Autonomous Layer for Data Integration in a Virtual Repository*,**

Kamil Kuliberda[1,4], Radoslaw Adamus[1,4], Jacek Wislicki[1,4], Krzysztof Kaczmarski[2,4], Tomasz Kowalski[1,4], and Kazimierz Subieta[1,3,4]

[1] Technical University of Lodz, Lodz, Poland
[2] Warsaw University of Technology, Warsaw, Poland
[3] Institute of Computer Science PAS, Warsaw, Poland
[4] Polish-Japanese Institute of Information Technology, Warsaw, Poland
{kamil, radamus, jacenty, tkowals}@kis.p.lodz.pl,
kaczmars@mini.pw.edu.pl,
subieta@pjwstk.edu.pl

Abstract. The paper describes a self-operating integration mechanism for virtual repository's distributed resources based on object-oriented databases in a grid architecture. The core architecture is based on the SBA theory and its virtual updateable views. Our virtual repository transparently processes heterogeneous data producing conceptually and semantically coherent results. Our integration apparatus is explained by two examples with different types of fragmentations – horizontal and vertical. A transparent integration process exploits a global index mechanism and an independent virtual P2P network for a communication between distributed databases. Researches presented in the paper are based on a prototype integrator which is currently under development.

1 Introduction

Recently, a term *virtual repository* becomes increasingly popular in database environments as a mean to achieve many forms of transparent access to distributed resources. A virtual repository user is not aware of an actual data form as he or she gets only needed information and the best shaped for a particular use. Among many other new concepts in modern databases this branch is evolving and developing very quickly as a definite answer from science to business needs. We already know many potential applications of a dynamic data integration technologies like e-Government, e-University or e-Hospital. In a modern society data must be accessible from anywhere, at any time, regardless of a place is stored in and other similar aspects. There are many approaches which try to realize this idealistic image. Some involve a semantic data description and an ontology usage extended by logic-based programs that try to understand users needs, collect data and transform it to a desired form (RDF, RDFQL, OWL). Other commercial systems like Oracle-10G offer a flexible execution of distributed queries but they are still limited by a data model and

* This work is supported by European Commission under the 6th FP project e-Gov Bus, IST-4-026727-ST.

** This work has been supported by European Social Fund and Polish State in the frame of "Mechanizm WIDDOK" programme (contract number Z/2.10/II/2.6/04/05/U/2/06).

R. Meersman, Z. Tari et al. (Eds.): OTM 2006, LNCS 4276, pp. 1290 – 1304, 2006.

languages not sufficient for distributed queries, in which programming suffers from inflexibility, complexity and many unpredictable complications. Our novel system offers all necessary features but it keeps programming very simple. The main idea is focused on P2P networks as a model for connecting clients and repositories combined with a powerful viewing system [2] enabling users to see data exactly in a required form [8]. This viewing system also allows an easy-to-operate mechanism for managing various data fragmentation forms as a consistent virtual whole.

The rest of the paper is organized as follows. Section 2 presents the idea of a data grid based on a virtual repository. Section 3 presents a virtual network for a data grid. Section 4 demonstrates the implementation and some examples. Section 5 concludes.

2 Distributed Data in Virtual Repository

The main difficulty of the described virtual repository concept is that neither data nor services can be copied, replicated and maintained in the global schema, as they are supplied, stored, processed and maintained on their autonomous sites [6, 7]. What is even more important, resources should be easily pluggable into the system as well as users can appear and disappear unexpectedly. Such a system by similarity to an electric grid was called a grid database or a data grid [1, 9].

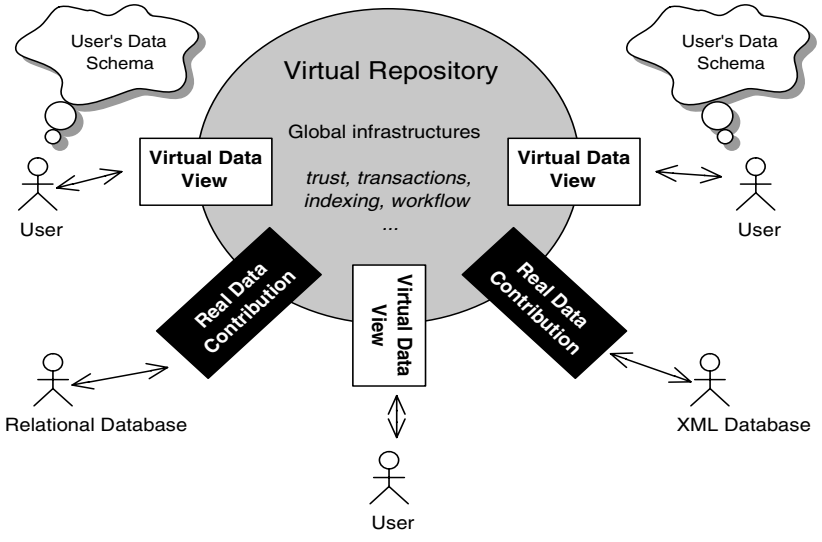

Fig. 1. The concept of a virtual repository. Users work with their own and favorite view of resources not knowing the real data or services.

