customers}$, R_{Orders} or R_{Film}, R_{Actor} as in Tables 3 and 4. There are two types of relevant XML functional dependencies. The *intra-relational* XML FD's are the XML FD's whose LHS and RHS paths are in the same relation. Hence, an intrarelational FD is one that involves a single relation. As highlighted by [19], most of the interesting FD's in XML datasets are not intra-relational., i.e., they do not contain only LHS

or RHS paths within the same relation. Consider the XML data set in Figure 6. Then, intra-relational FD's are `SName` $\rightarrow$ `Founded`, `SName` $\rightarrow$ `film`, and `film/Title, film/Year` $\rightarrow$ `film/Director, film/actor` in R_{Studio}.

3. FCA Grounded Discovery of Intra-relational Functional Dependencies in XML Data

In order to mine intra-relational functional dependencies using FCA, we can use the same procedure as in the flat representation of XML datasets. This algorithm is presented in detail in [8] and [7], in the following we give just a sketch of how it is done.

Consider, the tuple class $C_{Customers}$. First, we construct the formal context of functional dependencies for XML data, see [7]. The concept lattice of this context is represented in Figure 3. The concept lattice displays also the hierarchy of the analyzed data. For instance, the node labeled $Customers/Country$ is on a higher level than the node labeled $Customers/City$. The $Customers$ node with six attributes is a subconcept of the concept labeled $Customers/City$. In our XML data, every customer has different name, address, phone number, so these attributes appear in one concept node and imply each other.

We can also observe, that the information about $Products$ is displayed on the other side of the lattice. $Products$ are in many-to-many relationships with $Orders$, linked by $OrderDetail$ in this case. The specially designed software FCAMineXFD mines the functional dependencies. A part of these XML FD-s are shown in Figure 4.

Given the set of dependencies discovered by this tool, we adopt the normalization algorithm of [19] to convert one XML schema into a correct one. The resulting scheme is shown in Figure 5.

4. Mining Inter-relational Functional Dependencies in XML Data

XML data, due to its specificity, has two different representations: a flat and hierarchical (non-flat) representation. XML elements may be simple elements but they also may nest other elements. Consider the XML document from Figure 6. It displays information extracted from a movie database concerning film studios, films, actors, etc.

The XML schema notation (XSN) allows to specify sequences and choices of elements. The scheme $S_{MoviesDB}$ of XML document in XSN from Figure 6 is given by:

```
MoviesDB (studio*)
   studio (SName, Founded, film*)
         film (Title, Year, Director, actor*)
```

