Integration of Heterogeneous Semistructured Data Models in the Canonical One

L. A. Kalinichenko
Institute for Problems of Informatics
Russian Academy of Sciences
Vavilov str. 30/6, Moscow, V-334, 117900
E-mail: leonidk@synth.ipi.ac.ru

Abstract

To provide for interoperability of heterogeneous information objects it is required to establish a global, uniform view of the underlying digital collections and services. An information model is needed which is able to express uniformly the structure and semantics of heterogeneous data collections as well as the available services. Usually the mediator's layer is introduced to provide the users with the metainformation uniformly characterizing content of the underlying collections and with the canonical information model applied for definition of such metainformation and for querying integrated world of digital collections.

The paper focuses on the canonical model intended for homogeneous representation of various semistructured and hybrid data models that have been developed recently with orientation on the data contained in Web sites. The paper provides a short overview of several representative semistructured and hybrid data models. The canonical information model (based on the SYNTHESIS language) intended for uniform representation of heterogeneous information in mediators is introduced. The paper shows how different categories of semistructured models can be equivalently and homegeneously represented in the canonical model.

1 Introduction

Digital Libraries as advanced forms of information systems are based on theory and practice of acquisition, modeling, management and dissemination of digital information via networking media. The explosion of the World Wide Web and of the multimedia technology provide initial techniques for organization of digital information collections and indicate a technical framework for the libraries of the future. Digital Libraries – complex and advanced forms of information systems – are considered as distributed repositories of knowledge. Numerous forms of digital collections representations could be included in such repositories. Therefore, the fundamental for Digital Libraries problem should be resolved: how to map huge variety of digital collections into their uniform representation and how to support the basic library function of providing access to the integrated collection of heterogeneous information ?

First Russian National Conference on
DIGITAL LIBRARIES:
ADVANCED METHODS AND TECHNOLOGIES,
DIGITAL COLLECTIONS
October 19 - 21, 1999, Saint-Petersburg, Russia

The spectrum of possible forms of data representations ranges from structured and/or object databases through knowledge bases and semistructured data to completely unstructured data (containing texts or images). Database systems are able to provide comprehensive data management functions relying on a schema to which the data should conform. The database systems exploit the semantic content defined by the schema to perform such functions as index creation, query optimization, constraint management, interpretation of database objects behaviour. Knowledge bases are more flexible forms of information representation where significant portion of data is not directly stored but can be deduced applying rules. The schema and entity behaviour is less deterministic and more rule-based than in databases.

A large volume of information must be incorporated into a digital library without a formal schema or with a partially determined schema. There is a need to map raw information sources into forms that exhibit as much structure as possible. Here we can distinguish between sources exhibiting certain regularity and completely unstructured data. HTML pages constitute vast example of the data exhibiting certain regularity in cases when we can extract a structured representation of the data from pages containing them. Usually this task is performed by specific wrappers making possible to view the respective Web sites as autonomous heterogeneous databases. Completely unstructured, textual data are characterized by vocabularies communicating to users collections of terms and their relationships that could be found in textual documents.

Finally, many software services in the Internet represent behavioral information that also should be involved into the process of information discovery, retrieval and computation. Digital collections of specifications of services (components) represent them for identification, selection and interoperable composition of suitable behaviours.

The scale of diversity of forms of information available makes clear how hard is the problem of imposing representation uniformity and supporting integrated access to numerous heterogeneous digital collections.

Taking into account importance of collections existing in forms of the Web sites for the digital libraries, this paper focuses on the issues of integration of various structured and semistructured data models in the canonical model paradigm. The SYNTHESIS model [11] is analysed for such canonical role – fundamental for the middleware intended for the heterogeneous digital collection mediation technology.

The paper is structured as follows [1]. The paper starts with a brief survey of semistructured and hybrid data models. Then the canonical information model intended for uniform representation of heterogeneous semistructured models is introduced. Finally the paper shows how such models can be mapped into the canonical model to form their homogeneous representation.

2 Semistructured data modeling

2.1 An overview

The popularity of the Web has led to a significant body of techniques addressing the problems of accessing and querying information in the Web. Most often the data in the Web falls somewhere in between structured and unstructured data. Term 'semistructured data' denotes such representations possessing some of the following characteristics [7]:

1. the schema is not given in advance and may be implicit in the data,
2. the schema is relatively large (w.r.t. the size of the data) and may be changing frequently,
3. the schema is descriptive rather than prescriptive, i.e., it describes the current state of the data, but violations of the schema are still tolerated,
4. the data is not strongly typed, i.e., for different objects, the values of the same attribute may be of different types.

Several approaches are known to model the Web itself, structure of Web sites, internal structure of Web pages, and finally, contents of Web sites in finer granularities. The languages developed so far can be classified into the Web queries models (e.g., WebSQL [16], ADM [4]), Web site development languages (e.g., WebOQL [3]), Web data integration languages (Tsimmis, YAT [6], Ozone [15]). These languages are based on structured (ADM, OQL-doc [1] XML DTD [8]), semistructured (OEM) or hybrid data models (Ozone, YAT) considered further.

The Object Exchange Model (OEM) The Object Exchange Model (OEM) is a self-describing semistructured data model. [2] treats data as a graph with objects as the vertices and labels on the edges. All entities in OEM can be interpreted as objects with object identifiers (OIDs). Some objects are atomic literals and contain values from one of basic atomic types. Other objects are complex: their value is a set of object references denoted as a set of (label, OID) pairs. Data is not strongly structured. E.g., an object may have $0-n$ occurences of an attribute value, a type of an attribute may vary in different object instances up to the situation when in one instance this is a simple type and a complex type in another, instances may have structural differences. An OEM schema consists of a finite set of names R. To form instance of R a name function from R to set of atomic and complex object vertices is formed. An OEM database may be viewed as a labeled directed graph, with complex OEM objects as internal nodes and atomic OEM objects as leaf nodes. Named OEM objects form entry points into an OEM database.

[1] This work has been supported by the INTAS-OPEN grant 97-1109 and the Russian Foundation for Basic Research grant 98-07-91061

Araneus Data Model [4]. Another approach consists in extracting the structure from the HTML documents to present it as a view difinition above Web sites. One of the model, the Araneus Data Model (ADM), is a page oriented model intended to describe the structure of a set of homogeneous pages in a site. ADM can be considered as a subset of ODMG [17]. Unstructured HTML documents are analyzed to extract their structure. Each Web page is considered as an object with an identifier (the URL) and a set of attributes. Attributes of a page schema may have simple or complex types. Multivalued attributes are modelled by lists of tuples. ADM provides also a heterogeneous union type as an extension of the ODMG model.

Ozone Data Model Ozone [15] is a hybrid model that is intended to handle semistructured data alongside with conventional structured data. The unified representation and querying of such hybrid data is the objective. Ozone takes ODMG model (its ODMG'93 version) and its query language OQL as the core. Extension to the ODMG model uses OEM to represent semistructured portions of data. Main focus of such extension consists in allowing semistructured entities as values of the ODMG object attributes and ODMG typed values as OEM vertices treated as dynamically typed values. Main query language extensions consist in allowing combined path expressions traversing compositions of structured and unstructured data and relaxing typing constraints for semistructured data.