However, a grid database has some limitations that significantly distinguish it from an electrical grid:

- a user may have many providers of the same service;
- a service provider can be not the same as a connection provider;
- a trust policy is more extended and more complex.

Because of these reasons a user must exactly describe his or her needs in terms of business contracts and a service provider must be able to evaluate possibilities of fulfilling these demands. Figure 1 shows the general concept. A user may plug into a virtual repository and use resources according to his or her needs, assigned privileges and their availability. In the same way resource providers may plug in and offer data or services. We may say that Internet and World Wide Web accompanied with HTML, search engines and web services work according to that idea. But we also know that searching for information, whose results are stable and repeatable, is very difficult. This environment is not sufficient for databases and effective programming tasks. A virtual repository must be achieved by an additional layer of a middleware which will supply a full transparency of resources, providers and clients [8]. The goals of the approach are to design a platform where all clients and providers are able to access multiple distributed resources without any complications concerning data maintenance and to build a global schema for the accessible data and services.

Currently, our team is working on a data grid solution developed under the international eGov-Bus project (contract no. FP6-IST-4-026727-STP). The project objective is to research, design and develop technology innovations which will create and support a software environment providing user-friendly, advanced interfaces supporting "life events" of citizen or enterprises – administration interactions transparently involving many government organizations within the European Community [16]. It can be achieved only if all existing government and para-government database resources (heterogeneous and redundant) are accessible as a homogeneous data grid. We aim to integrate existing data stores (a cost of software unification and data migration would be extremely high and a time of such an operation unacceptably long) represented by various data models and stores (e.g. relational, object-oriented, semistructured ones).

2.1 Resources Description

We separate two different data schemata. The first one is a description of resources acceptable by a virtual repository and it is called a *contributory schema.* A virtual repository must not accept *any* data but only data that it "knows" how to integrate. Another reason for limitations in this area is a consortium, which is establishing a virtual repository. It also has certain business goals and cannot accept *any* data from *any* provider. We assume that a virtual repository is created for a certain use and certain institutions [3, 5].

The other description is called a *grid schema* or a *user schema.* It describes data consumed and services used by clients. The task of a virtual repository system is to transform data from a contributory schema into a user schema performing also necessary integration and homogenization. This task may be done with different tools based on data semantics, business analysis performed by experts or any other programming techniques. In our solution we propose a view-based system with a very powerful programming and query language. The views are able to perform any data transformation and they support updates without any limitation common in other systems. We emphasise that it is the best solution for transparent data integration not possessing drawbacks of other systems.

2.2 System Architecture

The global infrastructure is responsible for managing our grid contents through access permissions, discovering data and resources, controlling location of resources, indexing whole grid attributes. The realization challenge is a method of combining and enabling free bidirectional processing of contents of local clients and resource providers participating in the global virtual store [8].

Each resource provider possesses a view which transforms its local share into an acceptable contribution. It is called *a contributory view*. It connects to the repository and may immediately share data. Providers may also use extended wrappers to existing DBMS systems [6, 7].

Similarly, a client uses *a grid view* to consume needed resources in a form acceptable for his or her applications. This view is performing the main task of data transformation and its designer must be aware of data fragmentation, replication, redundancies, etc. [3, 5] This transformation may be described by an *integration schema* prepared by business experts or being a result of an automatic semantic-based analysis. Nevertheless, we insist that in real business cases a human interference is always necessary at this stage. The problem is how to allow transparent plugging in new resources and incorporate them into existing and working views. This question is discussed in the next section.

3 Virtual Network for Distributed Resources

The idea of a database communication in a grid architecture relies on unbound data processing between all database engines plugged into a *virtual repository*. Our approach depicts not only an architecture of the network with its features, but also additional mechanisms to ensure effortless:

- users joining,
- transparent integration of resources,
- trust infrastructure for contributing participants.

The general architecture of a virtual network concept solves above three issues though middleware platform mechanisms designed for an easy and scalable integration of a community of database users. The middleware platform creates an abstraction method for a communication in a grid community. The solution creates an unique and simple database grid, processed in a parallel peer-to-peer architecture [8].