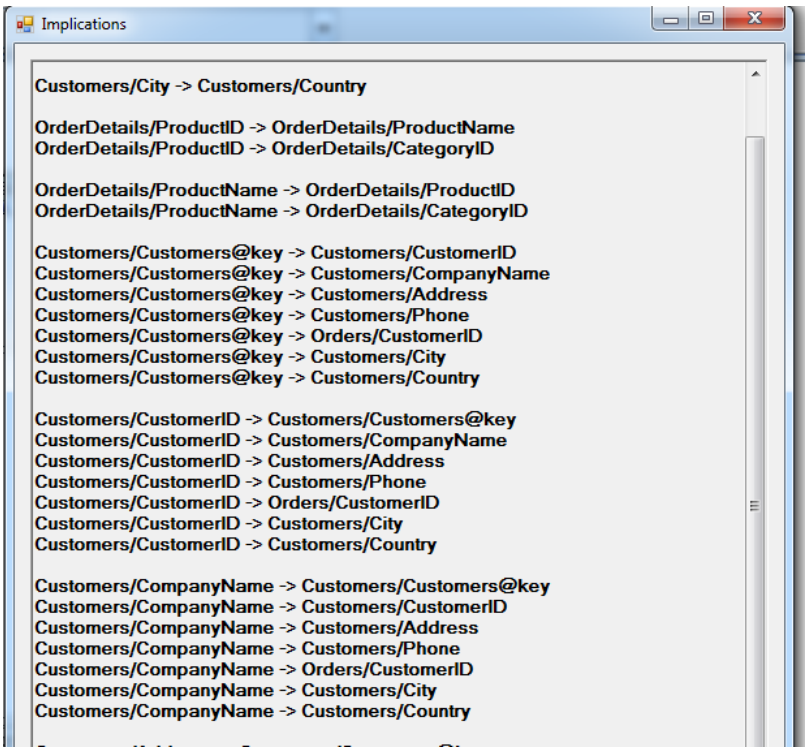

FIGURE 4. Functional dependencies in tuple class $C_{Customers}$

TABLE 1. Table R_{root}

@key	parent
1	$\perp$

TABLE 2. Table R_{Studio}

@key	parent	SName	Founded
10	1	Columbia Pictures	1924
50	1	Warner Bros. Pictures	1923

```
actor (AName, Gender, Born, BornY?)
```

In the flat representation, the data tree is represented as a single relational table. The hierarchical representation is more compact. The original XML tree is represented by a set of nested relations based on the XML schema, each relation R_p corresponds to an essential tuple class C_p (see [19]). In our movie

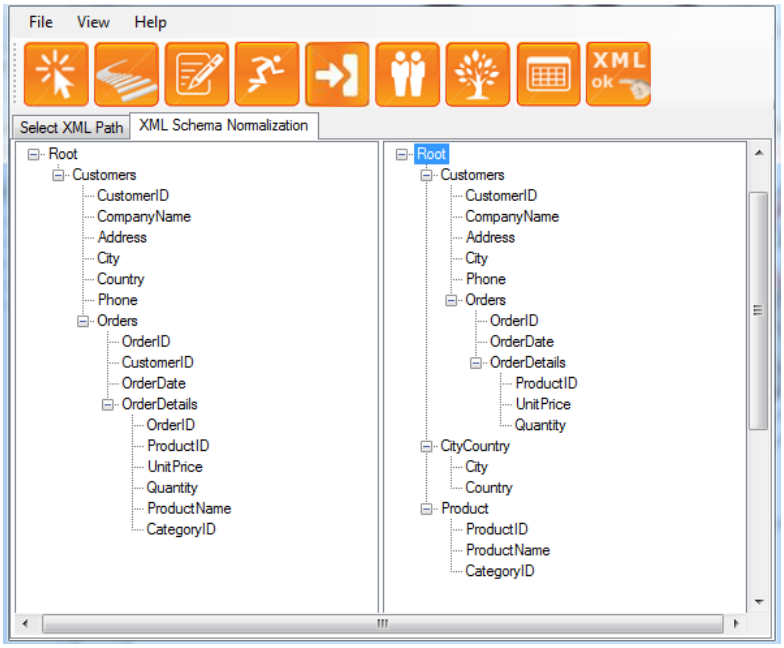

FIGURE 5. Correct XML Scheme

TABLE 3. Table R_{Film}

@key	parent	Title	Year	Director
13	10	Da Vinci Code	2006	Ron Howard
30	10	Captain Phillips	2013	Paul Greengrass
55	50	Extremely Loud & Incredibly Close	2011	Stephen Daldry
80	50	Gravity	2013	Alphonso Cuarón

TABLE 4. Table R_{Actor}

@key	parent	AName	Gender	Born	BornYear
20	13	Tom Hanks	M	USA	1956
25	13	Audrey Tautou	F	France	$\perp$
35	30	Tom Hanks	M	USA	1956
40	30	Barkhad Abdi	M	Somalia	$\perp$
60	55	Thomas Horn	M	USA	$\perp$
65	55	Tom Hanks	M	USA	1956
70	55	Sandra Bullock	F	USA	$\perp$
85	80	Sandra Bullock	F	USA	1964