YAT Data Model YAT [6] is an integration system relying on a semistructured middleware model with typing abilities. Actually, the O2 (ODMG) model is used and extended with the semistructured capabilities of YAT. This leads to an ability of defining combined structured/semistructured data types and constructing respective type instances. Specific features are added to query such combined data instances.

In the following subsections the more detailed information on ADM, YAT and Ozone models will be provided: these models were selected to show how heterogeneous models can be homogenized applying the canonical model.

2.2 ADM model characterization

ADM [5] is *page*-oriented, in the sense that it recognizes the central role that pages play in the Araneus framework. In fact, ADM introduces the notion of *page-scheme* to define the structure of a set of homogeneous Web pages. ADM also provides for limited constraints capabilities to catch knowledge of specific structural properties that occur in Web sites.

In ADM, a Web page can be seen as an object with an identifier, the URL, and several attributes, one for each relevant piece of information in the page. The attributes in a page can be either simple, like text, images, binary data or links to other pages, or complex, that is, lists of items, possibly nested. ADM also provides heterogeneous union and a form type, specifically needed to model the organization of Web pages. Pages sharing the same structures are grouped in page-schemes.

Simple attributes are mono-valued and correspond to atomic pieces of information, such as text, images or (other multimedia types), and links to other pages. Complex attributes are multi-valued and represent (ordered) collections of objects, that is, lists of tuples (repeated patterns in Web pages are physically ordered). Component types in lists can be in turn multi-valued, and therefore nested lists are allowed. An important construct in Web pages is represented

by forms. Conversely, forms are used to execute programs on the server and dynamically generate pages. ADM provides a *form* type: in order to abstract the logical features of an HTML form, it is seen as a virtual list of tuples; each tuple has as many attributes as the fill-in fields of the form, plus a link to the resulting page. ADM uses a heterogeneous union type in order to provide flexibility in modeling, according to the heterogeneous nature of the Web.

The set of ADM types is recursively defined as follows (each type is either mono-valued or multi-valued):

- each base type is a mono-valued ADM type;
- LINK TO D is a mono-valued ADM type; D, the destination of the link, is (i) either a page-scheme name, P , (ii) or a union type, P_1 union P_2 union $\ldots P_n$, where each P_i is a page-scheme name;
- LIST OF $(A_1 : T_1, A_2 : T_2, \ldots, A_n : T_n)$ is a multi-valued ADM type, if $A_1, A_2, \ldots, A_n$ are attributes and $T_1, T_2, \ldots, T_n$ are ADM types;
- FORM$(A_1 : T_1, A_2 : T_2, \ldots, A_n : T_n)$ is a multi-valued ADM type, if $A_1, A_2, \ldots, A_n$ are attributes and $T_1, T_2, \ldots, T_n$ are ADM types; exactly one attribute has type LINK TOD: this is used to denote the URL of the page generated in response to the submission of the form.

ADM supports the following kinds of constraints. A *link* constraint is a predicate associated with a link. It is used to document the fact that the value of some attribute in the source page-scheme equals the value of another attribute in the target page-scheme.

For two page-schemes, P_1 and P_2 connected by a link ToP_2, a *link* constraint between P_1 and P_2 is any expression of the following forms: $A = B$ or $A = v$, where A is a mono -valued attribute of P_1, B is a mono-valued attribute of P_2, and v is a constant.

An *inclusion* constraint is an expression of the form: $P_1.A_1 \subseteq P_2.A_2$ where P_1, P_2 are page-schemes with attributes A_1, A_2 towards page-scheme P.

On a page-scheme a special constraint can be specified: when a page-scheme is *unique,* it has just one instance, in the sense that there are no other pages with the same structure. Typically, at least the home page of each site falls in this category. For a unique page-scheme, the URL is supposed to be known, and it is documented in the ADM scheme.

Finally, to allow for null values, ADM introduces *optional* attributes.

A schema of the TitleIndexPage for DB and LP bibliography Web server (http://www.informatik.uni-trier.de/~ley/db/index.html) defined in the Araneus DDL looks as follows:

```
PAGE-SCHEME TitleIndexPage
        WorkList: LIST-OF
(Authors   : TEXT;
Title      : TEXT;
Reference  : TEXT;
Year       : TEXT;
Pages      : TEXT;
AuthorList: LIST-OF(Name          : TEXT;
                    ToAuthorPage : LINK-TO
                      AuthorPage;);
ToRefPage : LINK-TO ConferencePage UNION
                    JournalPage    UNION
                    SeriesPage     UNION
                    WebPage OPTIONAL;
);
END
```

In the last section it will be shown how this schema can be expressed in SYNTHESIS.

2.3 YAT model characterization

YAT [6] relies on a semistructured data model, but enhanced with type construction. In YAT data structures are presented as labeled trees (*patterns*). Labels are used to denote values, attributes or type information thus trying to avoid distinctions between data and schema. A pattern tree is an ordered tree whose nodes are labeled with data variables or constants. The model also supports references (denoted as &<name>) to other trees/objects in order to represent sharing of information. Structural capabilities of YAT are based on the implementation of the ODMG'93 model in O_2. Figure 1 shows how O_2 schema can be expressed in YAT.

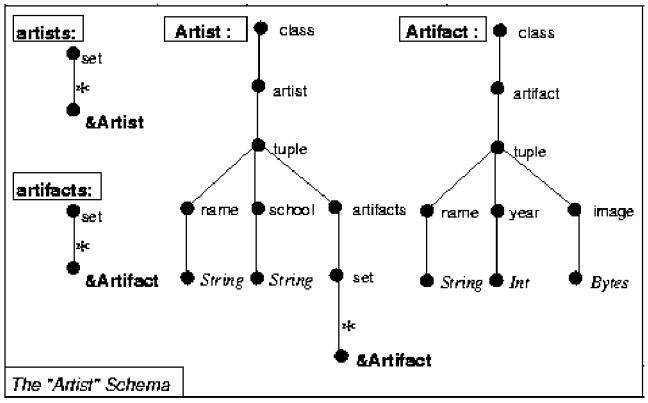

Fig. 1: O2 schema in YAT

Collections are represented through the use of the $*$ edge label. Schema and instance patterns are presented similarly. Union operation ($\vee$) is used to capture various alternative structures. For instance, a YAT pattern is either a simple node with any label and an arbitrary number of sons, all of which are YAT patterns, or a simple node with a reference.

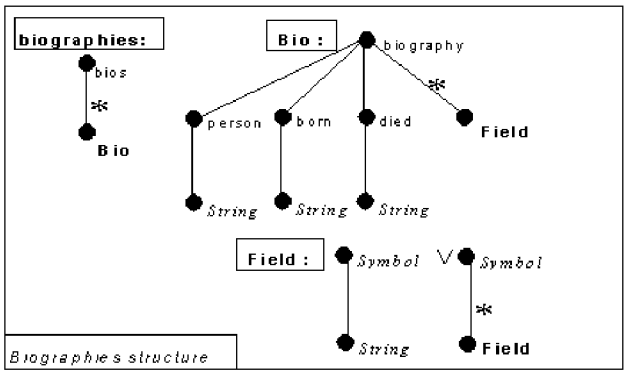

Fig. 2: XML-like structuring in YAT

How to mix structured and less-structured information in YAT is shown on Figure 2 relating to a collection of XML

documents (biographies). Each document (Bio) is described as a sequence of well defined fields (*person, born* and *died*) followed by an arbitrary number of other fields (Field). For these attributes it is only known that they have a name (*Symbol*) and that they are composed of a single string or of more nested fields. In this specific example, arbitrary information contained in biography documents (like influences, favorite places or exhibitions of artists) may be stored in these additional fields.

The following example [6] creating a view containing information about modern artists shows YAT data collection querying capabilities. The view is constructed above $O2$ database (artisits) and an XML-Wais database (biographies). The query resembles datalog rule. The use of explicit Skolem functions allows to control the creation of new pattern identifiers (Martist(N)). The biographies semistructured information (i.e., Field) is exported to the view.

```
modern_artists:
root *-> Martist(N):
  artist(-> name -> N,
         -> born -> B,
         -> died -> D,
         -> school -> S,
         -> biography *-> Field)   <=

artists:
set *-> &Artist:
  class -> artist -> tuple
       (-> name -> N,
        -> school -> S,
        -> artifacts -> set *-> &Artifact:
       class -> artifact -> tuple (-> name -> Na,
                                   -> year -> Da)),

biographies:
bios *-> Bio:
  biography( -> person ->,
             -> born -> B,
             -> died -> D,
             *-> Field),

Da > 1800,
N = P
```

Interpretation of YAT in SYNTHESIS will be shown in the last section.

2.4 Ozone model characterization

Ozone includes ODMG model with a new built-in type OEM. Thus ODMG types can be constructed that include semistructured data. Objects of the type OEM are of two categories: OEMcomplex and OEMatomic representing complex and atomic OEM objects respectively. An OEMcomplex object encapsulates a collection of (*label, value*) pairs where *label* is a string and *value* is an OEM object. The original OEM data model specification included only unordered collections of subobjects, but XML is inherently ordered. Therefore complex OEM objects are allowed with either unordered or ordered subobjects referred to as OEMcomplexset and OEMcomplexlist respectively.

The value of an OEMatomic object may have any valid ODMG type (including OEM). When the content of an OEMatomic object is of type T it is said to be of type OEM(T). Since OEM objects are actually untyped, OEM(T) denotes a "dynamic type" that does not impose any typing constraints. They are thought as *untyped containers* for the typed values.

Benefits of both models (ODMG and OEM) are available. Treating ODMG data as OEM allows queries to be written without complete knowledge of the schema, while still retaining access to all ODMG properties (such as methods, indexes, etc.). On the other hand, ODMG applications can access semistructured data using standard API's and structural optimization.

The query language for Ozone is OQL^s. The semantics of OQL^s on structured data is identical to OQL on standard ODMG data. The semistructured capabilities of OQL^s are mostly derived from Lorel.

When a *path expressions* in OQL^s query is being evaluated, a corresponding database path may involve all structured data, all semistructured data, or there may be *crossover* points that navigate from structured to semistructured data or vice-versa.

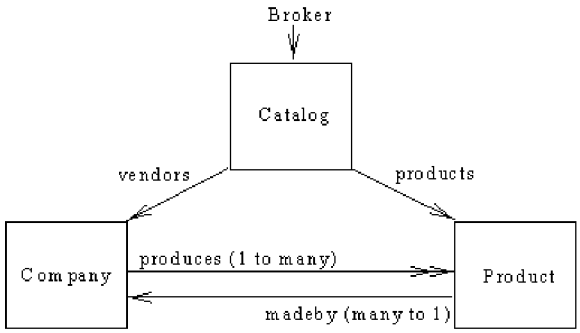

Fig. 3: ODMG retail-broker database

The example ([15]) considers a database supporting a broker that sells products on behalf of different companies. There as three ODMG classes in this database: Catalog, Company and Product. Class Catalog has one object which represents the on-line catalog maintained by the broker. The object has two attributes: a vendors attribute of type set(Company), denoting the companies whose products are sold in the catalog, and a products attribute of type set(Product), denoting the products sold in the catalog. The Company class defines a one-to-many produces relationship with the class Product of type list(Product). This relationship specifies the list of products manufactured by the company, ordered by product number. Likewise, the Product class defines the inverse many-to-one madeby relationship with the class Company, denoting the product's manufacturer.

The Company class contains other attributes such as name and address, and an inventory() method that takes a product name argument and returns the number of stocked units of the product of that name. The Product class contains other attributes such as name and prodnum (product number). The named object Broker of type Catalog provides an entry point to this database. Figure 3 depicts this schema without atomic attributes. In addition to this structured data, product-specific XML information is available for some products, e.g., drawn from Web sites of companies and analyst firms. This data might include manufacturer specifications (power ratings, weight, etc.), compatibility information if it applies (for instance, the strobes compatible with

a particular camera), a listing of competing companies and products, etc.

To integrate this XML data within the ODMG database, the Product class is enhanced with a prodinfo attribute for this product-specific data. Since this data is likely to vary widely in format, it is not possible to use a fixed ODMG type for its representation, and it is required to use the semistructured OEM data model. Therefore, the prodinfo attribute is a crossover point from ODMG to OEM data. There is also a need for referencing structured data from semistructured data. If a competing product (or company) or a compatible product appears in the broker's catalog, then it should be represented by a direct reference to the ODMG object modelling that product or company. If the competing product or company is not part of the catalog, only then is a complex OEM object created to encode the XML data for that product or company.

An example of OEM database graph for the prodinfo attribute of a product is shown on Figure 4. Note that in Figure 4, the competing product named "System12" is not part of the catalog database and therefore is represented by a (complex) OEM object; the other competing product and company are part of the catalog and are represented by references to Product and Company objects. Further, it is supposed that some review data are available in XML for products and companies. The information is available from Web pages of different review agencies and varies in structure. The example database is enhanced with a second entry point: the named object Reviews integrates all the XML review data from different agencies. Once again, the diverse and dynamic nature of this data means that it is better represented by the OEM data model than by any fixed ODMG type. Thus, Reviews is a complex OEM object integrating available reviews of companies and products. Here structured data can be referenced from semistructured data, since reviewed companies and products that are part of the catalog should be denoted by references to the ODMG objects representing them.

Figure 5 is a simplified example of the semistructured Reviews data. It is assumed that the reviews by a given agency reside under distinct subobjects of Reviews, and the names of the review agencies (ConsumersInc, ABC Consulting, etc.) form the labels for these subobjects. Reviews by ConsumersInc agency have a subject subobject denoting the subject of the review (either a product or a company), which may be a reference to the ODMG object representing the company or product, or may be a complex OEM object. Both cases are depicted in Figure 5.

The overall example scenario consists of hybrid data. Some of the data is structured, such as the Product class without the prodinfo attribute, while some of the data is semistructured, such as the data reachable via a prodinfo attribute or via the Reviews entry point.

In the last section it is shown how this example can be represented and queried in the canonical model.

3 Canonical information model for the mediator's level

The model used for uniform representation of various data and information models in one paradigm is called the "canonical" model. The model intends to provide for uniform (canonical) representation of heterogeneous digital collections (repositories of data, knowledge and programs) for their use as interoperable collections. The same model is intended also for description of ontological models of application domains.

The mediator's canonical model is based on the SYNTHESIS language [11] that has been developed for component-based information systems development in the wide range of pre-existing heterogeneous components. A set of the canonical model facilities used for the uniform representation of the information resources includes the following:

- **Frame representation facilities.** Frames are treated as a special kind of abstract values introduced mostly for description of concepts, terminological and semistructured information. In particular, the information resource metainformation (schema) is represented using the frame language. Frame representation facilities provides for expressing of arbitrary semantic associations of frames, for representation of unstructured, textual and temporal associations. All specifications in canonical model have a form of frames that become a part of the metabase. Collections of frames form worlds and contexts.
- **Unifying type system.** A universal constructor of arbitrary abstract data types as well as a comprehensive collection of the built-in types are included into a type system. Heterogeneous union type is a representative of the type system. For types a type specialization (subtyping) relationship is defined. Types are values themselves. Metatypes provide for classification of the type hierarchy. Type expressions are introduced providing for type compositions that are required to type the results of queries.
- **Class representation.** Classes provide for representing of sets of homogeneous entities of an application domain. Class hierarchies and type inheritance mechanisms make possible to define the generalization / specialization relationships. Class instances (objects) are defined on abstract data types. Metaclasses provide for introducing different classification relationships orthogonal to the class generalization relationship.
- **Multiactivity representation.** These are used for the specification and implementation of intecrconnected and interdependent application activities, for the specificaton of declarative assertions and concurrent megaprograms over the information resources. These facilities provide for specification of concurrent and asynchronous behavior of application systems and of interoperable resource environments as of dynamic discrete events systems.
- **Facilities for the logical formulae expressions.** A multisorted object calculus (typed first-order language) is used for querying the integrated set of digital collections as well as for specification of constraints and behaviour.

Information characterizing the entities and situations observed in a real world is represented in the information resource base as a collection of abstract values that can be immutable or mutable uniquely identified values (objects). In this range we can differentiate between:

- collections of self-defined objects or collections of frames (worlds);
- worlds with pre-defined frame associations;
- classes containing partially typed objects containing their own individual attributes that were not specified in a type of the class instance;

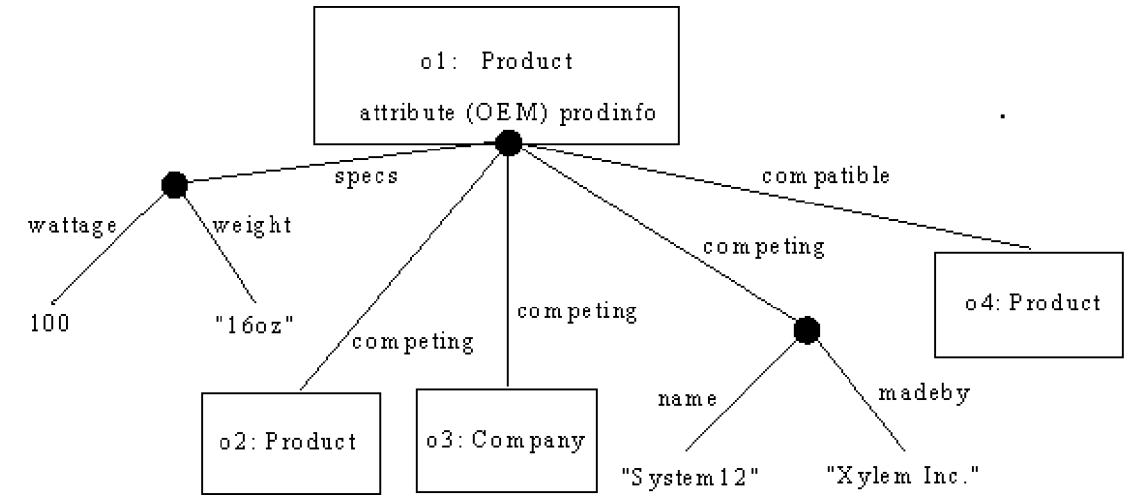

Fig. 4: OEM graph for prodinfo attribute

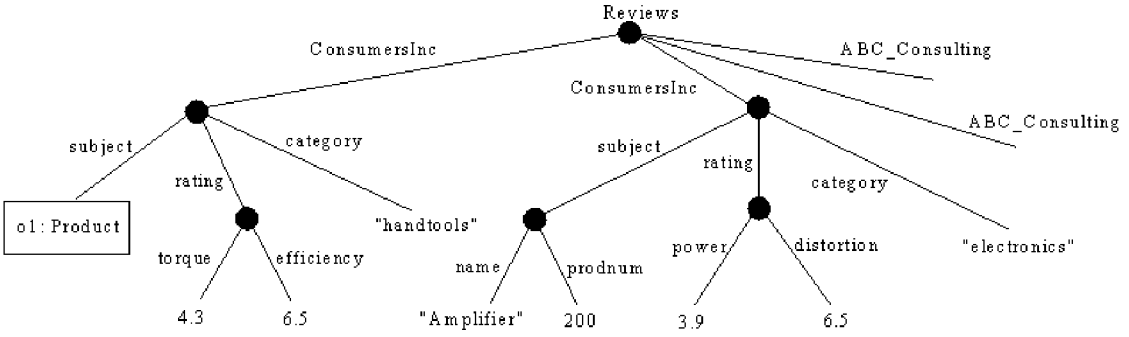

Fig. 5: Semistructured Reviews data

- strictly typed classes (a set of instance attributes is strictly fixed);
- classes of aggregates (associations of objects and frames).

3.1 Frame language

A frame is considered to be a symbolic model of a certain entity or a concept. A frame is represented as a set of attributes called slots being used for a description of properties of an entity or a concept. An order of slots given for a frame specification is significant. To each slot a collection of values can be associated; each value being generally an abstract value of arbitrary type defined in the SYNTHESIS language (of course, it may be another frame).

Any frame can acquire a unigue identifier to which the built-in slot self corresponds. The value of this slot cannot be modified.

In the frame language a knowledge base is considered as a collection of frames interrelated by means of binary associations. There is no means to represent classes of concepts or entities in the frame language: only individual instances can be represented. The facilities for specification of types and classes have higher level and extend the frame language upwards.

The frame language is an object-centered language that does not rely on a class-based object-oriented paradigm. Frames can appear as objects that exist autonomously, without types and classes to describe and create them. They are collections of slots representing both state attributes and operations. A basic mechanism for sharing of information is *delegation* instead of inheritance in class-based models. Delegation is a mechanism by which an object that cannot answer a message can delegate it to another object. Delegation serves for sharing of the characteristics of an object without sharing of a common class description (as with class inheritance).

In the SYNTHESIS language frames are used also for the metadefinition of other facilities of the language.

Frame is an important concept of the language: any language entity (including objects) is uniformly represented by means of frames. It means that each such entity possesses properties of frames given by the type (Tframe).

And finally, the frame language is used also for the external representation of constants of various SYNTHESIS types (including abstract data type values). In particular, self-defined objects (objects that exist separately and not

belong to any class) are also represented by frames. Frames can be combined into collections (worlds or contexts). The collections can include frames having arbitrary structure.

Generally, frames are abstract values, their type (Tframe) is a subtype of the abstract value type Taval. Slots of the frames are considered as functions appeared as $a : C \to V$, where a is a slot name, C is a context or a world containing a corresponding collection of frames, V is a collection of the slot values. Domains of such functions are collections of abstract values (here the collections of frames) in a context or in a world. At the same time defining a slot as a component of a frame specification a notation $a : v$ is used where v is a value belonging to V or to a subset of V.[2]

An additional metainformation can be associated with frames, slots and values. It also takes a form of frame. This metainformation provides for additional capabilities in specifyng of the complex concepts. The frame model provides for user-defined relationships (binary associations) keeping forward and inverse relationships.

Frame is always represented in figure brackets { and }. Identifiers of nested frames can be omitted. Slots and their values are separated by a colon. The values of a slot are separated by a comma. Semicolon terminates the listing of values of a slot. It is important that any frame component (identifier, slots, values) can be omitted. In case when a value of a slot is absent, semicolon is provided immediatelly after a colon. Values in frames containing no slots belong to implicit universal built-in slot.

Variables in frames are used for indication that content of their certain component can be arbitrarily chosen. Variables can get values of arbitrary types.

Separate frames in the frame base can be linked with each other by means of binary associations. For example, if a concept is a subconcept of another concept then the frames representing those concepts should be keeped in a association that reflects the corresponding relationship.

To set an association between the frame A and the frame B a slot is included into the frame A with a name of an association required. An identifier of the frame B should become a value of this slot.

3.1.1 Worlds and contexts

Frames that are contained in the frame base are subdivided into the subcollections (worlds). Every world can be seen as separate and relatively independent section of the frame base that is used for the formation of the frame collections and for their manipulation. A world is represented by a frame having the predetermined structure:

```
{< world identifier >;
in : world;
< collection of frames >
}
```

Generally a world is a unit (a module) for representation, compilation, storage and manipulation of the frame collections. If variables are used in a world, corresponding type definitions should be included into the same context where the world is contained. Such context is used for compilation and interpretation of the world. Each frame and all its nested frames should belong to one and the same world. Collections of worlds can form contexts. A context is also represented by a frame having the predetermined structure:

[2]Each frame, its components (slots and slot values) as well as its associations with other frames are considered as existing in time.

```
{< context identifier >;
in : context;
member :< list of world or context identifiers >
}
```

The association member (member_of is its inverse association) is used for establishing of membership of frames (or contexts) in contexts. Contexts in their turn can participate in various associations to form hierarchical or network structures. Context (world) names can be used as parameters of functions and predicates providing for localization of the corresponding functions in the context (or in the world) given. For data search, a context (or a world) explicitely indicates a space for the search by means of a certain formula.

3.2 Type system of the SYNTHESIS language

3.2.1 Principles of the type system construction

Abstract data types (ADT) constitute the basis for the type system of the language providing for construction of arbitrary data types. ADT definitions include a description of behaviour of the type values by means of specifications of their operations. We emphasize specification of ADT here separated from implementations. Parametric specifications of ADT are possible. ADT can define object or non-object types constituting collections of admissible values of the respective kind.

A set of built-in data types is included into the language alongside with ADT. The built-in types are defined by their interfaces. Built-in and arbitrary types (given by ADT) are subdivided into concrete and generic (parametric) types. Generic types are treated as functions of free variables contained in type specifications that deliver types as their resulting values. An application of such type functions provides for concretization of generic types. Usage of generic types as parameters for the function specifications is allowed for static and for dynamic type concretization.

The multilevel type system of the language is organized as follows. A set of all values expressible in the language (including frames and objects that are not type objects themselves) is located on the zero level (the level of values). At this level various computations on the values can be defined.

On the first level (the level of types) type objects are located (as well as type functions and expressions) that provide for the definition of the concrete and generic types and for the concretization of the latter ones. On the second level (the level of "types of types") the metatype objects are located that include the types of the first level as their instances. On the third level the metatypes objects are located that include the metatypes of the second level as their instances, and so on. Actually, metatypes are metaclasses having types or metatypes as their instances.

Thus the multilevel type system that imposes a classification relationship on the data types is defined.

Alongside with the type classification relationship, specific facilities are introduced for establishing of the subtype relationship. In the subtyping hierarchy any value of a type can be used everywhere where a value of the supertype is expected. Multiple subtyping is allowed. The classification relationship that is being set on types is orthogonal to the subtype relationship. It is essential that only types (metatypes) belonging to one and the same classification level can participate in the subtype relationship.

In the structure of the type system of the SYNTHESIS language the Taval type is used as the root of the lattice, the Tnone type - as its bottom. The Tnone type represents

abstract values with an "empty" semantics. For instance, a value none belongs to Tnone and can be returned as a result of functions creating an empty result of any type. All object types are subtypes of the Tsynth_object type in which an attribute self is defined. self is used as an object self-reference. [3]

A class combines properties of a type and of a set. A class supports a set of objects of a given type that constitutes an extent of the class.

Collection constitutes a virtual built-in type (instances of this type never exist). Sets, bags and sequences are defined as subtypes of the collection type. Sets resemble classes in a sense that they represent sets of ADT values or sets of objects. One and the same object can belong at the same time to several sets and to change dynamically its participation in them. A set is distinguished from a class in the following: 1) an object cannot be created by sets; 2) for a set there is no difference between the own and total extent as for a class; 3) classes as sets of objects are automatically supported; sets should be created by the program and supported by users.

At the same time a class is considered to be a subtype of a set type. Due to that these generally different constructs can be used quite uniformly: a class can be used everywhere where a set can be used. For instance, the object calculus formulae evaluation is based on collections of ADT values (or of objects). Classes can be used for such collections as specializations of the set type. Due to that a calculus and algebra on which the object calculus is based are closed with respect to the composition of the algebraic operations.

With each class an ADT specification is associated. This ADT defines properties of objects being class instances.

A subset of the set of built-in types of the language looks as follows:

< built − in type >::=< function type > |
< integer type > | < real type > |
< Boolean type > | < character type > |
< bit string type > |
< character string type > | < set type > |
< sequence type > | < bag type > | < array type > |
< enumeration type > | < range type > |
< union type > | < product type > | < frame type >

Notice that frame type is a built-in type as well as the union type.

All operations over typed data in the language are represented by functions. A function type defines a set of functions, each function of the set being a mapping of the function's domain (a cartesian product of the sets of the input parameters values) into a range of the function (a cartesian product of the sets of the output parameters values). Functions can be passed as parameters, can be returned as values, can be used as the abstract value attributes.

3.2.2 Type expressions

A type in the SYNTHESIS language is defined by its specification. A type also is a value represented by an object corresponding to the specification. A type can be determined by typed variables, can be inferred by the object calculus formulae and can be produced as the result of type expressions evaluation.

Main operations that are used for construction of type expressions (explicit or implicit) are operations of a type specification calculus [14]: *type reduct, meet and join.*

[3] Type specification calculus supporting the SYNTHESIS type system is defined in [14].

3.3 Partially typed objects

Objects can be completely typed (in this case all their attributes are defined by a class instance type specification), partially typed (part of their attributes are defined by a class instance type specification and other attributes can be arbitrarily defined for different instances of the class by the objects themselves) or can be completely autonomous (not associated to any class). Autonomous objects exist isolated or can be related to some worlds and are self-defined (types can be used for their definitions if required).

Kinds of associations that can be used by frames in a world can be restricted. To do that it is required to declare in a class specification the types of binary associations (of the frame language) by which frames can be interrelated. Such class corresponds to a world in a frame base. By means of such associations arbitrary relationships of objects and frames can be established to define flexibly additional information about objects.

Class specifications combined with frame language facilities give flexible abilities for the representation of information about information resources. In particular, the frame language is used for the representation of unstructured or semistructured information about application domain entities (including textual data). Such information can be keeped separately or can be related directly to objects on the basis of object - frame associations.

3.4 Object calculus formulae

3.4.1 General rules

Object calculus formulae (or simply formulae) in the SYNTHESIS language are used to define rules in object calculus programs, to formulate assertions (e.g., consistency constraints of an information resource base), to express queries to an information resource base, to form predicative specifications.

To specify formulae a variant of a typed (multisorted) first order predicate logic language is used. Every predicate, function, constant and variable in formulae is typed.

Predicates in formulae correspond to classes, worlds, collections and functions that are specified in information resource specification modules and have corresponding types.

For terms the variables, constants and function designators (in a particular case - expressions) are used. Each term has a well defined type. An expression denotes a function with arguments respresented by variables in the expression having types defined by an expression context. A result of a function and its type are implicitely defined by an expression.

An atom (atomic formula) appears as $p(t_1, ..., t_n)$, where p is a predicate and t_i are terms. Atomic formulae define simple statements concerning information resources.

In simple form a formula is an atom or appears as:

w_1 & w_2 (w_1 and w_2)
w_1 | w_2 (w_1 or w_2)
^w_2 (not w_2)
w_1 -> w_2 (if w_1 then w_2)
ex x/t (w) (for some x of type t, w)
all x/t (w) (for all x of type t, w)

where w, w_1 , w_2 are formulae.

Variables in formulae can be bounded by quantifiers. A scope of a quantifier in a formula with quantifier is a formula w. Occurences of variables denoted by a quantifier in the

scope of the quantifier are bounded and are free for their occurences out of the scope. A notation x/t defines that a type of a variable x is t.

Rules are represented as closed formulae of the form: $a \ :- \ w$ where a is an atom and w is a formula. a is a head and w is a body of a rule. All free variables of a and all free variables of w are assumed to be bounded by a universal quantifier placed before the rule. If w is absent and attributes of a have constant values then such rule is a fact.

If a formula has $x_1, ..., x_n$ as free variables then the formula results in a collection of substitutions of correctly typed values of $x_1, ..., x_n$ such that on each of them the formula interpreted on collections of value instances involved gets the true value. Formulae semantics are such that based on such substitutions a resulting collection is formed containing objects, ADT values or frames as its elements. In the sequel in cases when an exact nature of values and collections is not important, general terms "value" and "collections" are used (assuming that sets, bags, sequences, worlds and classes are specializations of collections and that objects and frames are special cases of ADT values).

The variables in formulae can be typed:

< typed variable >::=< variable > [/ < type expression >]

A syntax of typed variables provides for necessary modifications of denotations for establishing a correspondence of variables to class (type) attributes and for a proper denotation of resulting variables of subformulae to form a final result of a formula.

In formulae numbers and strings provide for representation of integer and real values, of range and enumeration type values, of bit and character strings. Constants of more complex types are represented by frames.

3.4.2 Interpretation of formulae

Collections of values of abstract data types constitute interpretation domains of formulae. A collection of values (e.g., collection of objects, collection of frames) corresponds to a predicate-collection (including a predicate-class and a predicate-context). Thus a predicate-class or a predicate-context are always unary. A collection type and a collection element type should be distinguished. Such unary predicate corresponds to a collection, an argument of the predicate takes as its values elements of the collection (ADT values) having a respective type of the collection elements or its reduct. Unary predicate with a type name assumes a set of admissible instances of this type.

Thus to an atom that appears as < predicate name >, or < predicate name > (< term >), where a predicate name defines a collection, a collection of ADT values corresponds that were created or derived (in case of views) up to the moment of the predicate evaluation. In case of a type predicate this is a set of admissible instances of the type: therefore, a type predicate is treated as a collection predicate in formulae. In case of a predicate-class all created (derived) objects in all direct and transitive subclasses of a given class are included. Such set is known as a *total* extent of a class. If it is required to use only *own* class extent that does not contain instances of its subclasses, it is necessary to mark predicate-classes in formulae by * placed before a name of a class. A singleton set corresponds to a predicate-resource represented by a single object.

For < term > a typed variable can be used to denote a collection element type. The variable can be typed by the collection element type itself, by its reduct, or by a reduct-like type whose intention is to introduce another type attribute names for their proper denotation in formula.

Arbitrary compositions of functions can be used in formulae (in this case a function designator is used as an actual parameter). Compositions of functions having a function designator for the first parameter play a specific role. Such compositions provide for construction of *path expressions* denoting a path in a compositional class (ADT) hierarchy.

For instance, let specifications of classes *family* and *person* appear as:

```
{Person;
 in:type;
 name: string;
 age: integer
}

{person;
 in: class;
 instance_section:
   {objtype: Person}
}

{family;
 in: class;
 instance_section:
   {name: string;
    mother: Person;
    father: Person;
    children: {set_of: Person}}
}
```

Let a variable f denotes an object of the class *family*. Then composition of functions *name(mother(f))* defines a mother name in a family f. Alongside with a bracketed notation more simple dot notation is allowed: *f.mother.name* having the same interpretation.

To form a general result of several subformulae related by &, |, &^ connectors ("and", "or","not" respectively) the following rules should be used.

A connector & denotes an intersection of collections corresponding to subformulae. A result contains a collection of ADT values that are contained in each of the initial collections. A type of elements of the resulting collection is a result of a *join* [14] of types of elements of the initial collections. A resulting join type is assumed to be defined.

A connector | denotes a union of collections. A result contains a collection of ADT values that are contained at least in one of resulting collections corresponding to initial subformulae. A type of elements of the resulting collection is a result of a *meet* [14] of types of the elements of the initial collection. A resulting meet type is assumed to be defined.

The type operations *meet* and *join* get an immediate common supertype (subtype) for the operand types.

A connector & ^ denotes a difference of collections corresponding to initial subformulae. A type of elements of a resulting collection coincides with a type of elements of the first collection. The resulting collection includes elements of the first collection that do not contain equal counterparts in the second one.

Formula with the existential quantifier produces a collection relevant to the quantified formula with type of its element substituted by a *reduct* [14] of the relevant collection type defined on attributes corresponding to the variables unbounded by the quantifier.

Interpretation of other formulae of the language is similar though a bit more complicated.

3.4.3 Frame predicates

Depending on the context of the formulae, frame predicate can be satisfied by a collection of frames or values of a product type having for its component types the types of the resulting variable list of the formula. If the resulting collection corresponding to the frame predicate is formed on the basis of the resulting variables then every value of the product type includes as its components (corresponding to the resulting variables) the values of frame components of relevant frames having the required types.

Subformulae in a formulae can be linked by symbols &, |, & ^, set theoretic interpretation of which depends on the resuts of the subformulae. Here we mention two frame predicates - functional and pattern.

Functional predicate gives an ability to construct path expressions traversing frame structure or combinations of object - frame (frame - object) structures. We have already mentioned that slots of frames are considered as functions $a : C \rightarrow V$, (where a is a slot name, C is a context or a world name, V is a collection of the slot values). A domain of such functions is a collection of frames in a context (or in a world). In case of slots interpreted as stored functions on using of the function designator *a(o)* an application of the generic functions get or set is implicitely assumed. These functions are extracting or modifying a value of the slot a of a frame or generally of an abstract value o. Thus, functional predicate is a logical representation of the result of an application of a function given by a slot. The resulting variable and its type are determined by the name and the type of the term corresponding to the resulting argument of the function. Compositions of functional predicates using dot notation give an ability to construct path expressions.

In formulae the typed variables can be used everywhere to define a type of a result of a formula combining classes and worlds as well as to set the identity of variables and slots. Admissible transformations of values are assumed in accordance with the types given. Variables are used in formulae to express the formula logic and to denote slots of the resulting frames.

A pattern predicate provides a set of frame patterns for a specification of relevant frames. Comprehensive variants of patterns can be specified.

4 Uniform representation of semistructured and hybrid data models in the canonical one

Object-oriented facilities of the canonical model has been checked to represent equivalently ODL of ODMG [13] though the SYNTHESIS model goes far beyond that. This reference shows how to represent structural aspects of semistructured data models in the canonical one.

4.1 Mapping of the ADM model into SYNTHESIS

Mapping of the ADM model into the SYNTHESIS model will be analyzed first. Similarly to ADM, a Web page can be seen in SYNTHESIS as an object if it can be structured. This object may have attributes (monovalued having, e.g., text, image or other abstract data types, or multivalued). ADM list types are mapped into the SYNTHESIS sequence types that can also be nested. The FORM type can be easily defined as ADT and heterogeneous union is the SYNTHESIS built-in type.

Constraints in SYNTHESIS are expressed as closed formulae of the object calculus. Besides that, there are built-in constraints to express simple assertions like *unique, obligatory, optional* and others.

LINK will be interpreted as an attribute belonging to specific attribute metatype LINK-TO providing necessary properties for an attribute (e.g., URL properties).

Due to the above, the mapping of the ADM model into SYNTHESIS is straightforward. Type definitions for the schema of the TitleIndexPage for DB and LP bibliography Web server will look as follows:

```
{TitleIndexPage;
 in: type;
 WorkList: {sequence; type_of_element: Author}};

{Author;
 in: type;
 Authors   : TEXT;
 Title     : TEXT;
 Reference : TEXT;
 Year      : TEXT;
 Pages     : TEXT;

 AuthorList: {sequence; type_of_element:
       {in: type;
        Name : TEXT;
        ToAuthorPage : AuthorPage;
        metaslot in: LINK-TO end }};

 ToRefPage : {union;  ConferencePage;
              JournalPage; SeriesPage;
              WebPage};
 metaslot in: LINK-TO, optional end
};
```

Mapping of the XML DTD into the SYNTHESIS canonical model is considered in [18].

4.2 Mapping of the YAT model into SYNTHESIS

YAT interpretation in SYNTHESIS is assumed as follows. Labeled trees (patterns) of YAT can be mapped into frames (objects).

References may be interpreted by the self slots (attributes) of frames (objects).

Definition of YAT classes (Figure 1) in SYNTHESIS looks like the following. First, the instance types should be declared:

```
{Artifact;
 in: type;
 name: string;
 year: integer;
 image: {set-of: char}}

{Artist;
 in: type;
 name: string;
 school: string;
 artifacts: {set_of: Artifact}}
```

Notice, that self is implicit attribute of any object. We assume that classes having Artifact and Artist as instance types are declared respectively as artifact and artist.

Sets of &Artist and &Artifact values (artists and artifacts in Figure 1) are defined in SYNTHESIS as

```
{artists; {set; type_of_element: Artist.self}};

{artifacts; {set; type_of_element: Artifact.self}};
```

We assume that these sets are instantiated with references to the instances of classes artist and artifact respectively (these sets may represent any subset of artist and artifact classes).

The biography type (Figure 2) and the respective set of biographies will be defined similarly:

```
{Biography ;
 in: type;
 person: string;
 born: string;
 died: string;
 bioprop: {set; type_of_element: Field}};

{Field; {union; string;
                {set; type_of_element: Field}}};

{biographies; {set; type_of_element: Biography}};
```

To create a set of modern artists it is assumed that the type Martist is declared as:

```
{Martist;
 in:type;
 name:string;
 born:string;
 died:string;
 school:string;
 bioprop: {set; type_of_element: Field}};
```

and a set of modern artists is defined as:

```
{martist; {set; type_of_element: Martist}};
```

Now, a transformation shown in the YAT-related subsection will be defined in SYNTHESIS as:

```
martist(self(N), N/name, B/born, D/died,
        S/school, bioprop) :-

ex UID/Artist.self(artists(UID) &
      artist(UID, N/name,S/school,
             artifacts(Na/name, Da/year))) &
biographies(P/person, B/born, D/died, bioprop) &
Da > 1800 & N = P
```

Here self(N) plays the role of a Skolem function - similarly to YAT.

4.3 Mapping of the Ozone model into SYNTHESIS

Before going into discussion of mapping of hybrid model language Ozone into SYNTHESIS it is worth of mentioning that SYNTHESIS itself has been developed as the hybrid one. Frames for unstructured and semistructured data and objects for the structured one provide the required facilities.

The frame language of SYNTHESIS can be considered as a self-describing semistructured data model, similar to OEM. Entities in OEM treated as objects with OIDs are easily interpreted as frames in SYNTHESIS. Multivalued attributes having $0-n$ occurences of an attribute value in OEM are easily interpreted: a slot of a frame is inherently multivalued. A slot value may be of any type admissible in SYNTHESIS (including the frame type). Therefore, similarly to OEM, a frame database may be viewed as a labeled directed graph, with frames as internal nodes and slots having atomic values as leaf nodes or complex values.

To work with semistructured data on the Web, each Web site page may be represented in SYNTHESIS by a frame embedding sub-frames representing the fragments of the page. An instance of a world is a collection of frames that can be reachable through hypertext-based frame relationships from the 'root' frame referenced by an URL. The frames involved can be represented differently (as frames with slots or without slots or combining both approaches).

Such world is interpreted as a labeled graph with nodes containing frames (that look as aggregates (or can be structured as those), as semistructured objects or as unstructured data) and arcs formed by hypertext links. In SYNTHESIS attributes of structured types can be typed with the frame type and frame slots can be instantiated with any SYNTHESIS type values (dynamic type capability).

After these general comments, it is easy to see how to map the Ozone data model into SYNTHESIS. Built-in type OEM is interpreted as the frame type. More precisely, the frame type corresponds to OEMcomplex. Any type of SYNTHESIS (meaning any built-in type) can be used to map OEMatomic type of Ozone. Frames are ordered collections of pairs and correspond to OEMcomplexlist type. Therefore, it is required to add in SYNTHESIS another frame type to deal with unordered frames: frame_set. This type will correspond to OEMcomplexset type.

Benefits of both models - object-oriented and frame-based - are available in SYNTHESIS. Treating frames as typed data allows queries to be written without complete knowledge of the schema. At the same time all object-oriented properties of the model (functions, subtyping, etc.) are available. Path expressions in SYNTHESIS may correspond to database path that may involve structured or semistructured data only, or there may be crossover points to navigate from structured to semi-structured data or vice-versa.

Mapping Ozone to SYNTHSESIS model is shown using the example on Figure 3. Specifications of Catalog, Company and Product types in SYNTHESIS follow.

```
{Catalog;
 in:type;
 vendors:{set_of: Company};
 products:{set_of: Product}};

{Company;
 in:type;
 name:string;
 address:string;
 produces: {sequence_of: Product};
 inventory: {in: function; params: {+x/Product.name,
             -stock_unit_numb/integer}}};

{Product;
 in:type;
 name:string;
 madeby:Company;
 prodnum:integer;
```

```
prodinfo:frame}};
```

We assume that the SYNTHESIS classes with instances of Catalog, Company, Product types are catalog, company, product respectively.

Representation of XML-like semistructured data for prodinfo attribute and for Reviews as frame data follow.

```
{specs: {wattage:100; weight:"16oz"};
 competing: 02/Product.self;
 competing: 03/Company.self;
 competing: {name: "system12";
             madeby: "Xylem Inc";
 compatible: 04/Product.self};

{Reviews;
 ConsumersInc: {subject:  01/Product.self;
                rating: {torque: 4.3;
                         efficiency: 6.5};
                category: "hand tools"};
 ConsumersInc: {subject: {name: "Amplifier";
                          prodnum: 200};
                rating: {power: 3.9;
                         distortion: 6.5};
                category: "electronics"};
 ABC_Consulting: ;
 ABC_Consulting: }
```

For example, the following query in OQL^s, Ozone query langusge, selects the names of all competing products and companies for all products in the broker catalog (Figure 3):

```
Select N
From Broker.products P, P.prodinfo.competing C,
     C.name N
```

P is statically known to be of type Product, but prodinfo is an OEM attribute, and C is therefore of type OEM; prodinfo is thus a crossover point from structured to semistructured data.

Analogous query in the SYNTHESIS languge is expressed using functional predicates as follows:

```
X="Broker" &
catalog(X/name).products.prodinfo.competing(N/name)
```

The following query has a semistructured to structured crossover since some of the bindings for C are OEM(Company) and OEM(Product) objects:

```
Select A
From Reviews.ConsumersInc R, R.subject C,
C.address A
```

Analogous query in the SYNTHESIS languge is expressed as follows (it is assumed here that reviews is a world containing Review frames):

```
reviews(X/Reviews) &
reviews(X).ConsumersInc.subject.C/Company.address
```

5 Conclusion

The paper provides an analysis how the SYNTHESIS model can be used as the canonical model to integrate various structured and semistructured Web-related data models. The technique for various data models integration in one paradigm has been considered in [10, 12]. This results in a uniform representation of very different forms of data used in digital library collections.

The idea of combination of structured and semistructured data has been probably firstly expressd in [9] and further elaborated in [10]. For object model this idea has been developed in [11].

References

[1] Abiteboul S. et al. Querying documents in object databases. International Journal on Digital Libraries, v.1, N 1, April 1997

[2] Abiteboul S. et al. The Lorel query language for semistructured data. International Journal on Digital Libraries, v.1, N 1, April 1997

[3] Arocena G., Mendelzon A. WebOQL: Restructuring documents, databases and Webs. In: Proceeedings of ICDE'98, February 1998, Orlando, Florida

[4] Atzeni P., Mecca G., Merialdo P. Semistructured and structured data in the Web: going back and forth. ACM Sigmod Record, N 1, 1998

[5] Atzeni P., Mecca G., Merialdo P. The Araneus Data Model (adm): a Logical Data Model for Web Sites. Technical Report, Universit'a degli Studi Roma Tre, T2-R01, July 30, 1998

[6] Christophides V., Cluet S., Simeon J. Semistructured and structured integration reconciled. http://www.rocq.inria.fr/verso/Jerome.Simeon/YAT/

[7] Florescu D., Levy A., Mendelzon A. Database techniques for the World-Wide Web: A survey ACM Sigmod Record, N 3, 1998

[8] A. Hopmann, et. al., Web Collections using XML, 1997 http://www.w3.org/pub/WWW/Member/9703/-XMLsubmit.html

[9] Kalinichenko L.A. Toward data description language for data base with partly determined schema. Proceedings of the IFIP Special Working Conference on Data Description Languages, Belgium, North-Holland Publishing Company, 1975

[10] Kalinichenko L.A. Methods and tools for heterogeneous data base integration. Moscow, Nauka, 1983

[11] Kalinichenko L.A. SYNTHESIS: the language for desription, design and programming of the heterogeneous interoperable information resource environment. Institute for Problems of Informatics, Russian Academy of Sciences, Moscow, 1993, 113 p.

[12] Kalinichenko L.A. Method for data models integration in the common paradigm. In *Proceedings of the First East European Workshop 'Advances in Databases and Information Systems'*, St. Petersburg, September 1997

[13] Kogalovsky M.R., Kalinichenko L.A. Refinement of the Synthesis language specification by the ODMG semantics. INTAS-94-1817 Project report, IPI RAS, Moscow, 1998

[14] Kalinichenko L.A. Compositional Specification Calculus for Information Systems Development. Proceedings of the East-West Symposium on Advances in Databases and Information Systems (ADBIS'99), Maribor, Slovenia, September 1999, Springer Verlag, LNCS, 1999

[15] Lahiri T., Abiteboul S., Widom J. Ozone: Integrating structured and semistructured data. http://www-db.stanford.edu/pub/papers/ozone.ps

[16] Mendelzon A., Mihaila G., Milo T. Querying the World Wide Web. International Journal on Digital Libraries, v.1, N 1, April 1997

[17] The Object Database Standard: ODMG 2.0. Ed. by R.G.G. Cattell, D.K. Barry, Morgan Kaufmann Publ., 1997

[18] Osipov M., Machulsky O., Kalinichenko L. XML data model mapping into the SYNTHESIS language object model. This Conference Proceedings.