The basic concept of a transport platform is based on a well known *peer-to-peer* (P2P) architecture. Our investigations concerning distributed and parallel systems like Edutella [10], OGSA [11] bring a conclusion that a database grid should be also independent from TCP/IP stack limitations, e.g. firewalls, NAT systems and encapsulated private corporate restrictions. The network processes (such as an access to the resources, joining and leaving the grid) should be transparent for the participants. Because grid networks (also computational grids) operate on a parallel and distributed architecture, our crucial principle is to design a self-contained virtual network with P2P elements.

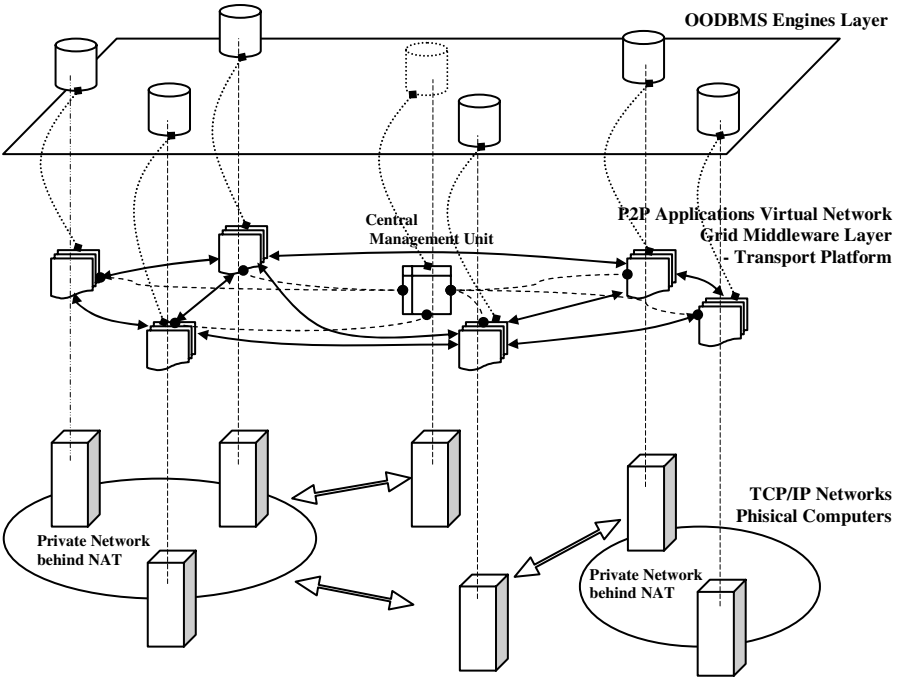

Fig. 2. Data grid communication layers and their dependencies

User's grid interfaces – in this proposition database engines – are placed over the (P2P) network middleware. For users DBMSs work as heterogeneous data stores, but in fact they are transparently integrated in the virtual repository. Users can process their own local data schemata and also use business information from the global schema available for all contributors. This part of a data grid activity is implemented above user applications available through database engines and SBQL query language [12, 13], see *OODBMS Engines Layer* in Figure 2.

In such an architecture, databases connected to the virtual network and peer applications arrange unique parallel communication between physical computers for an unlimited business information exchange.

3.1 P2P Transport Layer

The transport platform is based on JXTA Project [14, 15] as a core for a virtual network. It implements a part of P2P mechanism and provides the complete solution of a centralized and a decentralized P2P network structure. The virtual network has a centralized model and supports a management module for network mechanisms and grid processes (also a trust infrastructure). The transport platform is presented in Figure 2 as *P2P applications virtual network*. The P2P virtual network solution is a layer built on top of TCP/IP network. Its goal is to separate TCP/IP networks and make them transparent and independent of the grid community. The P2P virtual network has a complex structure containing two main blocks: JXTA Core and JXTA

Services with six additional virtual network protocols. The technical reference on how to build a P2P network using open JXTA platform is well explained in [15].

Peer-to-peer transport platform applications operate on a TCP/IP stack integrated with every operating system environment. A JXTA application implementation utilizes the Java Framework, so the peer applications of a virtual network can be used on any operating system supporting Java technology.

3.2 Central Management Unit

Our virtual network has a centralized architecture whose crucial element is a *central management unit* (CMU) – see Figure 2. In the virtual network there can exist only one CMU peer, its basic function is a responsibility for lifetime of data grid, besides it manages the virtual repository integrity and resource accessibility. Inside the P2P network level CMU is responsible for creating and managing the grid network – it means that CMU creates a *peer group* which is dedicated to linking data grid participants. CMU also maintains this peer group.