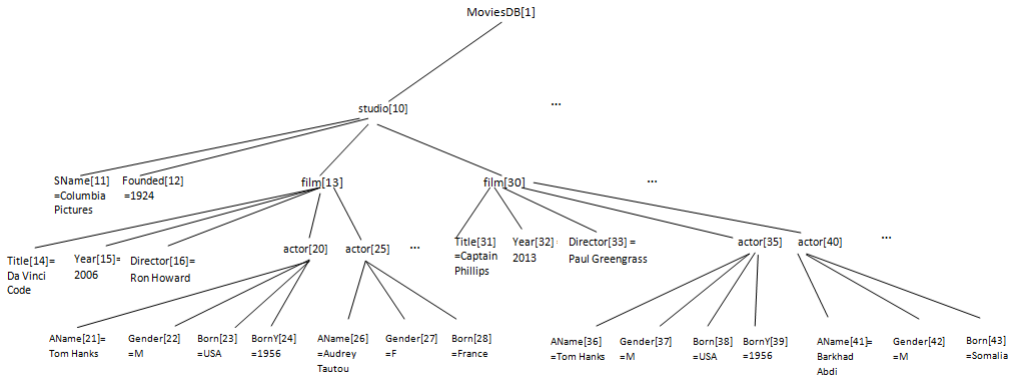

FIGURE 6. Movie Tree

dataset example, the hierarchical representation of the XML data from Figure 6 is given in the nested tables 1 - 4.

Every nested table is considered as a many-valued context. Through conceptual scaling, we obtain a multi-context wherefrom we can construct a tricontext.

In the following, we briefly recall some definitions.

Definition 11. A triadic formal context (shortly tricontext) is a quadruple $\mathbb{K} := (K_1, K_2, K_3, Y)$ where K_1, K_2 and K_3 are sets, and Y is a ternary relation between them, i. e., $Y \subseteq K_1 \times K_2 \times K_3$. The elements of K_1, K_2 and K_3 are called (formal) objects, attributes, and conditions, respectively. An element $(g, m, b) \in Y$ is read *object g has attribute m under condition b.*

Definition 12 ([17])**.** A multicontext of signature $\sigma\colon P \to I^2$, where I and P are non-empty sets, is be defined as a pair (S_I, R_P) consisting of a family $S_I := (S_i)_{i\in I}$ of sets and a family $R_P := (R_p)_{p\in P}$ of binary relations with $R_p \subseteq S_i \times S_j$ if $\sigma p = (i, j)$. A multicontext $\mathbb{K} := (S_I, R_P)$ can be understood as a *network* of formal contexts $\mathbb{K}_p := (S_i, S_j, R_p)$, with $p \in P$ and $\sigma p = (i, j)$. According to this understanding, the conceptual structure of a multicontext $\mathbb{K}$ is constituted by the concept lattices of its components $\mathbb{K}_p$.

In our example, every many-valued context representing a nested table of the XML dataset is nominally scaled. We obtain four contexts, which alltogether form the multicontext $\mathbb{K}_{Movie}$.

Example 4. The conceptual structure of the *Movie* multicontext is displayed in the Figures 7-10.

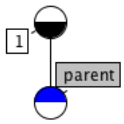

FIGURE 7. R_{root}

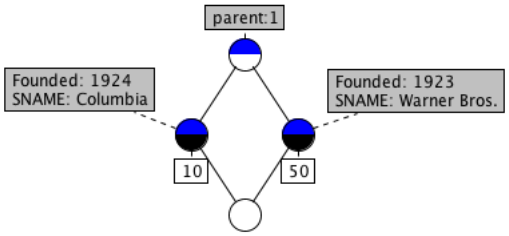

FIGURE 8. R_{studio}

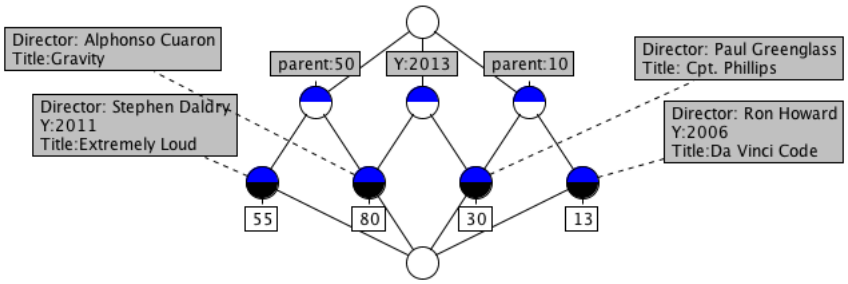

FIGURE 9. R_{film}

For a multicontext $\mathbb{K} := (S_I, R_P)$ of signature $\sigma\colon P \to I^2$, let $I_1 := \{i \in I \mid \sigma p = (i, j) \text{ for some } p \in P\}$ and $I_2 := \{j \in I \mid \sigma p = (i, j) \text{ for some } p \in P\}$.

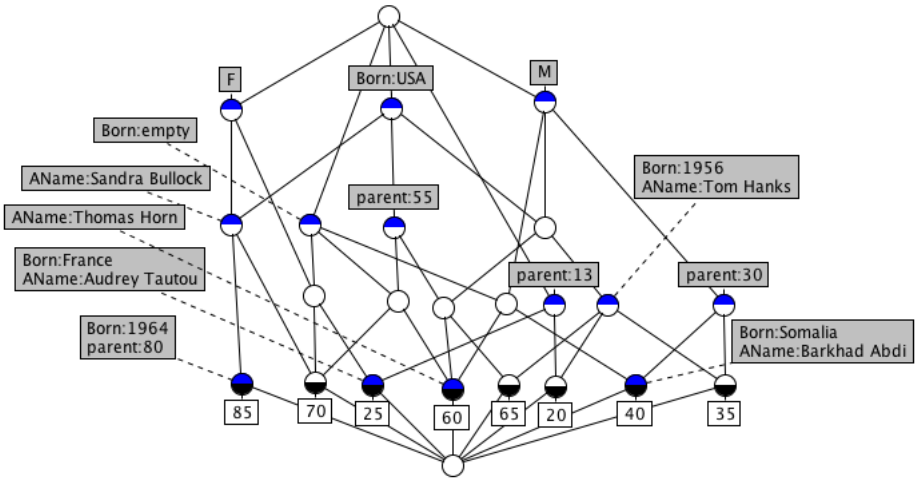

FIGURE 10. R_{actor}

Let $G_{\mathbb{K}} := \bigcup_{i \in I_1} S_i$ and $M_{\mathbb{K}} := \bigcup_{j \in I_2} S_j$. We can define a *triadic context* (for short tri-context) by $\mathbb{T}_{\mathbb{K}} := (G_{\mathbb{K}}, M_{\mathbb{K}}, P, Y_{\mathbb{K}})$ with $Y_{\mathbb{K}} := \{(g, m, p) \in G_{\mathbb{K}} \times M_{\mathbb{K}} \times P \mid (g, m) \in R_p\}$. The conceptual structure of $\mathbb{T}_{\mathbb{K}}$ can be seen as a natural triadic extension of the concept lattices $\mathfrak{B}(\mathbb{K}_p)$.

Definition 13. Given an XML database, the above construction gives us the *canonical translation* of the XML database as a formal tricontext.

Example 5. The triadic context generated by the *Movies* multicontext has as object set $G := \{1, 10, 50, 13, 30, 55, 80, 20, 25, 35, 40, 60, 65, 70, 85\}$, the attribute set is obtained by taking all nominally scaled attributes of the four many-valued contexts 1 - 4, and the condition set contains the labels of the four hierarchical levels from $\top$ to *Actor*.

Definition 14. For $\{i, j, k\} = \{1, 2, 3\}$ with $j < k$ and for $X \subseteq K_i$ and $Z \subseteq K_j \times K_k$, the $(-)^{(i)}$-derivation operators are defined by

$$X \mapsto X^{(i)} := \{(a_j, a_k) \in K_j \times K_k \mid (a_i, a_j, a_k) \in Y \text{ for all } a_i \in X\},$$

$$Z \mapsto Z^{(i)} := \{a_i \in K_i \mid (a_i, a_j, a_k) \in Y \text{ for all } (a_j, a_k) \in Z\}.$$

These derivation operators correspond to the derivation operators of the dyadic contexts defined by $\mathbb{K}^{(i)} := (K_i, K_j \times K_k, Y^{(i)})$, where

$$a_1 Y^{(1)}(a_2, a_3) \Leftrightarrow a_2 Y^{(2)}(a_1, a_3) \Leftrightarrow a_3 Y^{(3)}(a_1, a_3) \Leftrightarrow (a_1, a_2, a_3) \in Y.$$

Definition 15. For $\{i, j, k\} = \{1, 2, 3\}$ and for $X_i \subseteq K_i, X_j \subseteq K_j$ and $A_k \subseteq K_k$, the $(-)^{A_k}$-derivation operators are defined by

$$X_i \mapsto X_i^{A_k} := \{a_j \in K_j \mid (a_i, a_j, a_k) \in Y \text{ for all } (a_i, a_k) \in X_i \times A_k\},$$

$$X_j \mapsto X_j^{A_k} := \{a_i \in K_i \mid (a_i, a_j, a_k) \in Y \text{ for all } (a_j, a_k) \in X_j \times A_k\}.$$

These derivation operators correspond to the derivation operators of the dyadic contexts defined by $\mathbb{K}^{ij}_{A_k} := (K_i, K_j, Y^{ij}_{A_k})$ where

$$(a_i, a_j) \in Y^{ij}_{A_k} \Leftrightarrow (a_i, a_j, a_k) \in Y \text{ for all } a_k \in A_k.$$

Example 6. In the tricontext *Movies*, if g is an object, i.e., is a key of a node in the XML dataset about movies, then $i = 1, j = 2, k = 3$ and $g^{(1)}$ is the tree tuple having as root node the parent of g, while $g^{(K_3)}$ is the generalized tree tuple having g as a pivot element (see Definition 5).

This allows the algorithmic discovery of all generalized tree tuples with a given pivot element. These generalized tree tuples are playing an essential role in defining inter-relational functional dependencies as defined in [19]

If e is an element name, let A be the set of all nodes in the XML data set, having as label the element name e. Then $A^{(K_3)}$ is the tuple class of e (see Definition 6).

We have represented all important elements from an XML dataset using Triadic FCA. Discovering inter-relational functional dependencies can now be done using the algorithms developed for mining triadic implications.

Definition 16. ([5]) If $\mathbb{K} := (G, M, B, Y)$ is a tricontext, $R, S \subseteq M, C \subseteq B$, an expression of the form $R \overset{C}{\rightarrow} S$ is called *conditional attribute implication* and is read as R implies S under all conditions from C. A conditional attribute implication $R \overset{C}{\rightarrow} S$ *holds* in $\mathbb{K}$ if and only if the following is satisfied:

For each condition $c \in C$, it holds that if an object $g \in G$ has all the attributes in R then it also has all the attributes in S.

Definition 17. Let $\mathbb{K}$ be a the tricontext resulting from the canonical translation of an XML database. Let C_p be a tuple class. Then, the *formal tricontext of functional dependencies with respect to* C_p is defined as $\text{XMLFD}(\mathbb{K}) := (C_p \times C_p, M, P, Y)$, where M is the set of element names, P the set of nested tables, and $((g, h), e, p) \in Y$ if and only if the path values of g and h are equal with regard to path-value equality from Definition 4.

Proposition 1. The inter-relational functional dependencies of an XML database are exactly the conditional attribute implications in XMLFD($\mathbb{K}$).

5. Conclusion and Further Research

As far as described above, FCA proves to be a valuable tool for the conceptual design of XML data. XML data can be represented in hierarchical or flat form. In a recent work we give an FCA based approach for mining functional dependencies for flat XML data representation. In this paper we define a triadic FCA approach for a conceptual model of hierarchical XML data representation. The formal tricontext of functional dependencies with respect to a tuple class is given. This triadic approach is applicable in discovering inter-relational functional dependencies using algorithms developed for mining triadic implications.

As future work we propose to develop a software which will build the tricontext of an XML tree. The conditional attribute implications will give the functional dependencies from XML tree.

References

[1] M. Arenas, L. Libkin: *A normal form for XML documents.* TODS 29(1), pp. 195-232 (2004)

[2] S. Hartmann, S. Link: *More functional dependencies for XML.* In: Proc. ADBIS, pp. 355-369 (2003)

[3] S. Hartmann, S. Link, T. Trinh: *Solving the implication problem for XML functional dependencies with properties.* In: Logic, Language, Information and Computation, 17th International Workshop, WoLLIC, pp. 161-175 (2010)

[4] B. Ganter, R., Wille: *Formal Concept Analysis. Mathematical Foundations.* Springer, Berlin-Heidelberg-New York(1999)

[5] B. Ganter, S. Obiedkov, *Implications in Triadic Formal Contexts*, in ICCS 2004, LNAI 3127, pp. 186-195, Springer Verlag, 2004.

[6] J. Hereth: *Relational Scaling and Databases.* Proceedings of the 10th International Conference on Conceptual Structures: Integration and Interfaces LNCS 2393, Springer Verlag, pp. 62-76 (2002)

[7] K.T. Janosi-Rancz, V. Varga.: *XML Schema Refinement Through Formal Concept Analysis*, Studia Univ. Babes-Bolyai Cluj-Napoca, Informatica, vol. LVII, No. 3, pp. 49-64 (2012)

[8] K.T. Janosi-Rancz, V. Varga, T. Nagy: *Detecting XML Functional Dependencies through Formal Concept Analysis*, 14th East European Conference on Advances in Databases and Information Systems (ADBIS), Novi Sad, Serbia, LNCS 6295, pp. 595-598 (2010).

[9] K.T. Janosi-Rancz, V. Varga: *A Method for Mining Functional Dependecies in Relational Database Design Using FCA*, Studia Univ. Babeş-Bolyai, Informatica, Vol. LIII, Nr. 1 (2008), pp. 17-28.

[10] K.T. Janosi-Rancz, V. Varga, J. Puskas: *A Software Tool for Data Analysis Based on Formal Concept Analysis*, Studia Univ. Babeş-Bolyai, Informatica, Vol. LIII, Nr. 2 (2008), pp. 67-78.
[11] F. Lehmann, R. Wille: *A Triadic Approach to Formal Concept Analysis*, in: Ellis, G., Levinson, R., Rich, W., Sowa, J. F. (eds.), Conceptual Structures: Applications, Implementation and Theory, vol. 954 of Lecture Notes in Artificial Intelligence, Springer Verlag, (1995), pp. 32-43
[12] Gy. Szabó, A. Benczúr.: *Functional Dependencies on Extended Relations Defined by Regular Languages*, Annals of Mathematics and Artificial Intelligence, May 2013, pp. 1-39.
[13] V. Varga, K.T. Janosi-Rancz, C. Sacarea, K. Csioban: *XML Design: an FCA Point of View*, Proceedings of 2010 IEEE International Conference on Automation, Quality and Testing, Robotics, Theta 17th edition, Cluj Napoca, pp. 165-170 (2010)
[14] M. W. Vincent, J. Liu, C. Liu: *Strong functional dependencies and their application to normal forms in XML*, ACM TODS, 29(3), pp. 445-462 (2004)
[15] W3C. XML Schema, http://www.w3.org/XML/Schema (2014)
[16] J. Wang: *A comparative study of functional dependencies for XML*. In: APWeb, pp. 308-319 (2005)
[17] R. Wille: *Conceptual Structures of Multicontexts*. In Conceptual Structures: Knowledge Representation as Interlingua Lecture Notes in Computer Science Volume 1115, (1996), pp 23-39.
[18] S.A. Yevtushenko: *System of data analysis "Concept Explorer"*. (In Russian). Proceedings of the 7th national conference on Artificial Intelligence KII, Russia, pp. 127-134 (2000).
[19] C. Yu, H. V. Jagadish: *XML schema refinement through redundancy detection and normalization*, VLDB J. 17(2), pp. 203-223 (2008)

Babeş-Bolyai University, Cluj, Romania
E-mail address: csacarea@math.ubbcluj.ro

Babeş-Bolyai University, Cluj, Romania
E-mail address: ivarga@cs.ubbcluj.ro