Extensible Data Model

M. G. Manukyan

Yerevan State University
Yerevan, 375025
mgm@ysu.am

To memory of Klara Pilosyan, my mother.

Abstract. We investigated the problems of support of XML databases. An extensible data model appears as a result of this research. The offered data model is considered as an XML application. Our approach of support of XML databases is an attempt to create a data model which is based on the XML data model and combines the best features of the relational and object databases. To model the database concepts the so-called content dictionary mechanism is used. The content dictionary mechanism is a fundamental concept of our approach by means of which the extensibility of considered data model is achieved. To give the semantic constraints relative to the database concepts a formal type system is applied. The same type system is used to define metadata. To support declarative queries an element calculus is developed. It is possible to define derivative entities, virtual databases, triggers, and general constraints. Database and metadata are considered evolving in time.

1 Introduction

XML[3] is a new important and widespread standard for representing hierarchical tagged data. Popularity of XML as a textual language for representation and exchange of data on the Web usage quickly grows. Intensive development of XML databases is performed considering XML documents as databases, and DTD as database schema. It is important that in such databases DTD and DTD-less XML document types can be supported. Moreover, the XML data model is some compromise between conventional and semi-structured DMs because in contrast to:
- semi-structured DM, the concept of database schema in the sense of conventional DMs is supported;
- conventional DMs hard schemas, there is possibility to define more flexible database schemas (for example, DTD allows to define such fields which in future may not be used) [13].

The respective DBMSs comes from research (e.g., [14] migrating the Lore data model and query language to work with XML) or from industry (e.g., [24]). These databases can be queried using specific query languages[10,22]. A big disadvantage of DTD is that it does not contain tools to include information of types and integrity constraints. An important step in this direction is the XML
Schema[2,23] which is a formalism to restrict the structure of XML documents and also to extend XML with data types. An XML query data model[12] is developed which is based on the XML Schema type system. Notice that the XML query model is the foundation of the XML query algebra[11]. In the context of XML query data model and XML query algebra an XML query language[6] is suggested. Motivation of such research are:
- XML is a formalism which allows to label the information content of diverse data sources including structured and semi-structured documents, relational databases and object repositories;
- the conventional query languages have been developed for specific kinds of data. It is necessary to develop a new query language which explicitely supports all types of XML data sources.

In this connection our research brought us to a new eXtensible Data Model (XDM). We consider XDM as an XML application for databases which is created by the same principles and based on the same formalism as OpenMath[4].

1.1 The OpenMath Overview

The OpenMath is a standard for representation of the mathematical objects, allowing them to be exchanged between computer programs, stored in databases, or published on the Web. The considered formalism is oriented to represent semantic information and is not intended to be used directly for presentation, although there exist tools to facilitate this. Any mathematical concept or fact is an example of mathematical object. The OpenMath objects are such representation of mathematical objects which assume an XML interpretation. Formally, an OpenMath object is a labeled tree whose leaves are the basic OpenMath objects integers, IEEE double precision floats, unicode strings, byte arrays, variables or symbols. Among basic objects, symbols are the most interesting since they consist of a name and a reference to a definition in an external document called a Content Dictionary (CD). For example, the following XML element

```xml
<OMS name = "plus" cd = "arith1"/>
```

represents the usual plus function, as defined in the CD “arith1” (the element name OMS indicates an OpenMath symbol). A basic OpenMath object is an OpenMath object, although its XML representation will be

```xml
<OMOBJ>
  <OMA>
    <OMS name = "plus" cd = "arith1"/>
    <OMV name = "x"/>
    <OMI>1</OMI>
  </OMA>
</OMOBJ>
```

The OpenMath objects can be built up recursively in a number of ways. The simplest is function application, for example:

```xml
<OMOBJ>
  <OMA>
    <OMS name = "plus" cd = "arith1"/>
    <OMV name = "x"/>
    <OMI>1</OMI>
  </OMA>
</OMOBJ>
```
represents application of the operation of addition to variable $x$ and to constant (to 1), where OMV introduces a variable, OMI introduces an integer constant, and OMA is the application element. Another straightforward method is attribution which as the name suggests can be used to add additional information to an object without altering its fundamental meaning. For example, the variable $x$ has the type integer and is represented by the XML:

```xml
<OMOBJ>
  <OMATTR>
    <OMATP>
      <OMS name="type" cd="ecc"/>
      <OMS name="integer" cd="ecc"/>
    </OMATP>
    <OMV name="x"/>
  </OMATTR>
</OMOBJ>
```

where OMATTR is the attribution element and OMATP element contains additional information relative to OpenMath object (variable $x$). More interesting are binding objects which are used to represent an expression containing bound variables. For example, the encoding

```xml
<OMOBJ>
  <OMBIND>
    <OMS name="lambda" cd="fns1"/>
    <OMBVAR><OMV name="x"/></OMBVAR>
    <OMA>
      <OMS name="plus" cd="arith1"/>
      <OMV name="x"/>
      <OMI>1</OMI>
    </OMA>
  </OMBIND>
</OMOBJ>
```

represents the $x + 1$ expression.