The peer group specifies the basic features of a trust infrastructure model for a database interconnection. This part of a trust infrastructure is located in the lower level (the virtual network layer) referring to the OODBMS Engines Layer rather than a direct user interface. In this case a realization consists of binding of a networking policy between an OODBMS engine and a peer application. In the current implementation we are focusing on credentials for a contribution of privileged databases in the data grid through indexing their unique identifiers inside a CMU's index. Each peer connected to the virtual network is indexed in CMU (see chapter 4).

3.3 Communication Peers and DB Engines – Implementation Details

For regular grid contributors the virtual network is equipped with participant's communication peers. They are interfaces to the virtual repository for OODBMS user's engines. Each database has its unique name in local and global schemata which is bound with the peer unique name in the virtual network. If a current database IDs are stored in the CMU, a user can cooperate with the peer group and process information in the virtual repository (according to the trust infrastructure) through a database engine with a transparent peer application. Peer unique identifiers are a part of P2P implementation of JXTA platform [14, 15].

A peer contains a separate embedded protocol for a local communication with an OODBMS engine and separate ones for cooperating with an applicable JXTA mechanism and the whole virtual network. All exceptions concerning a local database operation, a virtual network availability and a TCP/IP network state are handled by a peer application responsible for a local part of a grid maintenance. Please notice that in one local environment there are residing two separate applications (a P2P virtual network application and a database engine) composing (in grid aspects) one logical application [8].

Figure 3 presents the logical interoperability of grid components concerning the OODBMS layer and the virtual network layer. Communication can be established between:

1. Applications in the virtual network. It concerns the virtual network management and the virtual network processing (JXTA-XML datagrams). Connections in this layer rely on the protocol provided by the JXTA platform. For a communication inside the virtual network we use the multithreaded JXTA Socket layer and XML formulated datagrams for an information flow. We use different datagrams to exchange information in the P2P network and to manage peers.
2. A database implemented under ODRA (our prototype OODBMS) and a virtual network peer application used by internal XML-based protocol instances (see Figure 3). A connection between these applications implements a protocol, which exploits the JXTA communication stub and the TCP/IP socket implementation in .NET C#. Because the OODBMS prototype engine is currently implemented in .NET C#, it has forced a creation of a bridge protocol between JXTA and C#. The approach based on the bridge brings additional benefits, such as flexibility and openness for new functionalities and resource kinds, in particular, other DBMS-s, other P2P networks, other programming languages, etc.

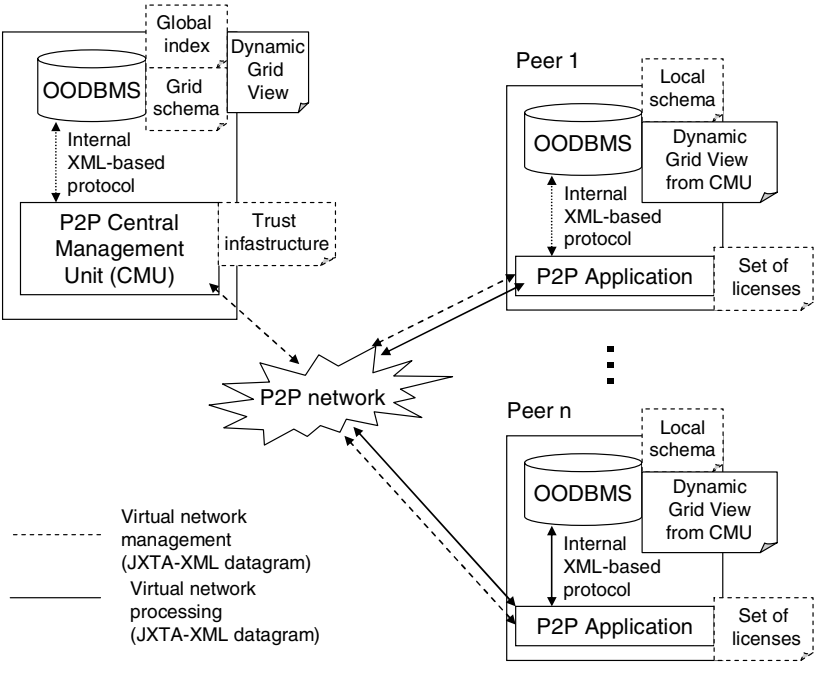

Fig. 3. Logical architecture of virtual network

All user operations on the virtual repository concern not only a local part of data stores, but also remote resources. User requests are packaged into XML-based protocol messages (equipped with some additional information about a source, a destination, etc.). The packages are sent through an XML-based protocol from a DBMS to an appropriate peer application. After receiving a XML datagram by a suitable peer application it is again wrapped according to the JXTA protocols into

JXTA-XML datagram. Such datagrams should be sent from a source peer to its destination peer application in the virtual network. A destination peer is responsible for decomposing a datagram and sending appropriate requests to a target DBMS. Notice that XML datagrams which come from a database engine have as source and destination attributes unique identifiers of participating databases. In the virtual network each database has an additional identifier as its native peer name. The current database is recognized in the virtual network by a string like *DB_name@Peer_name*.

4 Transparent Integration Via Viewing System and Global Index

The presented virtual repository based on a P2P networking must be permanently updated and this is the most important aspect of our virtual repository operation. For an easy management of the virtual repository's content we have equipped the CMU with a *global index* mechanism which covers technical networking details, management activities and it is also a tool for efficient programming in a dynamically changing environment. The global index not necessarily has to be really centralized as there are many ways to distribute its tasks. The system needs this kind of an additional control and it does not matter how it is organized. Its tasks are:

- controlling grid views available for users,
- keeping information on connected peers,
- keeping network statistics,
- registering and storing information on available resources.

The global index is a complex object which can be accessed with the SBQL syntax (as a typical database object) on every database engine plugged into the virtual repository. This means that we can evaluate queries on its contents. There is one substantial difference from processing typical virtual repository objects – as a result of an expression evaluation CMU can return only an actual content of index, like a list of online grid participants. The global index is the basic source of knowledge about the content of the virtual repository. Basing on indexed information referring the views' system we can easily integrate any remote data inside the virtual repository. If data have been indexed already, it can be transparently processed without any additional external interference. The *global index* has a specified structure which is a reflection of a *global schema* and it contains mainly additional objects for characterizing a type of a data fragmentation. These objects are dynamically managed through the views' systems whenever a virtual repository contents undergoes a change (e.g. when a resource joins or disconnects the virtual repository its local view cooperates with the global index). The global index keeps also dependencies between particular objects (complexity of the objects, etc.) as they are established in the *global schema*.

As a simple example of how the *global index* works we present grid objects named `Employee` which are registered in the central index. The grid controls two remote DBs (`DB_Krakow, DB_Lodz`), which map their objects named `Person` as the mentioned `Employee` objects in the grid schema (according to contribution views). Inside the global index list, each `Employee` object will contain two attributes as indexed location of Employee resources – the names of remote DBs – `DB_Krakow`

and `DB_Lodz`. Notice that the global index is a dynamic structure, thus any change in grid's available participants list will alter its content.

Each indexed object in the global index is equipped with a special object called *HFrag* (horizontal fragmentation) or *VFrag* (vertical fragmentation). Each of them keeps a special attribute named *ServerName*, whose content is a *remote object* – an identifier of a remote data resource (see Figure 6 and 8). If any new resource appears in a virtual repository, there will be added a suitable *ServerName* into the global index automatically with appropriate information about the resource.

Accessing the remote data can be achieved with calling the *global index* with:

```
GlobalIndex.Name_of_object_from_global_scheme.(Name_of_subobject).
HFrag_or_VFrag_object.ServerName_object;
```

Because every change of the virtual repository's content is denoted in the global index, accessing data in this way yields to be the only correct one.

Since every reference to a remote object must explicitly contain its location (a server it resides on), such a procedure would be too complicated for grid attendants. Moreover, it would not accomplish with transparency requirements and would complicate an automation of multiple resources integration process. Thus, we have decided to cover this stage together with automation of integration process behind a special procedure exploiting the updateable object views mechanism [4]. The process is described in the next chapter.

4.1 Examples of Transparent Accessibility of Remote Objects

Figure 4 depicts two examples of a distributed database schema to visualize our approach to data integrators. The first example realizes a transparent integration process concerning a horizontal fragmentation (Figure 4a). The complex objects Doctor and Ward are placed on three different servers, but the data structure for each sever is the same (see Figure 5), and the corresponding global index content is presented in Figure 6.

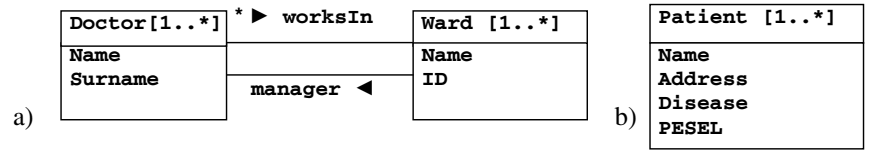

Fig. 4. Distributed databases schemata

The situation where data is horizontally fragmented in distributed resources forces merging all data as a single virtual structure, transparently achieved by all grid's clients. This process can be done by updateable object views [4] and the *union* operator. In our solution this operator is not needed, because virtual objects like *DoctorGrid* (and *WardGrid*) from a view definition procedure (see an example below) simply return a *bag* of identifiers to remote *Doctor* (or *Ward*) objects structured inside identically. The logical schema of this operation is presented in Figure 5.

