ANNALES UMCS INFORMATICA DOI: 10.17951/AI.2016.16.2.7

Heterogeneous Indexing Register for Object Database

Michał Chromiak
Institute of Computer Science
Maria Curie-Skłodowska University
Lublin, Poland
e-mail: mchromiak@umcs.pl

***Abstract*—Even though the object oriented persistent stores has not gained large commercial adaptation rate, it still is an interesting research field in many aspects including the data integration. Persistent data integration is a very challenging goal in modern computer systems. This paper presents a proposal for application of effective indexing integration scheme for distributed and heterogeneous data environment using an object database as the central store.**

Keywords—database; data integration; data; object database

I. INTRODUCTION

The problem of integrating data form distributed data sources has forced the need for flexible and sound architecture that could cover all of the integration issues. While such an architecture could be implemented in numerous flavours and ways, currently the predominant programming paradigm is the object-oriented approach. While the object model is not dominating in the world of persistent data stores, there are some interesting prototypes that can be potentially interesting in aspect of applying them towards integration solutions. In case of central integration register based on idea of Qboid [1,2] such an object-oriented store seems a natural and effective choice. This paper discusses advantages of using the object store model (AS) originated form ODRA prototypical database for implementing central integrating register. It also presents a possible adaptation and exemplary implementation of the Qboid-based integration data model using object store model.

II. THE INTEGRATION

The need for integration of resources forces, along many others, a consideration of the heterogeneity problem. First the heterogeneous term must be explained for the needs and scope of this paper.

Let us assume we have an environment of multiple (remote) heterogeneous (many vendor origin) database servers. There are two different areas that need to be covered. The location transparency and the distribution details. In other words this can be referred to as an answer to questions “how the data can be obtained and where from ? ”.

While considering integration in such conditions there is a strong need for separation of global access interface from local implementation of a data source. The first – “how”- issue requires an approach that would make the access to piece of a data possible in a common way regardless of its location. This problem could be solved by building a broker mechanism that would cover all the particularities regarding requirements for remote access. A broker would hide the location of a resource and the location specific access method from a client. Such client request might be remote, out on the network somewhere, but it also might be local, in the same process as the calling client. Therefore this could be the solution to heterogeneous access for numerous local data models.

Second issue can be addressed by dint of a resource integrator. The integratior would have to be a storage area supplying each request within environment containing an unambiguous information about address of every piece of requested resource form within the environment.

By dint of the distribution details from the resource integrator the entire requested data portion can be assembled into one resource map and then accessed separately, respectively to their data model, thanks to the interfaces provided by the broker mechanism. A resource map would mean here a lookup table that “maps” an ID of the piece of information to a value representing it unambiguously. The model for this storage would have to cope with complex details and behaviors required by the nature of integration metadata. In the following section this kind of mechanism for object store has been proposed.

III. MOTIVATION FOR OBJECT MODEL

The integration always requires means to persist the integrated data or its metadata in some way. An object database seems a good choice. This is not only to its flexibility, but mainly due to lack of impedance mismatch issues. The object nature presented by the ODMG standard [3] for object databases or database-related Java technologies [4, 5, 6], despite significant role of object-oriented solutions in the remaining areas of software development, have not become greatly important in the industry. However, an interesting approach has been developed aside the general standards. The Stack Based Approach (SBA), has introduced existing object-oriented mechanisms (classes, encapsulation, inheritance, polymorphism, objects) for database programming. Moreover by applying the SBA, some additional mechanisms have been

introduced, like the dynamic object roles [8, 9] or interfaces on the database views [10, 11].

Regarding the complex nature of integration metadata, its retrieval and modification requires sound tools. The answer to this need is a query language. It is the second argument for utilizing the SBA, meaning its powerful query language extended to a programming language i.e. SBQL (Stack Based Query Language). As the most important feature of the ODRA (Object Database for Rapid Application development) prototype SBA implementation, SBQL alone makes it possible to create fully fledged database-oriented applications. In the case of such solution the development of database application tasks with just one, very high level language, can greatly improve programmers' efficiency and software stability along the development life cycle and supports complex queries.

Object itself is an abstract entity representing or describing some idea existing in a real world. Object is distinguishable from other objects with its unique name and distinct limits.

