On a Conceptual Data Model with Orientation to Data Integration

Manuk G. Manukyan

Yerevan State University, Yerevan 0025, Armenia,
mgm@ysu.am

Abstract. In this paper a conceptual data model oriented to data integration is proposed. Formal definition of the considered conceptual data model is provided. To define the behavior of entities of the conceptual level, an algebra over such entities was developed. Formalization issues of data integration concept are discussed. Principles of mapping of source data models basic constructions into conceptual data model are considered. Mapping from data sources into conceptual schema is defined as an algebraic program.

Keywords: Data Integration · Data Warehouse · Mediator · Data Cube · Ontology · Reasoning Rules · OPENMath.

1 Introduction

For more than forty years, the problems of data integration have been the subject of research in the field of databases. Analysis of existing approaches to data integration can be found in the works [10, 14, 18]. Basically, these works are devoted to the problems of integrating homogeneous data sources. Typically, an extended relational or object data model was used as the target data model. Construction of mapping from arbitrary source data models into a target data model assumes using an extensible data model as the target one. In this connection, using the XML data model as the target one is preferred as this model is some compromise between conventional and semistructured data models because in contrast to:

- semistructured data model, the concept of database schema in the sense of conventional data models is supported;
- conventional data model hard schemas, there is possibility to define more flexible database schemas.

Since the end of the last century an approach to ontology-based data integration has appeared. Such approach allows to provide ontology-based data access. The current interest to data integration basically is connected with challenges of big data processing. Analysis and management of big data assumes the consideration of structured as well as semistructured and unstructured data sources. In this context, it becomes important to combine conventional and non-conventional approaches to data integration. For instance, machine learning

techniques can be applied to unstructured and semistructured data sources for obtaining structured data from them.

In the frame of this paper the issues of constructing an extensible conceptual data model with orientation to data integration are considered. To model conceptual entities an extensible formalism (OPENMath) is used [7]. Conceptual schema is defined as a collection of OPENMath objects. The property of extensibility of the OPENMath allows to construct a mapping from arbitrary data sources into conceptual schema. A distinguishing feature of the proposed approach to data integration is also the possibility of creating and accessing the integrated databases by means of conceptual level entities. Thus, it is possible to define an integrated database in concepts of a subject domain using the rich apparatus of computational mathematics, which is not unimportant when processing big data.

The paper is organized as follows: An extensible formalism to model conceptual entities is considered briefly in Section 2. Formal definition of an extensible conceptual data model is proposed in Section 3. Data integration concept formalization is offered in Section 4. Some issues of mapping from data sources into extensible conceptual data model are discussed in Section 5. Related work is presented in Section 6. The conclusion is provided in Section 7.

2 Formal Bases

This section provides a brief analysis of the OPENMath concept.

2.1 OPENMath Objects

OPENMath is an extensible formalism and is oriented to represent semantic information on the mathematical objects. Formally, an OpenMath object is a labeled tree whose leaves are basic OpenMath objects. Examples of basic OPENMath objects are: Integer (integers in the mathematical sense, with no predefined range), Symbol (a mathematical concept) and Variable (are meant to denote parameters). The compound objects are defined in terms of *binding* and *application* of the λ-calculus [11]. The following recursive rules for constructing compound OPENMath objects are proposed:

- Basic OPENMath objects are OPENMath objects.
- If $A_1, A_2, ..., A_n$ $(n \geq 1)$ are OPENMath objects, then *application*$(A_1,\ A_2, ..., A_n)$ is an OPENMath *application object*.
- If $S_1, S_2, ..., S_n$ are OPENMath symbols, and $A, A_1, A_2, ..., A_n$ $(n \geq 1)$ are OPENMath objects, then *attribution*$(A,\ S_1\ A_1,\ S_2\ A_2,\ ...,\ S_n\ A_n)$ is an OPENMath *attribution object* and A is the object *stripped* of *attributions*.
- If B and C are OPENMath objects, and $v_1, v_2, ..., v_n$ $(n \geq 0)$ are OPENMath variables or attributed variables, then *binding*$(B, v_1, v_2, ..., v_n,\ C)$ is an OPENMath *binding object*.

OpenMath objects have the expressive power to cover all areas of computational mathematics.

2.2 Types in OPENMath

A type system is built from basic types which are predefined as typed OPENMath objects (for example, *integer*, *string*, *boolean*, etc.) and the following rules by means of which typed OPENMath objects are constructed:

Attribution rule. If v is an OPENMath variable and t is a typed OPENMath object, then $attribution(v, type\ t)$ is typed OPENMath object. It denotes a variable with type t. The following is an example of typed OPENMath application object for trigonometric function $\sin v$:

$application(sin, attribution(v, type\ real))$

Abstraction rule. If v is an OPENMath variable and t, A are typed OPENMath objects, then $binding(lambda, attribution(v, type\ t), A)$ is typed OPENMath object and denotes the function that assigns to the variable v of type t the object A.

Application rule. If F and A are typed OPENMath objects, then $application(F, A)$ is typed OPENMath object.

2.3 Semantic Level

OPENMath is implemented as an XML application. Its syntax is defined by syntactical rules of XML, its grammar is partially defined by its own DTD (Document Type Definition). Only syntactical validity of the OPENMath object's representation can be provided on the DTD level. To check semantics, in addition to general rules inherited by XML applications, the considered application defines new syntactical rules. This is achieved by means of introduction of *signature files* concept (semantical constraints), in which these rules are defined. Signature files contain the signatures of basic concepts defined in some content dictionary and are used to check the semantic validity of their representations. Content dictionaries are used to assign formal and informal semantics to all symbols (concepts) used in the OPENMath objects. A content dictionary is a collection of related symbols, encoded in XML format and fixing the "meaning" of concepts independently of the application.

3 A Conceptual Data Model

To construct a conceptual data model with orientation to data integration, we are based on the concept of hierarchical relation as an entity of the conceptual level. Below we introduce the definitions of hierarchical relation schema and hierarchical relation. These definitions can be considered as strengthening of definitions of the relation schema and relation of the relational databases. Namely, unlike the relational databases, we allow use of semistructured data model constructions during subject domain modeling.

3.1 Formalization of Hierarchical Relations

Definition 1 *A hierarchical relation schema X is an attribution object and is interpreted by a finite set of attribution objects $\{A_1, A_2, \ldots, A_n\}$. Corresponding to each attribution object A_i is a set D_i (a finite, non-empty set), $1 \leq i \leq n$, called the domain of A_i.*

In the frame of the proposed extensible conceptual data model, the following OPENMath representation of hierarchical relation schema X is accepted:

$$attribution(X, type\ A, S_1\ A_1, S_2\ A_2, \ldots, S_k\ A_k), k \geq 0$$

Here $S_1, S_2, ..., S_k$ are OPENMath symbols, X is OPENMath variable, and $A, A_1, A_2, ..., A_k$ are OPENMath objects. In this representation, X is the name of the atttribution object, A represents the type of the attribution object (basic or composite), and by means of $\langle S_i\ A_i \rangle$ pair defines one property of the modeled object ($1 \leq i \leq k$). To construct composite types, we introduced the following type constructors: *sequence*, *choice* and *all*. The conceptual entities that are created using these type constructors have analogous semantics as the *sequence*, *choice* and *all* elements of XML Schema language. The arguments of these functions are typed attribution objects. We distinguish two types of typed attribution object: basic and composite. A composite attribution object is defined by a type constructor. The following is an example of a composite attribution object:

$$attribution(X, type\ application(sequence, A_1, A_2, \ldots, A_n))$$

where A_i ($1 \leq i \leq n$) is a basic or composite attribution object. In case of a basic attribution object the value of *type* symbol is a basic type (for example, *integer*, *string*, etc.). Below an XML Schema element definition and its equivalent representation in the frame of the conceptual data model (see Fig. 1.) is considered:

$< xs : element\ name =$ "Book"$>$
  $< xs : complexType >$
    $< xs : sequence >$
      $< xs : element\ name =$ "Author" $type =$ "$xs : string$"$/ >$
      $< xs : element\ name =$ "Title" $type =$ "$xs : string$"$/ >$
      $< xs : element\ name =$ "Publisher" $type =$ "$xs : string$"$/ >$
    $< /xs : sequence >$
  $< /xs : complexType >$
$< /xs : element >$

Definition 2 *Let $D = D_1 \cup D_2 \cup ... \cup D_n$. A hierarchical relation x on hierarchical relation schema X is a finite set of mappings $\{t_1, t_2, ..., t_k\}$ from X to D with the restriction that for each mapping $t \in x$, $t[A_i]$ must be in D_i, $1 \leq i \leq n$. The mappings are called hierarchical tuples or simply tuples.*