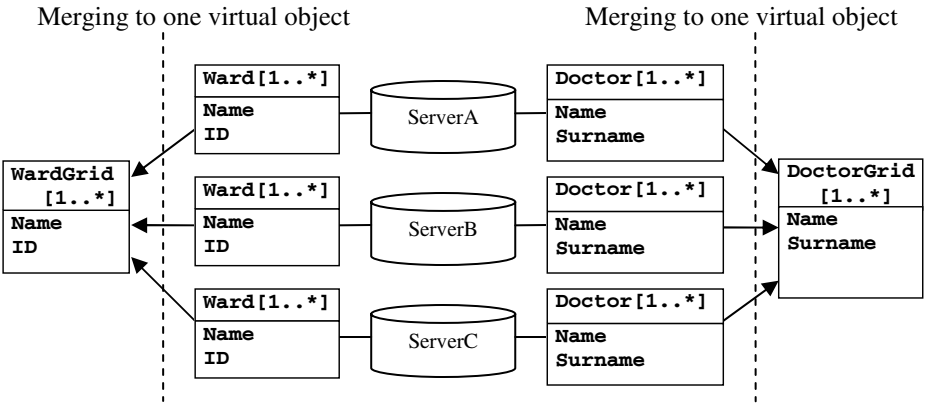

Fig. 5. Integration of distributed databases (with identical data structure) into one virtual structure for virtual repository

The updateable object views [4] action in integrating a horizontal fragmentation is presented by a following query evaluation (we retrieve names of all grid doctors who are working in the "cardiac surgery" ward):

(DoctorGrid ***where*** *WorksIn.WardGrid.Name = "cardiac surgery").Name;*

The *DoctorGrid* and *WardGrid* virtual objects definitions (through updateable object views [4]) are as follows:

```
create view DoctorGridDef {
virtual_objects DoctorGrid {
        return (GlobalIndex.Doctor.HFrag.ServerName).Doctor as doc};
                                    // return remote not fragmented objects doc
on_retrieve do {return deref(doc)}; //the result of retrieval virtual objects
        create view NameDef {
        virtual_objects Name {return doc.Name as dn}; //creating the virtual
                                    //subobjects Name of object DoctorGrid
        on_retrieve do {return deref(dn)}; //the result of retrieval virtual objects
        };

        create view SurnameDef {//…};

        create view worksInDef {
        virtual_pointers worksIn { return doc.WorksIn as wi};
        on_retrieve do {return deref(wi)};
        };
};
create view WardGridDef {
virtual_objects WardGrid {
        return (GlobalIndex.Ward.HFrag.ServerName).Ward as war};
on_retrieve do {return deref(war)};

        create view NameDef {
```

```
        virtual_objects Name {return war.Name as wn};
        on_retrieve do {return deref(wn)};
        };
        create view IDDef { //…};
};
```

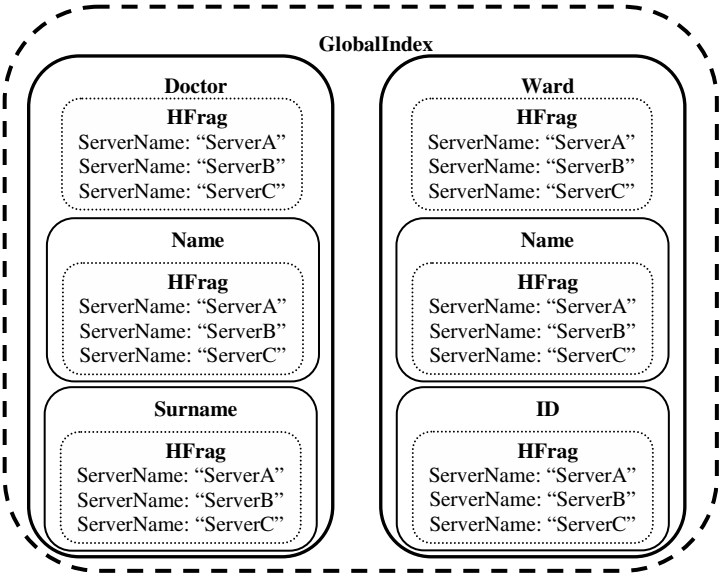

Fig. 6. The contents of CMU global index for example of horizontal fragmentation

Please notice, that for well keeping the original data schema inside a grid consentaneous to resource databases, the view must contain procedures for retrieving every subobject in an original objects' hierarchy. Therefore the views procedures, from above example, include subviews. These subviews also describe the methods of processing the called data.