Important aspect of an object is that it do not assume a need for determining an attribute (or a set of attributes) that identifies the object in an unambiguous way (so called "primary key") as it takes place in case of relational model. The object has its identity – OID – unrelated to object's state, content nor other objects. However OID is unique across entire system.

Apart from unique OID object has name that is a handle to this object. The name of an object does not have to be unique (we can create multiple objects named Entity, Schema, Row, Cell etc.).

According to ODMG model, this property name is a collection of objects, where the name is the name of entire collection not just the single element. In the discussed model we will assume that each and every object has its name e.g. Row, but there will be possibility to group entire collection as one object named Schema.

In aspect of Qboid-based integration it is very important that object might constitute multiple states that depends on current values combination due to dynamic nature of integrated sources which change their state continuously. Those changes must be reflected in the integration architecture in a sound, robust and elastic manner. It is required for the integration perspective to be an up-to-date view of the integrated resource grid.

Every object has state represented as a combination of its components, mainly values of all attributes and references to other objects. The state of an object can change in time. In our approach to represent the integration characteristics and nature of dynamically changed environment, we will discuss below some of the attributes that can be utilized to make the standards describing each course fit the store model.

- Atomic attribute: such as the integration pattern symbolic name. It includes exactly one value, which is indivisible from the point of view of a user byt states a role of e.g. unique discriminator for integration model entities
- Complex attribute: such as each record/row. It includes many atomic values. It has hierarchic structure where each branch of the hierarchy has its name (e.g. data source address, database name id, schema of origin)
- Pointer attribute: contains a value pointing to the adequate OID of referenced object which in integration domain can be a replica of an object or a different fragmentation pattern, still representing the same information value of the pointing object
- Repetitive attribute: it include a variable in time number of values. Those values can be of atomic, complex or pointer type (e.g. list of replicas that state the same semantic value)
- Optional attribute: in a particular instance of an object it can have a value but it is not mandatory this is when a potential replica for e.g. record can be replicated but the replication is not required along the integrated grid
- Derivative attribute: value that derives from other attributes; such as back-referencing, when a object refers back to the object that points it but is on the other hand, higher In object hierarchy
- Class attribute: value that is common to a set of objects belonging to the same class e.g. representing the same data schema

This list is not complete. However the rule of object relativity mentions that every object can be composed out of unlimited number of sub-objects. This way every attribute is an object. Moreover, each attribute has its type. Therefore, the combination of attributes' types is the type of an object

A. Abstract Store Model

To introduce the integration data information to object-oriented manner store, working with the SBA prototype implementation there is a need for adopting at least the simplest store model i.e. AS0 [5]. In contrast to relational model, object model require to use far more concepts. There is also different understanding for many terms. Therefore, it is hard to introduce a model that can be simple and at the same time applicable for all cases equally. The SBA includes the whole hierarchical family of store models each responsible for extending the possibilities of the predecessor but all basing on the same semantic base. For the purpose of this paper we mention only the most basic but sufficient for this appliance model – AS0.

The AS0 can cover arbitrarily connected hierarchical data structures. However, it does not include the aspects of the class, inheritance or interface. It was originally designed to express the semantics of relational query languages. What is the essential part of it, is the possibility of representing semistructural data in general and the XML data structures in particular.

Regarding this store model we will assume the object relativity rule and related to it rule for the inner identification. First one has already been mentioned in previous section. Second rule states that every object that can be a component of different object has to include its own unique inner identifier.

Let us explain some basic terms:

- *Inner identifier of an object*. It is given automatically by the system and cannot be used in the semantics of the outer manipulation of the objects. Its purpose is to identify objects stored in memory.
- *Outer object name*. In contrast to inner identifier this name is created by the system designer, administrator or a programmer. It is linked to conceptual model of The application working with AS0 based store. Moreover, it involves the use of informal semantics for the outer processes e.g. The name can be *Row* or *Tuple*. The outer object name such as *Row* does not have to be unique.
- *Atomic value*. It is a kind of object value that is Indivisible from the point of view of the creator Hence not including any parts.

Identifiers are marked *as i*, the names as *n* and the atomic values with the letter *v*.

In AS0 every object contains unique inner identifier, outer name and the value that can be atomic, pointer or complex. We will the objects by the following definition. The object is a triple:

- $<i, n, v>$ - when the object is going to be atomic
- $<i_1, n, i_2>$ - when the object is going to be a pointer or a reference object. This object is identified thanks to i_1 where the i_2 is the pointer value of the object being a reference to other object.
- $<i, n, T>$ where the *T* is a set of any type of objects. This object we will call a complex object. This rule is recursive, therefore enabling building objects with unlimited complexity and number of hierarchy levels

In AS0 the data store is defined as pair <S, R> where the S is a set of objects and the R is a set of object ids also named as the staring identifiers.

The R set sets the starting points for the data store i.e. those objects that can be a starting point for the navigation in the entire set of objects. Most often those objects would be just simply the ones that are in the main level of the object hierarchy i.e. those that are not included as part of the other objects

There are few rules regarding the data store that has to be complied:

- each and every object, sub-object, etc. in data store has its unique identifier
- if there is a pointer object $<i_1, n, i_2>$, then the pointed object i_2 has to exist.
- each and every identifier from the R set is an identifier of some object located in store

IV. The Architecture

Integration is a very complex and multilevel challenge. It requires an effective and elastic approach that must conform some unified workflows and strict rules. Thus, mechanism needed for managing this infrastructure needs a creation of a dedicated architecture.

There are a couple of issues that has been considered to satisfy such a challenging requirements to provide the solution of the problem.

- Server-based integration, is something that would involve centralized management, based on some kind of broker, and at the same time an integrator, while dedicating the server for the purpose of routing requests from clients to data resource
- The initial integration scheme must be applicable and elastic to fit more than one dedicated data source server
- The central instance of integration should be able to become decomposed into a multi-node infrastructure, possibly a cloud, or a microservice based central instance
- The architecture involving integrator and broker in each location node though seems the most challenging since the reduction of flaws regarding the centralized or only partially distributed environment (e.g. low fault tolerance and traffic overhead). In this case the architecture would have to be multiclient/multi-server like, so that each machine could be a client and a server relatively to the status of a request i.e. sending or receiving.

While focusing on database area, the problem can still be considered valid regarding integration of BigData unstructured sources. The goal is to enable easy access to such a heterogeneous environment's data from within ODRA-based integration server using its object-oriented query language – SBQL- indistinguishably of the data model and location. Therefore, let us start from centralized, ODRA side management of the distributed databases.

A. The solution

At first, let us presume, we have a simple communication scheme i.e. an ODRA server and one legacy database to represent its data in integration view. We have to face the problem of data model of a legacy DB. This problem can be handled by object-relational wrapper. Nevertheless, while the number of legacy databases increases, the communication scheme becomes more complex. Therefore, there is a need for a integration mechanism. In this case, the ODRA is assumed to be a client i.e. a process that makes calls to objects located on a remote, legacy DB server or within the ODRA client.

B. General Idea

In this section the general architecture components and their role is explained and motivated. Let us introduce the basic facilities utilized in distributed, heterogeneous environment (see Fig.1.):

- **Object Location Integrator (OLI)** – is a component that is responsible for collecting and storing information about data fragmentation and replication across the integrated legacy data sources. Moreover, it enables access to the Broker and would also be responsible for storing unified index representation
- **Broker** - facility storing the fast, native access methods for each grid integrated data source objects
- **Client** – is the party sending requests to OLI for integrated data entities form within the index present in OLI. The part of the client responsible for sending those requests would be a module compatible with OLI.
    - In this case the client could be any human or software party calling the integration REST API available at ODRA-based integration facility. However, any DBMS could be plugged in the OLI as long as they contain dedicated compatibility module. As the OLI would store the universal index representation, dedicated module would have to be responsible for transforming this representation into a native ODRA index. One should be aware that ODRA is an example of an object-oriented database which on specification change can freely be swapped with other database engine with different paradigm. Obviously in such case, the new paradigm particularities must be considered for a well designed index to work.
    - The transformation into ODRA index would have to face two problems:
        - how to transform the structure of the universal index into the native clientmanageable form (utilize the native ODRA index structure)
        - provide the facility for interpreting and sending the grid integrated legacy DB access methods for reaching the specific data source objects and then receiving the results. This results would be composed into native client index form, according to the index scheme, build out of the universal OLI index.
- **Legacy data source** – the grid node providing partial data for the global OLI data integration scheme. It would have to include the data source specific access wrapper and the mediator capable of maintain communication between the grid node and the OLI.
    - **Wrapper** – data source dedicated software process combining the legacy data model and interface of the integrated data source to the mediator level of integration
    - **Mediator** - responsible for integrating multiple data source wrappers per each machine. Mediators would be responsible for sending the registration information of the underlying data sources, monitoring their up/down state. Moreover, the mediator take part in passing client requests for particular data records from each of its underlying data sources. Those requests would be partial client requests for distributed and indexed data

What requires explanation is the description of this process along its lifecycle:

1. OLI/Broker initialization - could be considered as ODRA heterogeneous index (H-Index) module(s) or as standalone processes.
2. Each network data source that is to be integrated into grid, continues with the process of registration:
    a. Each machine needs to start the mediator infrastructure i.e. equip the Mediator with the underlying data schemata and its fast access methods.
    b. (Fig. 1. pos.1) In the beginning, establishing connection between OLI and Mediator takes place. Next the Mediator sends the underlying data sources schemes and fast access methods (FAMs) for accessing each data scheme part.
    c. (Fig. 1. pos.2) The received schemes are stored at OLI and the database object reference (DORs) (including the native fast access method – FAM) are moved to Broker which is treated as a DOR1[1] store. OLI would store only DOR reference to broker named rDOR.
3. (Fig. 1 pos. 3-5.) Along the registration, at a time when a data entity (e.g. DB table) in OLI occurs to be a complete snapshot of its present state in the grid2[2] (i.e. all its records has been registered within the OLI), the indexing towards this snapshot can be evaluated.

[1] Database Object Reference a structure for introducing distributed data access method

[2] This information is available thanks to administrative configuration (horizontal fragmentation; administrator could point the grid nodes to participate in the data entity horizontal integration) and data definition scheme (vertical fragmentation) present in the Qboids. When vertical fragmentation matches the data definitions and all the of the pointed nodes are involved then the index creation can be conducted.

4. The index creation selects the right parts of the integrated data scheme i.e. those that carries the index information and forms table consisting of the indexed values, their unique records' ids (i.e. best record ids – BRIs) and access methods. Transformation of such sequence of triplets, yields the list of records with their access methods combined with BRIs, grouped together per each indexed value/range. As already mentioned in 2c) OLI would have store only rDORs. However, for the purpose of forming universal index and reaching for actual values, they would have to be replaced by DORs (storing the detailed data) from the broker. In this phase we acquire an universal index structure for a client requested type of index i.e. dense, range etc.
5. (Fig. 1. pos.6.) At this stage, the clients' heterogeneous index (or, H-Index) module can use the universal index to incorporate its information into native index. The process however, is not over yet, because the client will only possess the information about the index's grid details and DORs, but it will still not include the indexed records explicitly (pos.7.).

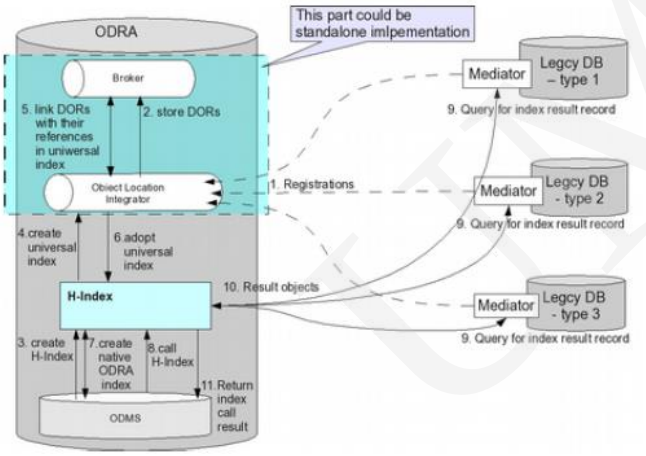

Fig. 1 General schema of heterogeneous index creation and utilization

Storing the grid details of the data distribution in the native index will force the native index, while reaching for a record, to call for a procedure that can return the explicit values presented in such record (pos. 8.). Therefore, before making the native index available for use in native client requests, all of the DORs found in such index would have to be send to a remote process able to transform each record and hand it over to native index in form of a complete and explicit database record, that can be further utilized by regular index mechanisms. This is the responsibility of the H-Index module. It would send (pos. 9.) the access queries to the appropriate grid nodes mediators. Next, according to the possessed implicit index structure scheme, the received results (pos. 10.) would have been utilized to build each record. In case of ODRA it would have been a regular ODRA database object. Fig.1. General schema of heterogeneous index creation and utilization. This object can be incorporated into ODRA index as a part of the ultimate native index, composed out of the H-Index received results.

V. Objects For Integration

The general goal of integration must conform some way of unification for integrated resources' data. To use the AS0 model for the purpose of storing integration data let us adopt the schema proposed in [2] for AS0 model.

A. Distributed Data Structure Map

Each legacy data source intuition and general schema would have to be devised. In [2] the distributed resource universal map has been introduced to represent three basic issues of integration – namely replication, vertical and horizontal fragmentation. Basing on some technical best row id (BRI) each record that is to be considered information equivalent in terms of semantic meaning along integration view assumptions shares the same BRI.

Conceptually each data source might get its own representation of record while still sharing BRI. This way one BRI might reflect multiple replications, and as the record might be arbitrary composed, also vertical fragmentations. The horizontal fragmentation in considered in record groups as the higher orders of composition within the Qboid concept.

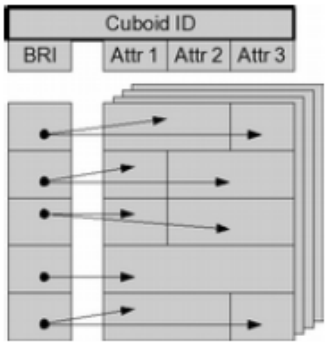

Fig. 2 Qboid build based on BRI matching

B. Example

Let us assume that the object-oriented model AS0 is going to store information on users. The general schema contains name, email address and age. Now this information is expected to be scattered across multiple data sources. In Listing 1. an exemplary schema for integrating such information is being exemplified:

```
<e1, entity, {
  <u1, users, {
    <recGr1, recGr, rDOR_1to10_DB1 > # ID_1to10
    <recGr2, recGr, { # ID_13
      <n13, name, rDOR_DB2_13_name>
      <em13, emailAddress, {
        <l13, login, rDOR_DB20_13_login>
        <d13, domain, rDOR_DB21_13_domain>
```

```
            }
            >
            <a13, age, rDOR_DB2_13_age>
            }
         >
            }
      >
      <u2, users, {
         <recGr3, recGr, rDOR_1to10_DB100 >
         <recGr4, recGr, recGr2 >
         }
      >
      <u3, users, {
         <recGr3, recGr, {recGr1, recGr2 } >
         }
      >
      <u4, users, recGr2>
         }
>
```

Listing 1. Exemplary AS0 utilization to represent the integrated data

We can observe here the exemplary schema being configured according to the system integrating specification. The general top level object, according to AS0, is representing the entity. Entity is a general purpose instance in integration scheme that represents conceptually equivalent term as a table might be. However, entity is also responsible for storing potential replicas and mixed fragmentation patterns of the integration perspective. In the presenting Listing1. The entity represents users as an domain entity that covers thirteen users. The entity is designed as a complex object with id, name and a set of objects representing entity, which in this case are users. Each user is also represented by a complex object with its unique id, name and list of complete user characteristics. In case of u1 object it is responsible for representing users form mixed fragmentation pattern; one horizontal, and one vertical fragmentation pattern. The first – horizontal – pattern is represented by *recGr1* and *recGr2* object. The *recGr1* object is an atomic object storing only the requested rDOR for ten users stored at the data source DB1, while the *recGr2* object is a complex object representing additionally vertical fragmentation within the *emailAddress* complex object for use with id 13. Even though the *recGr2* represent a single entity record, it is still required to bound the vertical but also horizontal fragmentation of the specific user data. For user with id 13 one can easily find the horizontal fragmentation due to its name and email address being stored in a completely different data sources (DB2 and DB20, DB21). Additionally the email address itself must be bind with use of vertical fragmentation pattern, which in this case states that the user login is stored at DB20 while the adequate domain must be reached form data source referenced as DB21. Additionally the age attribute as a third field of the recGr2 complex object is also stored at the same database as its name, i.e. DB2.

What is more, the user pattern is not the only integration challenge that need to be faced. Along the integration process one has to be aware of the replications that can occur on multiple integrated sources regardless of their physical independence. For instance one can easily imagine that the same company's employee is present in HR database, IT department database and the JIRA database. While still being the same employee for the company the context of the data source is completely different in this case. Therefore. The additional entity instances are represented along the Listing1.

The u2 and u3 represent the same, or almost the same set of information on the users. However, the nature of replication is somewhat different. While in case of u2 object it is a complex object representing the same data as u1, however the u2 has different data source (DB100) for users with id 1-10 while the recGr2 becomes a reference object (*recGr4*) pointing to the recGr2, meaning that there is the same algorithm to combine its content. This design enable future proofing towards enabling future load balancing while accessing users with id 1-10 which in this case can be retrieved from two different sources automatically and transparently.

The object *u3* is a complex object with reference types towards *u1* components. The *u3* object here provides the *u1* functionality and at the same time without replicating nor disclosing the *u1* details.

On the other hand the case of *u4* is quite different. In this case we can see that while it enables referring to the users, however it provides only data only for one user with id 13. Additionally the details of this user access methods are not disclosed towards the requesting party. This specific behavior provides two benefits. Firstly the entity designer might decide to disclose only the *u4* limited user database access, secondly due to reference object the contact details for the actual object are not disclosed and are only available for the party with privileges sufficient to disclose the *recGr2* user details originating in *u1*.

VI. Conclusions

The goal for this paper was to enable a prototype object oriented integration scheme based on AS0 object model originating from prototypical object-oriented database. It has been proven that even relatively simple object model can be adapted and successfully used for representing integration metadata for Qboid based integration architecture. However, one can easily proceed with extending the AS0, which covers relational, nested-relational and XML-oriented databases. AS0 assumes hierarchical objects with no limitations concerning the nesting of objects and collections, pointer links (relationships) between objects. Moreover, in case the AS0 if model is to be considered insufficient additional research can be done to provide all of the goodness of the complete objectoriented

model including classes and static inheritance, object roles and dynamic inheritance, or encapsulation. The SBA assumes just right store models in form of AS1, AS2 and AS3 to provide this complete set of object related features. Additionally, such an extension would give not only all of the possibilities of a object oriented approach, but also gains that a database engine provide towards data storing, such as persistence, durability, high availability, reliability with transactions.

REFERENCES

[1] Michal Chromiak, Piotr Wisniewski i Krzysztof Stencel. "A Universal Cuboid-Based Integration Architecture for Polyglotic Querying of Heterogeneous Datasources". W: Beyond Databases, Architectures and Structures - 11th International Conference, BDAS 2015, Ustroń, Poland, May 26-29, 2015, Proceedings. 2015, s. 170–179. doi: 10.1007/978-3- 319-18422-7_15

[2] Michal Chromiak i Krzysztof Stencel. "A Data Model for Heterogeneous Data Integration Architecture". W: Beyond Databases, Architectures, and Structures - 10th International Conference, BDAS 2014, Ustron, Poland, May 27-30, 2014. Proceedings. 2014, s. 547–556. doi:10.1007/978-3-319-06932-6_53.

[3] Cattell R. G. G., Barry D. K., The Object Data Standard: ODMG 3.0. Morgan

[4] Cook W. R., Rosenberger C., Native Queries for Persistent Objects A Design White Paper (2006); http://www.db4o.com/about/productinformation/whitepapers/Native %20Queries%20Whitepaper.pdf

[5] Lentner M., Subieta K., ODRA: A Next Generation Object-Oriented Environment for Rapid Database Application Development Advances in Databases and Information Systems, 11th East European Conference, ADBIS 2007, September 29-October 3, 2007, Proceedings., LNCS 4690, Springer, ISBN 978-3-540-75184-7 (2007): 130

[6] Hibernate - Relational Persistence for Java and .NET. http://www.hibernate.org/ (2006).

[7] Subieta K., Theory and Construction of Object-Oriented Query Languages. PJIIT – Publishing House, ISBN 83-89244-28-4 (2004), 522 pages (in Polish).

[8] Albano A., Bergamini R., Ghelli G, Orsini R., An Object Data Model with Roles. Proc. VLDB Conf. (1993): 39.

[9] Jodlowski A., Habela P., Plodzien J., Subieta K., Objects and Roles in the Stack-Based Approach.Proc. DEXA Conf., Springer LNCS 2453 (2002).

[10] Kozankiewicz H., Updateable Object Views. PhD Thesis (2005);

[11] Kozankiewicz H., Leszczylowski J., Subieta K., Updateable XML Views. Proc. of ADBIS'03, Springer LNCS 2798 (2003): 385.