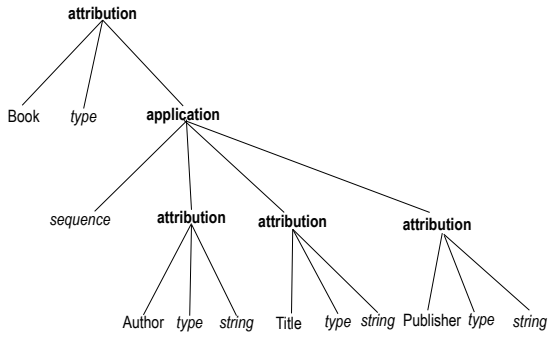

Fig. 1. From XML Schema to labeled tree: Transformation example.

A hierarchical relation is an instance of a hierarchical relation schema. The following are examples of instances of the above considered hierarchical relation schema:

<Book>
<Author> David Maier < /Author>
<Title> The Theory of Reltional Databases < /Title>
<Publisher> Computer Science Press < /Publisher>
< /Book>

or

{
"Book": {
"Author": "David Maier",
"Title": "The Theory of Relational Databases",
"Publisher": "Computer Science Press"
}
}

Definition 3 *A key of a hierarchical relation x on hierarchical relation schema X is a minimal subset K of X such that for any distinct tuples $t_1, t_2 \in x$, $t_1[K] \neq t_2[K]$.*

We added the symbols X, x, *tuple* and *key* to conceptual data model to formalize the concepts of hierarchical relation schema, hierarchical relation, tuple and key correspondigly. A database schema S is a finite set of schemas of the hierarchical relations. A database D on database schema S is a collection of the hierarchical

relation $\{x_1, x_2,..., x_n\}$ such that for each schema of the hierarchical relation schema $s \in S$ there is a hierarchical relation $x \in D$ such that x is a hierarchical relation with schema s that satisfies every constraint defined in s. We introduce a symbol d to conceptual data model to denote the set of all hierarchical relations expressible in the frame of this model.

3.2 Algebra of Hierarchical Relations

In fact, data model defines operations over data. In our case, a unit for data manipulation is a hierarchical relation (a set). Below the following operations over hierarchical relations are proposed:

To support the n-ary union of sets, we introduced the symbol *union*. This symbol is used to denote associative/commutative union operation of sets:

$$union : x \times x \to d$$

To support the n-ary join of sets, we introduced the symbol *join*. This symbol is used to denote associative/commutative join operation of sets:

$$join : x \times x \to d$$

The symbol *minus* is used to denote the difference of sets:

$$minus : x \times x \to d$$

To support a filtering operation, we introduced the symbol σ. This symbol is used to denote a select operation on the set:

$$\sigma : \{x \to \{p : \{tuple\} \to boolean\}\} \to d$$

To support a projection operation, we introduced the symbol π. This symbol is used to denote a unary operation on the set:

$$\pi : x[name^*] \to d$$

Here *name* denotes the name of an attribution object and is defined as follows:

$$name : \{Attribution\} \to string$$

For integrating data, aggregating functions play a significant role. We introduced the *min*, *max*, *count*, *sum* and *avg* symbols to support the corresponding aggregate functions of the relational algebra. Let $f \in \{avg, sum, count, max, min\}$, then

$$f : x[name] \to numericalvalue$$

Often, we need to consider the tuples of a hierarchical relation in groups. For this purpose, we introduced a grouping symbol γ. This symbol is used to denote a unary operation on the set:

$$\gamma : x[name^* (, f : (tuple[name^*])^* \to numericalvalue)^*] \to d$$

3.3 Conceptual Schema

The conceptual schema is an instance of conceptual data model and is intended for formal knowledge representation in the form of a set of concepts of some subject domain and relations between them. Such representations are used for reasoning about entities of the subject domains, as well as for the domains description. Conceptual schema is defined as a collection of typed attribution objects. Our concept to data integration assumes constructing mapping from arbitrary data sources into conceptual schema. Mapping generation from data sources into conceptual schema assumes generating queries over data sources and transforming these data into set of hierarchical relations. A distinguishing feature of the conceptual level is its stratification of the local and global levels to model the hierarchical relations. On the local level a homogeneous representation of data sources is provided. The global level is intended to support various data integration technologies, such as data warehouses, mediators and etc. The entities of the global conceptual level are defined by algebraic programs. To support entities of the conceptual level we use mechanisms of content dictionaries and signature files of the OPENMath (for more details, see Appendix A and B).

4 Data Integration Concept Formalization

Our approach to data integration assumes formalizing the mediator, data warehouse and data cube concepts in the frame of proposed conceptual data model. Formalization results of these concepts are so-called reasoning rules of conceptual level to support data integration concept. These rules are interpreted by means of algebraic programs. Supporting the reasoning rules assumes developing new content dictionaries to assign informal and formal semantics of data integration basic concepts (for instance, hierarchical relation, hierarchical relation schema, algebraic operations, etc.). Formalization of basic concepts is achieved by applying the concept of OPENMath content dictionaries. Also we should formalize signatures of these basic concepts for checking the semantic validity of its representations. In this case, we will be using the OPENMath signature files concept to formalize signatures of basic concepts.

4.1 Mathematical Model

The considered research object is the data integration concept. The proposed formalization of this concept will be the mathematical basis for constructing the reasoning rules.

Mediator rule. Let *sch* and *wrapper* correspondingly denote the set of all mediator schemas and the set of all subsets of the wrappers which are defined on source data schemas to support the mediator concept, and let *med* denote the set of all mediators, then

$$med \subseteq sch \times wrapper$$

Warehouse rule. Let *wsch* and *extractor* correspondingly denote the set of all data warehouse schemas and the set of all subsets of the extractors which are defined on source data schemas to support the data warehouse concept, and let *whse* denote the set of all data warehouses, then

$$wshe \subseteq wsch \times extractor$$

Cube rule. Our approach to create data warehouses is mainly oriented to support data cubes. In typical OLAP applications, there is some collection of data called the *fact table* which represents events or objects of interest [9]. Usually, fact table contains several attributes representing dimensions, and one or more dependent attributes that represent properties for the point as a whole. The concept of the data cube assumes generation of the power set (set of all subset) of the aggregation attributes. For some dimensions, there are many degrees of granularity that could be chosen for a grouping on that dimension. When the number of choices for grouping along each dimension grows, it becomes non-effective to store the results of aggregating based on all the subsets of groupings. Thus, it becomes reasonable to introduce materialized views. The materialized view is interpreted by OPENMath *application* concept.

Let *ssch*, *sgty* and *view* correspondingly denote the set of all fact table schemas which are defined on source data schemas, the set of all granularities and the set of all materialized views to support the data cube concept, and let *cube* denote the set of all data cubes in this context, then

$$cube \subseteq ssch \times sgty \times view$$

Let *source* denote the set of all hierarchical relation schemas and let *dir* denote the set of all data integration reasoning rules, then

$$dir \subseteq source \times (med \cup whse \cup cube)$$

4.2 Reasoning Rules

The conceptual data model should be extensible for providing reversible mapping from arbitrary source data models into conceptual data model. One of the reasons for choosing OPENMath as a formalism to support the concept of data integration is its extensibility. The extension of the conceptual data model assumes introducing new concept(s) in the frame of this model. Thus, the extension of the conceptual data model leads to defining new symbols to support data integration concept on the conceptual level. For applying a *symbol* on the conceptual level, the following rule is proposed:

Concept ← *symbol* OPENMath object

To support entities of local conceptual level, a *source* symbol to conceptual data model is introduced. The following construction to define these entities is proposed:

attribution(Local, *source* S_1, *source* S_2,..., *source* S_n)

It is assumed that there are n source data ($n \geq 1$). The value of *source* symbol is a set of schemas of source data and each element of this set is defined as an application object:

application(*list*, $A_1, A_2, ..., A_k$),

where A_i is a typed attribution object ($1 \leq i \leq k$).

To support entities of global conceptual level the following reasoning rules are considered.

Warehouse rule. For supporting this rule a *whse* symbol to conceptual data model is introduced. The following construction to define this rule is proposed:

attribution(Name, *whse* Algebraic Program)

The value of *whse* symbol is an algebraic program (an application object) by means of which a mapping from data sources into data warehouse is defined.

Mediator rule. The following construction to define this rule is proposed:

attribution(Name, *med* Algebraic Program)

The proposed construct to model mediator rule is interpreted analogously as in the case of data warehouse.

Cube rule. For formalizing the concept of materialized view we introduced to conceptual data model the following symbols: *cube*, *dim*, *granularity* and *partition*. A dimensional table schema is the value of a *dim* symbol and also is interpreted by a typed attribution object. The materialized view is generated by means of an algebraic program. To support hierarchical dimensions, we introduced *granularity* and *partition* symbols. Hierachical dimension is defined by an attribution object and is the value of a *granularity* symbol. The value of *partition* symbol is division units of hierarchical dimension. The following construction to define a fact table is considered:

attribution(Fact, *type application*(*sequence*, A_1, A_2,...,A_n), *dim* D_1,

dim D_2,..., *dim* D_m, *granularity* G_1, *granularity* G_2,..., *granularity* G_k),

where A_j is a basic attribution object ($1 \leq j \leq n$), $G_i (1 \leq i \leq k)$

is an attribution object and is defined as follows:

attribution(Name, *partition application*(*list*, $P_1, P_2, ..., P_l$))

The following is a granularity concept example:

attribution(*Date*, *partition*

application(*list*, $Days, Months, ...,$ $Quarters, Years$))

The following construction to define the cube rule is proposed:

attribution(Name, *cube* Algebraic Program)

5 From Source Data Models to Conceptual Data Model

This section discusses the rules for mapping basic concepts of structured, semistructured, and also unstructured data models into conceptual data model. If necessary, the conceptual model is expanded by adding new concepts (symbols) to it.

From aggregate models to conceptual data model. According to the aggregate model, an aggregate is treated as data atomic unit and is presented as a collection of related objects. In the frame of conceptual data model an aggregate is modeled by means of the following construction:

attribution(AGName, *type application*(typeConstructor, $A_1, A_2, ..., A_m$)),

where A_i $(1 \leq i \leq m)$ is a basic or composite attribution object.

From relational data model to conceptual data model. Let $R = \{A_1, A_2, ..., A_n\}$ is a relational schema. To model the considered schema the following construction is used:

attribution(R, *type application*(*sequence*, $A_1, A_2, ..., A_n$)),

where $A_i (1 \leq i \leq n)$ is a basic attribution object.

From graph data model to conceptual data model. Graph data model allows to present entities and relationships between these entities. Entities are also known as nodes, and relations are known as edges. To model a node the following construction is proposed:

attribution(NName, *type application*(*set*, $P_1, P_2, ..., P_m$),

edge application(*set*, E_1, E_2, ..., E_n)), where $P_i(1 \leq i \leq m)$ is a basic attribution object (graph node property), and $E_j (1 \leq j \leq n)$ is a composite attribution object (graph node outgoing edge).

It is assumed that node has m properties, from node is outgoing n type of edges, and each type of edge is connected with k_j nodes.

From XML data model to conceptual data model. Basic concept of XML data model which is used to model subject domain is XML-element. The following constructions are used to model XML-element:

attribution(E, *type* basic type, *attribute application*(*set*, $A_1, A_2, ..., A_k$)),

or *attribution*(E, *type application*(typeConstructor, $E_1, E_2, ..., E_m$),

attribute application(*set*, $A_1, A_2, ..., A_k$)),

where A_i $(1 \leq i \leq k)$ is a basic attribution object (XML-attribute), and E_j $(1 \leq j \leq m)$ is a basic or composite attribution object (XML-element). We introduce a symbol *attribute* to model the concept of XML-attribute.

From object data model to conceptual data model. Base concepts of object data model are attributes, relationships and methods. The following construction is used to model the attribute concept:

attribution(AName, *type* basic type), or

attribution(AName, *type application*(typeConstructor, collectionType)

We are using the following type constructors *set*, *bag*, *list*, *array*, *dictionary*, and *structure* to support concepts of the attribute and relationship. To model the relationship concept the following constructions are used:

attribution(RName, *relationship application*(typeConstructor,

className) [*inverse* RName]), or

attribution(RName, *relationship* className [*inverse* RName])

To support the relationship concept, in addition to the considered type constructors, we also introduced the *relationship* and *inverse* symbols, which have obvious semantics.

The following construction is proposed to model the method concept:

attribution(FName, *arg application*(*list*, A_1, A_2,...,A_n), *return* Type),

where the value of introduced symbol *arg* is a list of method arguments, A_i ($1 \leq i \leq n$) is a basic or composite attribution object (method argument), and the value of the introduced symbol *return* is type of the value returned by the method.

6 Related Work

One of the first works in the area of justifiable data models mapping for heterogeneous databases integration is [12]. In this paper, the concept of reversible mapping of an arbitrary source data model into a target one (canonical model) were proposed. In particular, the following [13, 15] papers are devoted to problems of the heterogeneous database integration based on this concept.

A significant contribution to the theory and practice of data integration was made by the research group of M. Lenzerini (for instance, see [1–5]). Their investigations were carried out in the frame of the traditional approach of the data integration as well as in the frame of the paradigm of ontology-based data access and integration. In these investigations only relational data as source data are considered. To define ontology as well as mapping between ontology and data sources, the description logic is used. Finally, in the frame of these investigations, a formal approach to data quality is proposed. Namely, one of the most important dimensions (consistency) of data quality is considered. The following papers [19, 20] can be considered as some development of the works of M. Lenzerini group, in which as a query language on the ontology level SPARQL is considered. A SPARQL-program is translated into efficient (federated) SQL-program over data sources based on the proposed optimisation techniques.

In [17] a system to automatically generate direct mappings between relational databases and given target ontologies is developed. The considered system is based on an intermediate internal graph representation that allows the representation of both factual knowledge and heuristically observed patterns from the input. An approach to ontology-based data integration is proposed in [16], which also allows to generate a mapping from arbitrary data sources into ontology.

In [8] an analysis to use machine learning techniques to solve different problems to integrate unstructured and semistructured data is provided, namely, the use of machine learning techniques for entity resolution, data fusion, data extraction and schema alignment. Thus, the use of machine learning techniques allows to automate parts of different integration problems.

Finally, about our approach to data integration: In the frame of this approach an extensible conceptual data model with orientation to data integration is proposed. It is important that the proposed approach allows to generate a mapping from arbitrary data sources into conceptual schema. Principles of mapping of source data models (structured, semistructured and unstructured) basic constructions into conceptual data model are considered. Mapping from data sources into conceptual schema is defined as an algebraic program. A computationally complete language is used to model the subject domain. Modelling means of the conceptual level are insensitive to the extension of the conceptual data model. In other words, expanding the conceptual data model is reduced to introducing new concepts within this model using OPENMath formalism. This property is the distinctive feature of the proposed approach to data integration.

7 Conclusions

In this paper the issues of creating and accesing the integrated databases by means of conceptual level entities are investigated. Outcome of this research is a conceptual data model with orientation to data integration. The proposed conceptual data model is based on the OPENMath formalism and is an extensible conceptual data model. To support the conceptual data model, new content dictionaries in the frame of OPENMath concept was constructed. The property of extensibility allows to integrate arbitrary data sources by using a computationally complete language. Extension of the conceptual data model leads to defining new concepts to model integrable data on the conceptual level. We introduced a hierarchical relation concept as entities of conceptual level and formalized based on the OPENMath concept. Conceptual schema is defined as a collection of OPENMath objects. To define the behavior of objects of the conceptual level, an algebra of hierarchical relations was developed. By means of algebraic programs, so-called reasoning rules are interpreted. These rules are used to model mediator, data warehouse and data cube concepts. The formal basis of these rules is the proposed by us mathematical model of data integration concept. Principles of mapping of the source data models basic constructions into conceptual data model are considered. It is important that the proposed approach to data integration allows to generate a mapping from data sources into conceptual schema.

References

1. D. Calvanese, G. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The $DL-Lite$ family. *JAR*, 39(3):385 – 429, 2007.

2. D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Ontology-based data access and integration. In *Encyclopedia of Database Systems*, pages 1 – 7, 2017.
3. D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Query processing under GLAV mappings for relational and graph databases. In *PVLDB*, pages 61 – 72, 2012.
4. D. Calvanese, D. Lembo, G. D. Giacomo, M. Lenzerini, A. Poggi, M. R. Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The mastro system for ontology-based data access. *Semantic Web*, 2(1):43 – 53, 2011.
5. M. Console and M. Lenzerin. Data quality in ontology-based data access: The case of consistency. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1020 – 1026. AAAI 2014, 2014.
6. J. H. Davenport. A small openmath type system. *ACM SIGSAM Bulletin*, 34(2):16–21, 2000.
7. M. Dewar. Openmath: An overview. *ACM SIGSAM Bulletin*, 34(2):2–5, 2000.
8. X. L. Dong and T. Rekatsinas. Data integration and machine learning: A natural synergy. *VLDB*, 11(12):2094 –2097, 2018.
9. H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, USA, 2009.
10. B. Golshan, A. Halevy, G. Mihaila, and W. Tan. Data integration: After the teenage years. In *PODS'17*, pages 101 – 106, 2017.
11. J. R. Hindley and J. P. Seldin. *Introduction to Combinators and λ-Calculus*. Cambridge University Pressl, Great Britain, 1986.
12. L. A. Kalinichenko. Data model transformation method based on axiomatic data model extension. In *VLDB*, pages 549 – 555. Springer, 1978.
13. L. A. Kalinichenko. Methods and tools for equivalent data model mapping construction. In *Advances in Database Technology-EDBT'90*, pages 92–119. Italy, Springer, March 1990.
14. L. A. Kalinichenko. Effective support of databases with ontological dependencies: Relational languages instead of description logics. *Programmirovanie*, 38(6):315 – 326, 2012.
15. M. G. Manukyan. On an approach to data integration: Concept, formal foundations and data model. In *CEUR-WS*, volume 2022, pages 206 – 213, 2017.
16. M. G. Manukyan. An ontology approach to data integration. In *CEUR-WS*, volume 2790, pages 33 – 47, 2020.
17. C. Pinkel, C. Binnig, E. Jimenez-Ruiz, E. Kharlamov, A. Nikolov, A. Schwarte, C. Heupel, and T. Kraska. Incmap: A journey towards ontology-based integration. In *BTW 2017*, pages 145–164. Lecture Notes in Informatics, 2017.
18. G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, and M. Zakharyaschev. Ontology-based data access: A survey. In *IJCAI-18*, pages 5511 – 5519, 2018.
19. G. Xiao, D. Hovland, D. Bilidas, M. Rezk, M. Giese, and D. Calvanese. Efficient ontology-based data integration with canonical IRIs. In *ESWC 2018*, pages 697 – 713, 2018.
20. G. Xiao, R. Kontchakov, B. Cogrel, D. Calvanese, and E. Botoeva. Efficient handling of SPARQL OPTIONAL for OBDA. In *ISWC 2018*, pages 354 – 373. Springer, 2018.

A Basic Concepts Formalization

Data integration concepts such as hierarchical relation, hierarchical relation schema, and algebraic operations are mathematical concepts. Thus, it is natural to use the OPENMath content dictionaries to formalize these basic concepts. The content dictionaries are used to define semantical information on the basic concepts of data integration. A content dictionary which contains representation of basic concepts of the subject domain contains two types of information: one which is common to all content dictionaries, and one which is restricted to a particular basic concept definition. Definition of a new basic concept includes the name and description of the basic concept, and also some optional information about this concept. To support basic concepts of data integration and the type system of XML Schema, two content dictionaries have been developed. Below an example of a basic concept definition is considered:

<CDDefinition>
<Name> X < /Name>
<Description>
To support the concept of hierarchical relation schema we introduce the symbol X. Below we are using the *Attribution* symbol which has been defined in OPENMath.
< /Description>
<CMP> $X : Attribution^* \rightarrow \{Attribution\}$ < /CMP>
< /CDDefinition>

The above used XML elements have obvious interpretations. Only note, that the element "CMP" contains the commented mathematical property of the defined algebraic concept. Content dictionaries contain just one part of the information that can be associated with a basic concept in order to stepwise define its meaning and its functionality. Specific information pertaining to the basic concepts like the signatures is defined separately in the so-called signature files. In other words, a signature file is used to formalize the basic concepts formats.

B Semantical Constraints

As is mentioned above, to check the semantic validity of the basic concepts representations, we associate extra information with content dictionaries in the form of signature files. A signature file contains the definitions of all basic concepts signatures of some content dictionary which is associated with this file. We use Small Type System [6] to formalize the basic concept signatures. Below the definition of the signature of the above considered symbol X is provided:

<Signature name = "X">
<OMOB>
<OMA>
<OMS name = "mapsto" cd = "sts"/ >

```
        <OMA>
          <OMS name = "nary" cd = "sts"/ >
          <OMS name = "attribution" cd = "sts"/ >
        < /OMA>
        <OMS name = "attribution" cd = "sts"/ >
      < /OMA>
    < /OMOB>
< /Signature>
```

In the considered definition, the symbols *mapsto* and *nary* were defined in OPENMath. The symbol *mapsto* represents the construction of a function type. The first n-1 children denote the types of the arguments, the last denotes the return type. The symbol *nary* constructs a child of *mapsto* which denotes an arbitrary number of copies of the argument of *nary*. The operator is associative on these arguments, which means that repeated uses may be flattened/unflattened.