The similar case concerns the GlobalIndex object, where the content also brings back the objects hierarchy like in participating databases, but in reality this is only a map for the indexing mechanism and the content is the information where the original objects and subobjects are physically stored.

Such an approach is always required where distributed resources and grid schema must have the identical representation of their contents.

The other example depicts a transparent integration process concerning a vertical fragmentation, which is more complicated, because we must join different data structures stored on physically separated servers. For solving this problem we use the *join* operator with a join predicate specific for an appropriate integration of distributed objects. The database schema (according to global schema) is presented in Figure 4b, where the complex object Patient is placed on three different servers where the data structure of stored objects is different (see Figure 7), according to this the global index content is presented on Figure 8.

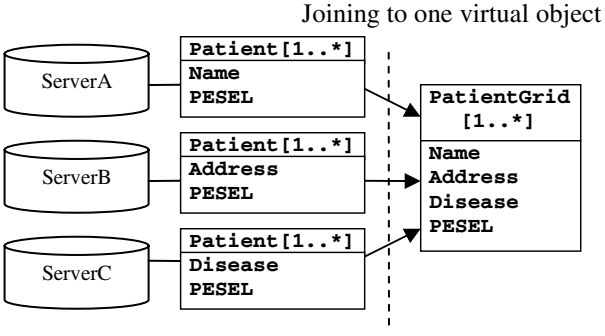

Fig. 7. Integration of distributed databases (with different data structure) into one virtual structure for virtual repository

The conclusion about a grid structure from the above example is that each server participating the virtual repository has a differential structure of stored data except for the PESEL object which has an identical content on each server. We utilize the knowledge about the object PESEL and its content to make "join" on the fragmented Patient object. The PESEL attribute is an unique identifier (predicate) for joining distributed objects into a virtual one.

This integration for a vertical fragmentation can be exemplified with a query evaluation: we retrieve names of all grid patients who suffer from cancer):

*(PatientGrid **where** Disease = "cancer").Name;*

The PatientGrid virtual objects definitions (through updateable object views [4]) are following:

```
create view PatientGridDef {
virtual_objects PatientGrid {
        return { (((GlobalIndex.Patient.VFrag.ServerName).Patient as pat).(
        ((pat where exist(Name)) as pn) join
        ((pat where exist(Address)) as pa where pa.PESEL = pn.PESEL) join
        ((pat where exist(Disease)) as pd where pd.PESEL = pn.PESEL)).(
                pn.Name as Name, pn.PESEL as PESEL, pa.Address as Address,
                pd.Disease as Disease)) as Patients };
                // return remote not fragmented objects Patients
on_retrieve do {return deref(Patients)}; //the result of retrieval virtual objects

        create view PatNameDef {
        virtual_objects Name {return Patients.Name as PatN};
        on_retrieve do {return deref(PatN)};
        };

        create view PatDiseaseDef {
        virtual_objects Disease {return Patients.Disease as PatD};
        on_retrieve do {return deref(PatD)};
        };
};
```

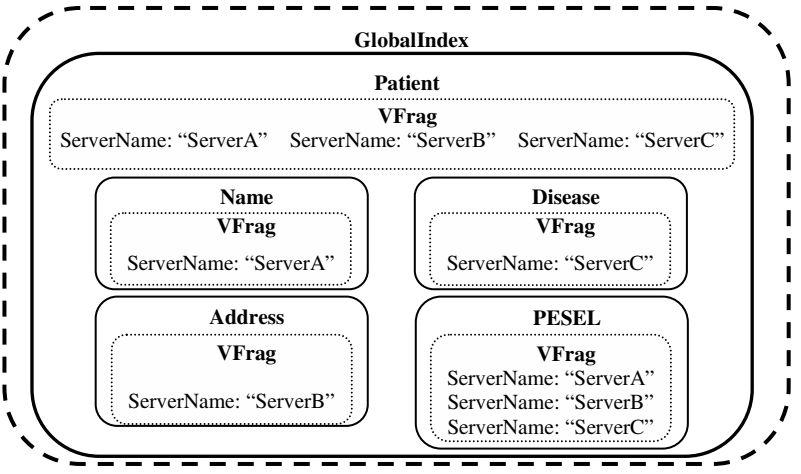

Fig. 8. The contents of CMU global index for example of vertical fragmentation

In the above examples we create virtual objects explicitly, what implies that a grid designer must be aware of fragmented objects in the grid schema. He or she does not need any knowledge of the fragmentation details, but must know which objects are fragmented. The rest of the integration process is executed automatically through SBQL syntactic transformations. Basing on the presented approaches it is easy to define an integration mechanism of objects fragmented both horizontally and vertically. In order to do so, we must combine them into a specific model. Also, we are able to extend our proposal for mixing fragmentation for different resources where some resources can store an object with a few different subobjects. Current object also can store simultaneously identical subobjects in different resources as a replication of the data. For this approach we have prepared the GlobalIndex object and we have equipped it with *HFrag* and *VFrag* subobjects. Our research aims to extend the global index structure with some information about a replication and redundant resources for reach transparency in a grid over such data structures and dependencies.

5 Conclusions and Future Work

We have presented a generic approach to a transparent integration of distributed data in a virtual repository mechanism. Our solution utilizes a consistent combination of several technologies, such as P2P networks developed on the ground of JXTA, a SBA object-oriented database and its query language SBQL with virtual updateable views. Our preliminary implementation solves a very important issue of independence between technical aspects of distributed data structure managing (including additional issues such as participants' incorporation, resource contribution) and a logical virtual repository content scalability (a business information processing). We expect that the presented methods of integration of fragmented data will be efficient and fully scalable. We also expect that due to the power of object-oriented databases and SBQL

such a mechanism will be more flexible than other similar solutions. The prototype is fully implemented and preliminarily tested.

The mentioned grid architecture is described in details in [6, 7, 8]. The presented P2P transport layer (also depicted as a grid/transport middleware) is independent of an application composition which provides a transparent communication between databases (grid data resources). Database's UI and engines are equipped with a separate protocol for a connection with P2P applications (at the transport layer level) – the details are described in [8] and in Figure 3.

Currently we are working on extending the presented idea to achieve a generic integration process through new functionalities like managing the mixed form of fragmentations, data replicas and redundancies of resources.

References

1. Foster I., Kesselman C., Nick J., Tuecke S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Global Grid Forum, June 22, 2002.
2. Kaczmarski K., Habela P., Kozakiewicz H., Subieta K.: Modeling Object Views In Distributed Query Processing on the Grid. OTM Workshops 2005, Springer LNCS 3762, 2005, pp.377-386
3. Kaczmarski K., Habela P., Subieta K.: Metadata in a Data Grid Construction. 13th IEEE International Workshops on Enabling Technologies (WETICE 2004), IEEE Computer Society 2004, pp. 315-316
4. Kozankiewicz H.: Updateable Object Views. PhD Thesis, 2005, http://www.ipipan.waw.pl/~subieta/, Finished PhD-s
5. Kozankiewicz H., Stencel K., Subieta K.: Implementation of Federated Databases through Updateable Views. Proc. EGC 2005 - European Grid Conference, Springer LNCS 3470, 2005, pp.610-619
6. Kuliberda K., Wislicki J., Adamus R., Subieta K.: Object-Oriented Wrapper for Relational Databases in the Data Grid Architecture. OTM Workshops 2005, Springer LNCS 3762, 2005, pp.367-376
7. Kuliberda K., Wislicki J., Adamus R., Subieta K.: Object-Oriented Wrapper for Semistructured Data in a Data Grid Architecture. 9th International Conference on Business Information Systems 2006, Klagenfurt Austria, Proceedings in Lecture Notes in Informatics (LNI) vol. P-85, GI-Edition 2006, pp.528-542
8. Kuliberda K., Kaczmarski K., Adamus R., Błaszczyk P., Balcerzak G., Subieta K.: Virtual Repository Supporting Integration of Pluginable Resources, 17th DEXA 2006 and 2nd International Workshop on Data Management in Global Data Repositories (GRep) 2006, Proceedings in IEEE Computer Society, to appear.
9. Moore R., Merzky A.: Persistent Archive Concepts. Global Grid Forum GFD-I.026. December 2003.
10. Nejdl W., Wolf B., Qu C., Decker S., Sintek M., Naeve A., Nilsson M., Palmer M., Risch T.: EDUTELLA, a P2P networking infrastructure based on RDF. Proc. Intl. World Wide Web Conference, 2002.
11. Open Grid Services Architecture, Data Access and Integration Documentation, http://www.ogsadai.org.uk
12. Subieta K.: Theory and Construction of Object-Oriented Query Languages. Editors of the Polish-Japanese Institute of Information Technology, 2004 (in Polish)

13. Subieta: Stack-Based Approach (SBA) and Stack-Based Query Language (SBQL). http://www.ipipan.waw.pl/~subieta, Description of SBA and SBQL, 2006
14. The JXTA Project Web site: http://www.jxta.org
15. