Conceptual Graphs Based Modeling of Semi-Structured Data

Viorica Varga, Christian Săcărea, and Andrea Eva Molnar

Babes-Bolyai University, Cluj Napoca, 400081 str. Kogalniceanu 1, RO
andrea.molnar@math.ubbcluj.ro
ivarga@cs.ubbcluj.ro
csacarea@math.ubbcluj.ro

Abstract. Due to the fast growing of data in the digital world, not only in volume but also in its variety (structured, un-structured or hybrid), traditional RDBMS are complemented with a rich set of systems, known as NoSQL. One of the main categories of NoSQL databases are document stores which are specifically designed to handle semi-structured data, for instance XML documents. In this paper, we present a modeling method for semi-structured data based on Conceptual Graphs and exemplify the method on an XML document. The expressive power of Conceptual Graphs makes them particularly suitable for conceptual modeling of semi-structured data.

1 Introduction and Related Work

NoSQL data stores vary not only in their data but also in their query model. Document based NoSQL systems are based on the semi-structured data model, but most NoSQL data stores do not enforce any structural constraints on the data; they are usually referenced as schema-less data. On the other hand, for managing and retrieving data, its inherent structure proves to be significant. Not knowing the general structure of the data, makes tasks like application development or data analysis very difficult. Especially for modeling purposes, conceptual design of semi-structured data proves to be an important task. Several visualization methods have been developed over time in order to enhance understanding and to offer reasoning support for non-experts. For instance, Visual Query Systems (VQS) give a visual solution for non expert users which no longer have to understand query languages such as SQL or XQuery. A survey on visual query systems is given in [7] with the purpose of facilitating querying databases to non expert users. For instance, the modeling language of Conceptual Graphs (CG) can be used to model relational database design and querying [14]. In this research, we continue the ideas developed in [14] and describe how CG can be employed for conceptual modeling of semi-structured data.

Originally, CGs have been introduced by J. Sowa in [11] to model database interfaces and then further elaborated in [12]. Since then, CG have been used as a conceptual schema language, as well as a knowledge representation language with goal to provide a graphical representation for logic which is able to support

human reasoning. Among the plethora of interesting applications, we would like to mention a consistent, graphical interface for database interaction. CG have been successfully mathematically formalized for representing database structures in [3] and [4].

A unified CG approach to represent database schema - including relations between tables, and to model queries in databases has been described in [14]. XQuery, the standard language to query XML data, is gaining increasing popularity among computer scientists. A representation method of XQuery using CG which proves to be a good visual tool for even for non-experts has been discussed in [8], [9]. The structure of the XML data is given as a CG, helping the user to construct the query on the selected data. A graphical user interface BBQ (Blended Browsing and Querying) is proposed in [10] for browsing and querying XML data sources. BBQ is a query language for XML-based mediator systems (a simplification of XML-QL) which allows queries incorporating one or more of the sources. In BBQ XML elements and attributes are shown in a directory-like tree and the users specify possible conditions and relationships (as joins) among elements.

In this paper, we continue our previous research on modeling various databases using FCA and CG [14], [13] and we propose a CG based modeling of semi-structured data. This conceptual model can be useful both for document-oriented database design and for XML data modeling. We exemplify the developed methods on a toy XML data set. This research come along with a software tool for XML data design using CG. This tool has been developed targeting a wide range of users, from researchers wanting to validate their work using the CG grounded conceptual model developed in this paper to non-experts who would like to rely on the expressive power of CG as a visual interface which gives the user the possibility to formulate a specific hierarchical view without any knowledge of the detailed structure and the content of the database constructs.

2 Graphic Representation of Semi-Structured Data Model by Conceptual Graphs

In the following, we propose a CG grounded representation of semi-structured data model.The method is similar to the representation with E/R diagrams and consists of the following three main steps:

Step 1: Identify all complex objects. Every object will be modeled as a CG **concept** and graphically represented as a rectangle.

Step 2: Define the relationships between the objects. The relationship between two or more **concepts** will be represented by means of a **relation** and graphically represented as an oval. Similarly to the approach in [11], a directed arc from the first concept node to the relation node and another arc from the relation to the second concept node will represent this relationship. If a relation has only one element, the arrowhead is omitted. If a relation has more than two elements, the arrowheads are replaced by integers $1, \ldots, n$. The **direction of arcs** are **from the root to leaf** levels. According to the type of the relationship between the objects, a relation can be:

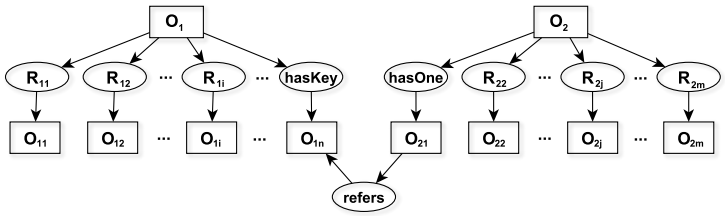


Fig. 1: CG describing a connection between the related objects O_1 and O_2.

- `hasOne`: models a one-to-one relationship between two objects.
- `hasMore`: illustrates a one-to-many relationship.
- `hasKey`: specifies the key object of a given object;
- `refers`: represents a reference between two or more object. It can be considered as a graphical representation of the foreign key constraint: the directed arc points from the foreign key to the key element.
- `isOptional`: designs the case when the minimum number of occurrences of an object is zero. If this relation is omitted, the default value of the number of occurrences (of the corresponding object) is one.

The structure of the semi-structured data gives a natural representation of the *one-to-many* relationships. The most common representation is that the child object is embedded into the parent object. The advantage of this modeling is that we do not have to perform a separate query in order to get the details of the embedded data, so the execution time of a query will decrease. The disadvantage is that we cannot access the embedded details as independent entities.

By contrast, the modeling of *many-to-many* relationship is particularly challenging in XML, but there exist a couple of possible solutions to model it: nested in one of the parents and one foreign key referencing to the other parent, or as in relational databases, at root level with two foreign keys. The most common representations use the first possibility. Figure 1 presents the way of designing the relationship between two related objects. Let O_1 and O_2 be two complex objects. We assume that O_{1n} is the key object (represented by the relation `hasKey`) of O_1. Object O_{21} (as foreign key) of O_2 references O_{1n}, represented by the relation `refers`.

Step 3: Identify all descendant objects of the complex objects, defined at *Step 1.* Remark that, in the case of Document Store Databases there is no possibility to define the type of the elementary attributes. Hence, this step will be presented in detail in the next paragraph, concerning to the XML Data Model.

3 CG Based Representation of XML Data Structure

Similar to the above concept, we present in the following how XML data structure can be represented using CGs (referenced as XML Schema CG in the following).

Step 1. Identify all complex elements. The **concept type** can represent: an *element* (`E`); an *attribute* (`A`); a *root element* (`R`); the *type of the data* (`T`); an *enumeration constraint* (`N`). At the same time, the **concept referent** can be:

- the *name of* the corresponding *node* (element or attribute);
- the *data type of a simple element* or *an attribute*;
- the *set of acceptable values* of the corresponding *element*, in the case of enumeration. In this particular case, the concept name will be composed by the predefined values of the enumeration, separated by (|).

Step 2. Define the relationship between the nodes. As we defined earlier, the relationship between two or more **concepts** will be illustrated by a CG **relation**. The relations presented in the previous section are adaptable also to the XML Schema documents. Hence, we analyze only the additional relations:

- `refers`: as in the general case, it represents a linkage between two or more complex elements. In the case of XML Schema, this relation can express the ID/IDREF couples with the help of the *hasKey* relation.
- `hasType`: defines the data type of a simple element or an attribute. If the *hasKey* relation is set, this relation can be omitted in the case of the corresponding element or attribute.
- `hasChoice` and `isPossibility`: specify the child elements of a choice compositor. In the XML Schema representation, the definition of a choice element allows only one of the elements contained in the declaration to be present within the containing element.
- `isEnum`: represents the enumeration constraint, which limits the content of an element to a set of acceptable values.

Step 3. Identify all child elements or attributes of the complex elements, defined at *Step 1*, give the data type for each one and specify also the potential restrictions of the elements or attributes.

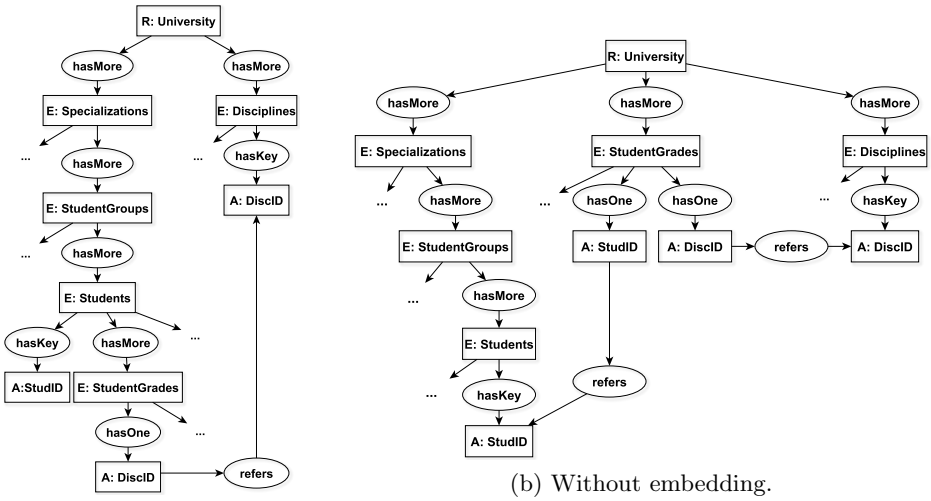
4 An example to illustrate the representation of XML Schema with Conceptual Graphs

In this section, we present the previously detailed methods on a graphical representation of XML Schema using CG through a detailed example. Let us consider the structure of data in the case of a university. The data is stored in XML documents, but we want a graphical representation which illustrates more profoundly the hierarchical structure of our data.[1]

Step 1. We define a concept, called `University`, which will represent the root element of the XML Schema. The `University` element contains some different child elements, e.g. `Specializations` and `Disciplines`, which are all complex elements. So, let us construct their descendants and let us define the relationships among them.

Step 2. Students of a specialization are organized in groups. Therefore, we have three complex elements in the graph as concepts according to the following hierarchy: `Specialization` becomes the root element, while `StudentGroups`

[1] Due to obvious space reasons, we have not included the complete XML schema (referred as *UniversityXSD* in the following) and the corresponding XML document, but they can be consulted at http://www.cs.ubbcluj.ro/~fca/semistructured-data/.



(a) With embedding.

(b) Without embedding.

Fig. 2: Two possible solutions for designing the many-to-many relationships between the `Students` and `Disciplines` elements.

and `Students` become child elements of `Specialization` and `StudentGroups`, respectively; this parent-child relationship will be represented by `hasMore` relations. Now, let us construct the elements `Disciplines` and `StudentGrades`. Notice that students have grades for more disciplines. Hence, there exists a many-to-many relationship between `Students` and `Disciplines`, which is represented by the `StudentGrades` element. Figures 2a and 2b show two main possible hierarchies, which can be formulated for this many-to-many relationship. Figure 2a designs the classic solution: the `StudentGrades` element is nested into the `Students` element. Hence, the relationship between these two elements is defined. The linkage between `Disciplines` and `StudentGrades` elements is represented by the dyadic relation `refers`, which links the `DiscID` (as "foreign key") from `StudentGrades` with the *key* element of `Disciplines` element, i.e. the key attribute `DiscID`. Remark that, in the *UniversityXSD* XML Schema the many-to-many relationship is represented in this standard way. Another possibility to illustrate this type of relationship would be embedding the `StudentGrades` element into the `Disciplines` element. This representation is identical to the above solution, so we omit its graphical representation.

Figure 2b shows the third possible method for designing the many-to-many relationship. The `StudentGrades` element is selected as the child of the root element. In this situation, the `StudentGrades` element links the `Students` element with `Disciplines` element through the reference elements `StudID` and `DiscID`, respectively. Remember that, in the case of ID/IDREF couples the use of `hasType` relation and the data type concept, respectively, could be omitted (see the attributes `StudID` and `DiscID`). Notice also that the name of the connecting nodes can be different in the parent and child elements.

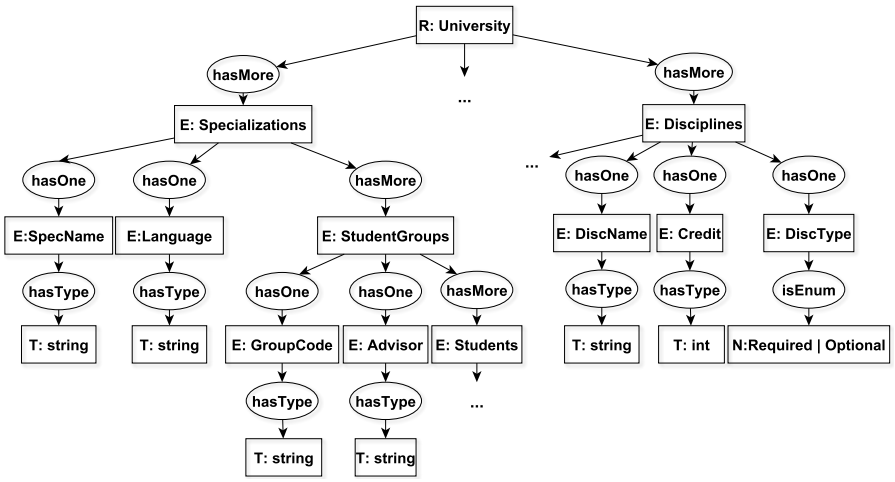


Fig. 3: The part of the `University` XML Schema CG emphasizing the hierarchy between the `Specializations`, `StudentGroups` and `Students` elements and showing the structure of the `Specializations`, `StudentGroups` and `Disciplines` elements.

Step 3. A detailed design for the complex elements is illustrated in Figures 3 and 4. Let us analize the `Students` element (see Figure 4). We have two possibilities to define the `StudentName` element, using the `hasChoise` relation: we can give the full name of the student as one string or we can give the parts of the name separately. According to the definition of the choice element, one of the possibilities must be selected. Element `BirthPlace` is selected as optional. Notice that, a `hasMore` relation can connect a complex element with a simple element, e. g. the relation between `Students` and `PhoneNumber` elements. In the case of the `Disciplines` element it is defined an enumeration constraint by the `DiscType` element, see Figure 3: we can choose between *Required* and *Optional*.

5 The Visual Interface

We have implemented a tool, named *XSD Builder*, which provides a user-friendly interface in form of CGs and gives the user the possibility to build the graphs of personalized XML Scheme. A snippet from the structure of the detailed XML Schema example, presented in the previous section (*UniversityXSD*) visualized by *XSD Builder* can be seen in Figure 5. The user can select one concept or relation from the toolbox, by dragging it to the schema pane. Then, the user can give the name of the selected object, by writing it into the rectangle or the oval. It is also easy to specify the relationship between the concepts and relations: the user can draw a line between them. The tool gives the possibility to generate XML Scheme from the CG designed in it with a single click. Also, given an XSD file, the application will visualize the corresponding CG.

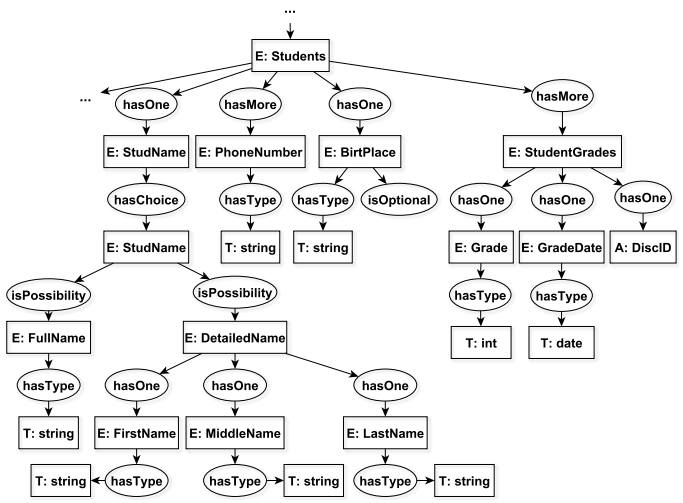


Fig. 4: The snippet of the `University` XML Schema CG illustrating the structure of the `Students` and `StudentGrades` elements.

6 Conclusions and Future Work

The goal of CGs is to provide a graphical representation for logic which is able to support human reasoning. This article proposes an application which provides a graphical interface for Semi-Structured Data Design in form of CGs. We implemented a software for XML Data Design using CG. Our future goal is to extend the list of available tools in the case of XML schema representation (e.g. involving the group element). The presented representation using CG can be implemented as a tool for MongoDB data structure too.

References

1. Braga, D., Campi, A., Ceri, S.: XQBE (XQuery By Example): A visual interface to the standard XML query language. ACM Trans. Database Syst. 30(2), 398–443 (Jun 2005)
2. Cattell, R.: Scalable SQL and NoSQL data stores. SIGMOD Rec. 39(4), 12–27 (May 2011)
3. Dau, F.: The Logic System of Concept Graphs with Negation And Its Relationship to Predicate Logic, Lecture Notes on Computer Science, vol. 2892. Springer Berlin, Heidelberg (2003)
4. Dau, F., Hereth, J.C.: Nested concept graphs: Mathematical foundations and applications for databases. In: Using Conceptual Structures. Contributions to ICCS, pp. 125–139. Shaker Verlag, Aachen (2003)
5. Elmasri, R., Li, Q., Fu, J., Wu, Y., Hojabri, B., Ande, S.: Conceptual modeling for customized XML schemas. Data and Knowledge Engineering 54(1), 57–76 (7 2005)
6. Guy, M.W., Moulin, B., eds., J.S. (eds.): Conceptual Graphs for Knowledge Representation, Lecture Notes in AI, vol. 699. Springer (1993)

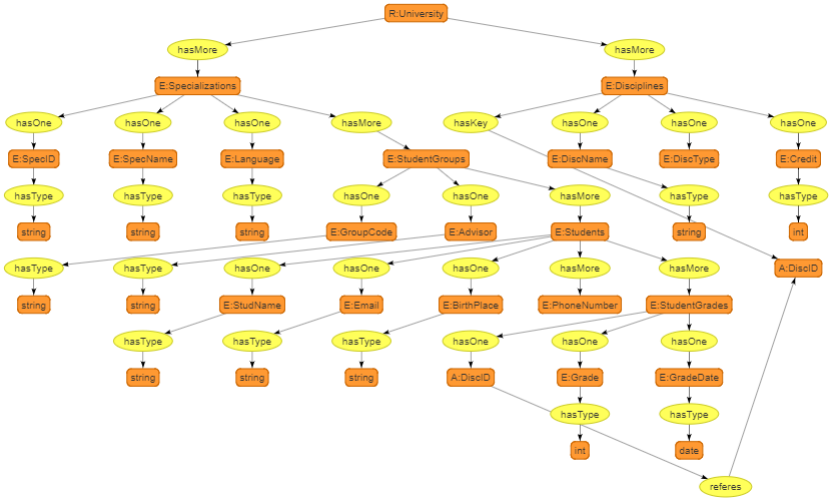


Fig. 5: Application screen-shot for UnivesityXSD design

7. Lloret-Gazo, J.: A survey on visual query systems in the web era. In: Database and Expert Systems Applications: 27th International Conference, DEXA 2016, Porto, Portugal, September 5-8, 2016, Proceedings, Part II, pp. 343–351. Springer International Publishing (2016)
8. Molnar, A., Varga, V., Sacarea, C.: Conceptual graph driven modeling and querying methods for RDMBS and XML databases. In: Proceedings of the 13th International Conference on Intelligent Computer Communication and Processing (ICCP 2017), pp. 55–62. IEEE (2017)
9. Molnar, A., Varga, V., Sacarea, C.: Conceptual graphs based modeling and querying of XML data. In: Proceedings of the 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), pp. 23–28. IEEE (2017)
10. Munroe, K.D., Papakonstantinou, Y.: BBQ: A visual interface for integrated browsing and querying of XML. In: Proceedings of the Fifth Working Conference on Visual Database Systems: Advances in Visual Information Management. pp. 277–296. VDB 5, Kluwer, B.V., Deventer, The Netherlands, The Netherlands (2000)
11. Sowa, J.F.: Conceptual graphs for a database interface. IBM Journal of Research and Development 20(4), 336–357 (1976)
12. Sowa, J.F.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley (1984)
13. Varga, V., Janosi-Rancz, K.T., Kalman, B.: Conceptual design of document NoSQL database with Formal Concept Analysis. Acta Polytechnica Hungarica 13(2), 229–248 (2016)
14. Varga, V., Sacarea, C., Takacs, A.: Conceptual graphs based representation and querying of databases. In: Proceedings of the International Conference on Automation, Quality and Testing, Robotics (AQTR) - Volume 03, pp. 1–6. IEEE (2010)