The final kind of OpenMath object is an error object which is built up from a symbol describing the error and a sequence of OpenMath objects. For example:

```xml
<OMOBJ>
  <OME>
    <OMS name="unexpectedSymbol" cd="error1"/>
    <OMS name="pluse" cd="transc1"/>
  </OME>
</OMOBJ>
```

represents the error which might be generated when an application sees a symbol it does not recognise from a CD it thought it knew about. Such object has no direct mathematical meaning.

Notice that OpenMath objects have the expressive power to cover all areas of computational mathematics [4]. The OpenMath is realized as an XML application. Its syntax is defined by syntactical rules of XML, its grammar is partially
defined by its own DTD. But because of DTD deficiencies pointed out above, and because of specificity of the given application, only syntactical validity of the OpenMath objects representation can be provided on the DTD level. To check semantics, in addition to general rules inherited by XML applications, the OpenMath defines new syntactical rules. This is achieved by means of introduction of CDs. CDs are used to assign formal and informal semantics to all symbols (for example: +, \forall, \pi, \sin, etc.) used in the OpenMath objects. CD is a collection of related symbols, encoded in XML format. In other words, each CD defines symbols representing a concept from the specific area of mathematics. It is possible to associate extra information with CDs, in particular type information. The type system can be used to formalize of signature of mathematical symbols and to check semantic validity of OpenMath objects representation. The OpenMath may be used with any type system. For example, simple signatures can be formalized by Small Type System [9]:

```xml
<Signature name="plus">
  <OMOBJ>
    <OMA>
      <OMS name = "mapsto" cd = "sts"/>
      <OMS name = "NumericalValue" cd = "sts"/>
      <OMS name = "NumericalValue" cd = "sts"/>
      <OMS name = "NumericalValue" cd = "sts"/>
    </OMA>
  </OMOBJ>
</Signature>
```

where Signature introduces a symbol (+) and the mapsto symbol is used to construct non-dependant function spaces. The first \( n - 1 \) children denote the types of the arguments, the last denotes the return type. An important feature of the OpenMath is the formal type system (an adaptation of the Extended Calculus of Constructions [19] language), which consists of basic types and primitive constructors of types. It is possible to construct by primitive constructors of types a strong (typed) OpenMath object[5]. Including formally specified type information in CDs allows to assign precise semantical meaning to OpenMath objects corresponding to mathematical notions and therefore to perform automatic validation on OpenMath objects. It is essential that validation of the OpenMath objects can be done without any knowledge of Phrasebooks (a software to convert an OpenMath object to/from the internal representation) and depends exclusively on the context determined by the CDs and on some type information carried by the objects themselves (types of the variables). In addition to above-mentioned, in [5] offered are algorithms for type inference and type checking of strong OpenMath objects and their decidability is proved.

### 1.2 Motivation of XDM

Now the relational, object, and XML databases generate definite interest among researchers of databases. The strength of the relational data model is in used formalism which allows to create an effective DBMS. Its weakness is in the type
system. The attractiveness of the object data model is explained by its type system. The argumentation of the XML databases is brought above. XDM is an attempt to create a data model which is based on the XML data model and combines best features of the relational and object databases.

We analyze XML Schema in context of its usage as data definition language for XDM. Notice that XML Schema = XML + data types. Here data type has a non-classical interpretation: The value set of that data type is essentially defined without corresponding operations[8]. Therefore on the level of XML Schema it is impossible to support derivative entities, virtual databases, integrity constraints (there is a possibility to give only simplest constraints), triggers, etc. XQuery[6] is a query language for XML which allows to give queries across all these kinds of data, whether physically stored in XML or viewed as XML via middleware. The type system of XQuery is the type system of XML Schema. XQuery is not a declarative query language (more detailed see in [8]) and does not support the database modification operations. For some reasons we cannot use XML Schema and XQuery as a data definition language and a query language for XDM correspondingly.

OWL[20] is a formalism to support semantic Web, and is based on the description logic. One of the sublanguages of OWL (OWL DL) provides the computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). Notice that the type system of OWL mainly is based on the type system of the XML Schema.

In fact, the database schema is a set of formal definitions about application domain concepts. Each definition is specifying a new concept (metadata) in terms of known definition, and with capability can be checked formally. The concept of database schema assumes a classical interpretation of data types: data type = values set + operations set. Thus, we needs a type system in the classical meaning and a formalism to define the syntax of operations of the considered type system as only structural constraints can be given on the DTD level. As mentioned above our approach to create an XML application for databases is based on the OpenMath formalism. This is explained with the fact that database concepts are mathematical objects and the OpenMath formalism can be easily extended to support the XML databases. In contrast to XML Schema XDM approach of data types allows to define the derivative entities in the database schema (the operation is considered as mathematical object and is defined in the CD). To support the XML databases the basic objects, basic and parameterized types and attribution symbols are added to OpenMath basic objects and to formal type system correspondingly: element, attribute, type, virtual element, reference, index, trigger, assert, boolean, datetime, character, etc. To give the declarative queries, an element calculus is developed. Notice that there is a possibility to give queries by means of a λ-program[16], and support the database modification operations. It is essential that we can combine the considered variants to give queries. Finally, we can define complex general constraints by means of closed formulas of element calculus language. In contrast to XML Schema our approach of creating an XML application for databases assumes that database and meta-
data are considered evolving in time. We analyze known database languages with temporal orientation and temporal extensions of XML to support point, interval, relative and periodical events[15,21]. As a formalism, we use [1,17] and special temporal facilities to support such events in XDM.

Notice that the latest three DTDs are based on the XDM objects DTD because XDM symbols, their signatures and metadata are XDM objects. In the context of XDM the database is considered as an XML document.

2 Formal Definition of XDM Objects

In this section we provide a self-contained description of XDM objects. Actually, the definitions of XDM objects coincide with OpenMath objects definitions. XDM represents database concepts as terms or as labeled trees (in general case, graphs) that are called XDM objects or XDM expressions. The definitions of an abstract XDM object are the following.

**Basic objects.** The basic XDM objects form leaves of the XDM object tree (graph). The following XDM basic objects are considered:

- **b1.** To model the constants of short, unsigned short, integer, unsigned integer, long, unsigned long, character, float, double, string, logical, octet, enumeration, datetime, interval, and time period types the corresponding XDM basic objects are introduced.

- **b2.** XDM basic bytarray object is meant to model a sequence of bytes.

- **b3.** XDM basic symbol object encodes two fields of information, a *name* and a content dictionary.

- **b4.** To model the concepts of variable and type of the programming languages, the concepts of element and reference of the XML Schema, the concepts of index, unique index, virtual element, trigger, assert of the SQL the corresponding XDM basic objects are defined (the variables of different sorts).

**Compound objects.** Compound XDM objects are built recursively as follows:

- **c1.** Basic XDM objects are XDM objects.

- **c2.** If $F, A_1, ..., A_n \ (n \geq 0)$ are XDM objects, then $\text{application}(F, A_1, ..., A_n)$ is an XDM application object.

- **c3.** If $S_1, ..., S_n$ are XDM symbols, and $A, A_1, ..., A_n \ (n > 0)$ are XDM objects, then $\text{attribution}(A, S_1, A_1, ..., S_n, A_n)$ is an XDM attribution object and $A$ is the object stripped of attributions.

- **c4.** If $B$ and $C$ are XDM objects, and $v_1, ..., v_n \ (n \geq 0)$ are XDM variables or attributed variables, then $\text{binding}(B, v_1, ..., v_n, C)$ is an XDM binding object.

- **c5.** If $S$ is an XDM symbol and $A_1, ..., A_n \ (n \geq 0)$ are XDM objects, then $\text{error}(S, A_1, ..., A_n)$ is an XDM error object.

Brief explanations of some XDM objects is brought below:

**Bytearrays** are sequence of bytes. The difference between strings and bytearrays is the following: a character string is a sequence of bytes with a fixed interpretation, whereas a bytarray is an uninterpreted sequence of bytes with no intrinsic meaning.
Application constructs an XDM object from a sequence of one or more XDM objects. The first argument of application is referred to as “head” while the remaining objects are called “arguments”. An XDM application object can be used to convey the mathematical notion of application of a function to a set arguments like in $\text{count}(x)$. For instance, suppose that the XDM symbol $\text{count}$ is defined in a content dictionary for aggregate functions, then $\text{application}(\text{count}, x)$ is the abstract XDM object corresponding to $\text{count}(x)$.

Binding objects are constructed from XDM object, and from a sequence of zero or more variables followed by another XDM object. The first XDM object is the “binder” object. Arguments 2 to $n-1$ are always variables to be bound in the “body” which is the $n^{th}$ argument object. Binding can be used to express functions or logical statements. Binding of several variables as in $\text{binding}(B, v_1, v_2, \ldots, v_n, C)$ is semantically equivalent to composition of binding of a single variable, namely $\text{binding}(B, v_1, \text{binding}(B, v_2, (\ldots, \text{binding}(B, v_n, C)\ldots))$.

Attribution decorates an object with a sequence of one or more pairs made up of an XDM symbol, the “attribute”, and an associated XDM object, the “value of the attribute”. The value of the attribute can be an attribution object itself. Composition of attributions, as in $\text{attribution}(\text{attribution}(A, S_1 A_1, A_2, \ldots, S_k A_k, A_{k+1}, \ldots, S_n A_n), S_{k+1} A_{k+1}, \ldots, S_n A_n)$, is semantically equivalent to a single attribution, that is $\text{attribution}(A, S_1 A_1, A_2, \ldots, S_k A_k, S_{k+1} A_{k+1}, \ldots, S_n A_n)$.

Error is made up of an XDM symbol and a sequence of zero or more XDM objects. This object has no direct meaning in the context of database.

3 XDM Symbols

In this section a minimal set XDM symbols are considered to support the XML databases.

3.1 XDM objects and Types Concept

As mentioned above a formal type system[5] is offered in the OpenMath based on the Extended Calculus of Constructions (ECC). The ECC consists of an underlying term calculus and a set of inference rules of judgements. The basic expressions of the term calculus are called terms. The constants (basic types), variables, types constructors (for instance, $\Sigma x : A.B$) are examples of terms. The later is a constructor of the cartesian product type, intuitively represents the set of (dependent) pairs of elements of $A$ and $B$ ($B$ may be dependent of elements of $A$): $\{(a, b) | a \in A, b \in B(a)\}$. Elements of $\Sigma x : A.B$ can be analyzed by using the two projections: $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$ (the projections are also terms).

XDM type system is an extension of the OpenMath formal type system. The considered type system is used for formalization of signature of XDM symbols and to define metadata. Including formally specified type information in XDM CDs allows to assign precise semantical meaning to XDM objects corresponding
to mathematical notions and therefore to perform automatic validation on XDM objects. This system is represented in XDM CD ecc. The considered CD consists of basic types symbols (constants), object constructors, and types constructors symbols. XDM objects in the class defined below are called typed XDM objects and are built using attribution, application, and symbols from ecc or defined in CDs that use ecc.

**Basic Objects.** The XDM symbols considered in the CD ecc: short, ushort, int, uint, long, ulong, char, boolean, string, float, double, date, time, datetime, interval, timeperiod, octet, enumeration, bytearray, id, idref, idrefs, element, symtype, xdmtype are reserved symbols for XDM types and they are typed XDM objects. They correspond to the symbolic constant terms short, ushort, int, uint, long, ulong, etc. Notice that the symbol xdmtype has no type, the type of symtype is xdmtype and the type of everything else is symtype.

XDM symbols defined in CDs that use ecc are typed XDM objects and correspond to constants of appropriate type (as defined in their signature).

Basic objects like int, string, bytearray, etc. are typed XDM objects and correspond to constants of type integer, string, bytearray, and etc. Basic objects like variable, type, element, reference, index, unique index, virtual element, trigger and assert are typed XDM objects and correspond to variables.

**Attribution.** If \( v \) is one of XDM basic object variable, type, element, virtual element, index, unique index , and \( t \) is a typed XDM object, then \[ \text{attribution}(v, \text{type } t) \] is typed XDM object. It denotes an XDM variable with type \( t \) (typed variable), \( v \mid t \) and correspond to the judgement \( v : t \). Thus, a type attribution symbol can be used for expressing a typing judgment according to the rules of the ECC.

**Abstraction.** If \( v \) is a XDM variable and \( A, t \) are typed XDM objects, then \[ \text{binding} (\lambda v : t . u, A) \] is typed XDM object. It corresponds to \( \lambda v : t . u \)[\( v \leftarrow A \)], where \( u \) corresponds to the abstraction symbol can be used for expressing a typing judgment according to the rules of the ECC.

**Application.** If \( F \) and \( A \) are typed XDM objects, then \[ \text{application} (F, A) \] \( \) is typed XDM object. It corresponds to the lambda term \( (\hat{F} \hat{A}) \). The following XDM object \[ \text{application} (F, A_1, A_2, ..., A_n) \] corresponds to the term \( (\ldots((F \hat{A}_1 \hat{A}_2)\ldots \hat{A}_n) \) and is accordingly abbreviated as \( FA_1A_2...A_n \). The mathematical meaning of abstraction and application is given by \( \beta \)-reduction rule: \( (\lambda v : t . u) \Omega \leftrightarrow u[v \leftarrow \Omega] \), where \( u[v \leftarrow \Omega] \) denotes the term obtained from \( u \) by simultaneously substituting every occurrence of \( v \) by \( \Omega \).

**Function Space.** The type constructor for the function space type is \( \Pi \text{Type} \). If \( t \) and \( u \) are typed XDM objects, and \( v \) is an XDM variable, then \[ \text{binding} (\Pi \text{Type}, \text{attribution} (v, \text{type } t), u) \] is typed XDM object. It represents the type of functions mapping an argument \( v \) of type \( t \) to a result of type \( u \). It corresponds to the term \( \Pi v : t . u \). If \( v \) does not occur in \( u \), then the object denotes \( t \to u \).

**Cartesian Product.** The type constructor for the cartesian product type is \( \Sigma \text{Type} \). If \( t \) and \( u \) are typed XDM objects, and \( v \) is an XDM variable, then \[ \text{binding} (\Sigma \text{Type}, \text{attribution} (v, \text{type } t), u) \] is typed XDM object. It
represents the type of pairs consisting of an object $v$ of type $t$ and an object of type $u$: namely $\Sigma v : t.\hat{u}$. If $v$ does not occur in $u$, then the object denotes $t \times u$.

**Pair.** If $A_1, A_2, S$ are typed XDM objects, then application$(Pair, A_1, A_2, S)$ is typed XDM object. Notice that $S$ is an optional XDM object and represents the type of the pair. It corresponds to the term $(A_1, A_2)_S$.

The following abbreviation will be used: $(A_1, A_2, ..., A_{n-1}, A_n)_S$ for the nested pair $(...⟨A_1, A_2⟩_{S_2}...A_n⟩_{S_n}$ with $S = \Sigma x_{n-1};(\Sigma x_{n-2}; ...(\Sigma x_1 : S_1,S_2)...,S_{n-1})_n$. Pairs are typically used for data structures.

**Projection.** If $A$ is a typed XDM object, then application$(PairProj1, A)$, application$(PairProj2, A)$ are typed XDM object and they denote first and second projection. They correspond to the terms $(\pi_1 \hat{A})$, and $(\pi_2 \hat{A})$.

The behavior of pairing and projection is given by the $\sigma$-reduction rule: $(\pi_i ⟨A_1, A_2⟩_S) \rightarrow_\sigma A_i (i = 1, 2)$.

**New Binders.** If $v$ is a XDM variable, $B$ is a typed XDM symbol other than lambda, PiType, SigmaType and $t, C$ are typed XDM objects, then binding$(B, \sigma)$ is typed XDM object and denotes the logical function that binds variable $v$ of type $t$ in the object $C$. It corresponds to the term $(B i(\lambda v : t.\hat{C}))$. In particular, by means of this construction we can model the universal and the existential quantifiers.

To support the element type concept of XML we use the parametric types mechanism of ECC. Notice that parametric types are function types. The following parametric types are considered:

**Sequence.** If $A_1, A_2, ..., A_n (n \geq 1)$ are XDM objects of element type, then application$(sequence, A_1, A_2, ..., A_n)$ is a typed XDM object (the object information items match the objects in sequential order);

**Choice.** If $A_1, A_2, ..., A_n (n \geq 1)$ are XDM objects of element type, then application$(choice, A_1, A_2, ..., A_n)$ is a typed XDM object (the object information items match one of the objects);

**All.** If $A_1, A_2, ..., A_n (n \geq 1)$ are objects of element type, then application$(all, A_1, A_2, ..., A_n)$ is a typed XDM object (the object information items match the objects in any order).

Notice that the value set of element type is set of well-formed XML element definitions. Let $F \in \{sequence, choice, all\}$. Then $\Pi y : (\Sigma x : element.element).element$ is the signature of $F$ as application$(F, A_1, A_2, ..., A_n)=$ =application$(F, A_1, application(F, A_2, ..., A_n))$.

**ZeroOrOne.** If $A$ is an object of element type, then application$(zeroOrOne, A)$ is a typed XDM object (the object $A$ may occur zero or one times).

**ZeroOrMore.** If $A$ is an object of element type, then application$(zeroOrMore, A)$ is a typed XDM object (the object $A$ may occur zero or more times).

**OneOrMore.** If $A$ is an object of element type, then application$(oneOrMore, A)$ is a typed XDM object (the object $A$ may occur one or more times).

Let $F \in \{zeroOrOne, zeroOrMore, oneOrMore\}$. Then $\Pi x : element.element$ is the signature of $F$.

**Inference rules.** The considered inference rules of types of XDM objects coincide with the inference rules of types of the OpenMath objects[5] as XDM objects...
= OpenMath objects + new basic objects (to support constants of XDM basic types and variables of different sorts (element variable, type variable, assert variable, and etc.). In other words, to infer the types of the considered basic objects the corresponding inference rules of analog basic objects of the OpenMath are sufficient.

3.2 Attribution Symbols

We use attribution symbol to add additional information to XDM object (for example, variable, element, attribute, trigger, etc.) without altering its fundamental meaning.

To define an attribute for XDM element an attribute symbol is used. The value of this symbol is an XDM attribution object by means of which we can define a typed XDM attribute. To define properties of an attribute an use and a fixed symbols are used. The value of use symbol may be: required (the attribute must always be provided) or optional (no default value of the attribute is provided). The value of fixed symbol can be an XDM basic object (for example, integer, string, date, e.g.). To define the value of an attribute by default a default symbol is used. Notice that the symbols default and fixed are used to define the content of an element also by default or fixed correspondingly. The value of an exp symbol may be an application or a binding object. The value of a nillable symbol may be an yes or a no symbol. If the value of nillable symbol is symbol yes then an element can have the empty instance. Otherwise, if the value of nillable symbol is symbol no then an element can not have the empty instance. To give identity-constraints the following symbols are considered:

- primaryKey: the value of this symbol is an element or an attribute, or an application(Pair,elmName/attrName,..,elmName/attrName) object.
- unique: the value of this symbol is defined in analogy to primaryKey.
- foreignKey: the value of this symbol is an application(Pair,elmName/attrName,..,elmName/attrName,elmName,elmName/attrName,..,elmName/attrName) object. In the context of foreignKey symbol an onUpdate and an onDelete symbols are used. The value of this symbol can be the following symbols: cascade or setNull. Considered symbols to support of identity - constraints have the same meaning as in SQL.

To give predicative-constraint a check symbol is used. The value of this symbol is an XDM application object. Notice that the identity-constraints and predicative-constraints can be named. Analogously, we can introduce the corresponding symbols to support triggers. To give application information and human information an info symbol and a comment symbol is used respectively. The value of info symbol is an XDM object, and the value of comment symbol is an XDM string object. Database and metadata are considered evolving in time. To support the temporal constituent of existence and the temporal constituent of belief the following symbols are introduced:

- belief: the value of this symbol is a temporal constituent of belief during which the system “believes” into existence of some fact. The value of temporal constituent of belief is provided by the system;
valid: the value of this symbol is a temporal constituent of existence during which some fact exists according to user beliefs. The semantics of temporal constituent of existence is rather complicated and generally is defined by some mathematical formula.

To support the temporal constituent of existence and the temporal constituent of belief for metadata a mbelief and mvalid symbols are introduced. These symbols are defined analogously to the belief and the valid symbols. Let $x$ is a metadata, then by means of attribution($x, \text{mbeliefValue}, \text{mvalidValue}$) construction we can connect the temporal constituents with $x$ metadata.

4 XDM Metadata and Database

Below the DTD for XDM metadata is considered:

```
<!ELEMENT name (#PCDATA) >
<!ELEMENT url (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ENTITY % xdmobjectdtd SYSTEM "xdmobj.dtd">
%xdmobjectdtd;
<!ELEMENT info (xdmobj?)>
<!ELEMENT md (name,url,(comment | info | xdmattr)+)>
</!−− end of DTD for XDM Metadata −− >
```

Element name. The content of this element is a name of XDM metadata. Element url. The content of this element should be a valid URL where the source file for XDM metadata encoding can be found. Element comment. This element contains information which is intended for human consumption (user information). Element info. This element contains information which is intended for automatic processing (application information). Element md. The content of this element contains a series of xdmattr elements, each of which defines a new metadata (a new concept in the considering application domain).

An XDM metadata is an instance of XDM metadata DTD. In turn, an XDM metadata is a DTD again. An XDM database is an instance of such DTD where XDM metadata constraints must be held.

5 Query Language

To give declarative queries an element calculus is offered. It also is possible to gives queries by means of $\lambda$-expressions. Generally, we can combine the considered variants of queries.
5.1 Element Calculus

Element calculus formulas (simply formulas) are used to give assertions (e.g., consistency constraints of XDM database), to support virtual elements and triggers, to express declarative queries to XDM Database. To specify formulas a variant of the multisorted first order predicate logic language is used. Notice that element calculus is developed in the style of object calculus[18]. If \( e \) is a path expression[7] then \((e)\) is the result of this path expression converted to multiset. Let us \( pe \) be a path expression and \( x \) is element variable.

The atoms are defined according to the following rules:

a1. \((pe)(x)\) is an atom means that \( x \in (pe) \).

a2. \( y_1 \Theta y_2 \) is an atom, where \( y_1 \) and \( y_2 \) are element variables or constants, \( \Theta \in \{=, \neq, >, \geq, <, \leq\} \).

The formulas are defined recursively according to the following rules:

f1. Any atom is a formula.

f2. If \( \psi \) is a formula, then \( \text{distinct}(\psi) \) is a formula. The truth set of this formula is a set of elements.

f3. If \( \psi \) is a formula, then \( \bar{\psi} \) is a formula (not \( \psi \)).

f4. If \( \psi_1 \) and \( \psi_2 \) are formulas, then \( \psi_1 \land \psi_2 \) and \( \psi_1 \lor \psi_2 \) are formulas (\( \psi_1 \) and \( \psi_2 \) and \( \psi_1 \lor \psi_2 \)).

f5. If \( \psi_1 \) and \( \psi_2 \) are formulas, then \( \psi_1 \rightarrow \psi_2 \) is a formula (if \( \psi_1 \) then \( \psi_2 \)).

f6. If \( \psi \) is a formula, then \( \exists x(\psi) \) is a formula (for some \( x, \psi \)).

f7. If \( \psi \) is a formula, then \( \forall x(\psi) \) is a formula (for all \( x, \psi \)).

f8. If \( \psi \) is a formula, then \( \text{groupBy}(x_1, x_2, \ldots, x_k, \psi)(x) \) is a formula means \( x \in \{\text{groupBy}(x_1, x_2, \ldots, x_k, \psi)\} \). Each element of truth set of this formula contains elements \( x_1, x_2, \ldots, x_k \) and collection of elements \( <y_1, y_2, \ldots, y_m> \) which are included in the truth set of \( \psi \) with fixed values for \( x_1, x_2, \ldots, x_k \) (here \( x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_m \) are free element variables of \( \psi \)).

f9. If \( \psi \) is a formula and \( e_1, e_2, \ldots, e_k \) are expressions, then \( \text{orderBy}(e_1 \text{ order}, e_2 \text{ order}, \ldots, e_k \text{ order}, \psi) \) and \( \text{orderBy}(e_1 \text{ order}, e_2 \text{ order}, \ldots, e_k \text{ order}, \psi)(x) \) are formulas (where order:=asc|desc, and the truth sets of these formulas are lists). The function \( \text{orderBy} \) is defined ordering relationship on the truth set of \( \psi \).

f10. If \( e \) is an expression, then \( x := e \) is a formula (assignment).

f11. If \( \psi \) is a formula, then \( \psi \text{ valid temporalExp}^1 \) and \( \psi \text{ belief temporalExp} \) are formulas.

f12. If \( \psi \) is a formula, then \( \psi \text{ with identificationList} \) is a formula.

f13. If \( \psi \) is a formula, then \( (\psi) \) is a formula.

5.2 Aggregate Functions

We consider the following aggregate functions: \( \text{min}, \text{max}, \text{count}, \text{sum}, \text{avg} \). In these functions as an argument an expression is used. Considered functions have well known interpretations.

1 The detailed discussion of the temporal aspects of XML databases are beyond the topic of this paper.
5.3 Modification of the Database

The following conventional the database update operators are considered: *insert*, *delete* and *update*.

*Insertion elements.* Syntactically the operator *insert* is defined as follows: *insert* (selector, element constructor). To define the scope of the *insert* operator a selector construction is used. Notice that the selector construction is a restricted XPath [7] expression. The element constructor (is defined in the same way as in XQuery) is used to generate element (s) to insert in database.

*Deletion elements.* Syntactically the operator *delete* is defined as follows: *delete* (expression). To extract the deleting element (s) from database an expression construction is used.

*Updating elements.* Syntactically the operator *update* is defined as follows: *update* (expression, element constructor). To extract the updating element (s) the expression construction is used. The element constructor is used to assign new value (s) to the element (s).

6 Conclusion

The paper proposes an approach for creating a data model which is based on the XML data model and combines the best features of the relational and object databases. In the frame of the offered approach an extensible data model as an XML application is developed. An important feature of that application is a formal type system which is an extension of the OpenMath formal type system. In result of such extension, the attribution symbols, the basic types, and the parametric types symbols for databases are refinemented in the frame of the OpenMath formal type system. The considered type constructors are sufficient to formalize the signatures of XDM symbols. Notice that the same formal type system is applied to define metadata. We use the so-called content dictionary mechanism to model the database concepts. The content dictionary mechanism is a fundamental concept of our approach by means of which the extensibility of offered data model is achieved. To model the fundamental concepts of databases an extension of OpenMath basic objects is suggested. The considered formalism for XDM is sufficient for application different programming paradigms to model the relational, object and XML data. To give declarative queries an element calculus is offered. It also is possible to give queries by means of λ-expressions. Moreover, one can combine considered variants of queries. Using element calculus and capabilities of the applicative programming it is possible to define derivative entities, virtual databases, triggers, and general constraints. We can use the same formalism to support the database modification operations. Database and metadata are considered evolving in time. Finally, the considered formalism to support the XML databases is sufficient to represent database concept on the Web.
References

7. J. Clark, S. DeRose (Editors), XML Path Language. http://www.w3.org/TR/
A XDM DTD Encodings

An encoded XDM object is placed inside `xdmobj` element.

```xml
<!DOCTYPE DTD for XDM Objects -->
<!ENTITY % xdmel " xdmsh | xdmush | xdmvl | xdmch | xdmbl | xdmstr |
| xdmf | xdmdb | xdmob | xdmot | xdmtp | xdmfit | xdmis | xdmri | xdmui |
x dmattr | xdmv | xdmvir | xdmuidx | xdmv | xdmidx | xdmtrg | xdmelm | xdma |
x dmattr | xdmbind | xdmne | xdmtype | xdm | xdmast” >
<!ENTITY % xdmvar “ xdmv | xdmattr " -->
<!ELEMENT xdmEMPTY>  
<!ATTLIST xdm name CDATA #REQUIRED cd CDATA #REQUIRED>  
<!ELEMENT xdmv EMPTY>  
<!ATTLIST xdmv name CDATA #REQUIRED>  
<!ELEMENT xdmtype EMPTY>  
<!ATTLIST xdmtype name CDATA #REQUIRED>  
<!ELEMENT xdm EMPTY>  
<!ATTLIST xdm name CDATA #REQUIRED>  
<!ELEMENT xdmelm EMPTY>  
<!ATTLIST xdmelm name CDATA #REQUIRED>  
<!ELEMENT xdmidx EMPTY>  
<!ATTLIST xdmidx name CDATA #REQUIRED>  
<!ELEMENT xdmuidx EMPTY>  
<!ATTLIST xdmuidx name CDATA #REQUIRED>  
<!ELEMENT xdmui EMPTY>  
<!ATTLIST xdmui name CDATA #REQUIRED>  
<!ELEMENT xdmvir EMPTY>  
<!ATTLIST xdmvir name CDATA #REQUIRED>  
<!ELEMENT xdmast EMPTY>  
<!ATTLIST xdmast name CDATA #REQUIRED>  
<!ELEMENT xdmsh (#PCDATA)>  
<!ELEMENT xdmsh (#PCDATA)>  
<!ELEMENT xdmvir (#PCDATA)>  
<!ELEMENT xdmui (#PCDATA)>  
<!ELEMENT xdmui (#PCDATA)>  
```
<! -- long -->
<!ELEMENT xdm1 (#PCDATA)>
<! -- Ulong -->
<!ELEMENT xdmul (#PCDATA)>
<! -- char -->
<!ELEMENT xdmch (#PCDATA)>
<! -- boolean -->
<!ELEMENT xdmbl (#PCDATA)>
<! -- string -->
<!ELEMENT xdmstr (#PCDATA)>
<! -- float -->
<!ELEMENT xdmf (#PCDATA)>
<! -- double -->
<!ELEMENT xdmd (#PCDATA)>
<! -- byte array -->
<!ELEMENT xdmbr (#PCDATA)>
<! -- octet -->
<!ELEMENT xdmob (#PCDATA)>
<! -- enumeration -->
<!ELEMENT xdmecn (#PCDATA)>
<! -- datetime -->
<!ELEMENT xdmdt (#PCDATA)>
<!ATTLIST xdmdt qualifier (date | time | datetime) “date”>
<! -- time interval -->
<!ELEMENT xdmit (#PCDATA)>
<! -- time period -->
<!ELEMENT xdmtp (#PCDATA)>
<!ATTLIST xdmtp qualifier (date | time | datetime) “date”>
<! -- apply constructor -->
<!ELEMENT xdma (%xdmel;)+>
<! -- binding constructor -->
<!ELEMENT xdmblind ((%xdmel;), xdmvar, (%xdmel;))>
<!ELEMENT xdmvblvar (%xdmvar;)+>
<! -- error -->
<!ELEMENT xdmec (xdms,(%xdmel;)*)>  
<! -- attribution constructor & attribute pair constructor -->
<!ELEMENT xdmattr (xdmatp,(%xdmel;))>
<!ELEMENT xdmattrp (xdms,(%xdmel;)+)>  
<! -- reference -->
<!ELEMENT xdmr EMPTY>
<!ATTLIST xdmr ref CDATA #REQUIRED>
<! -- XDM object constructor -->
<!ELEMENT xdmobj (%xdmel;)>  
<! -- end of DTD for XDM -->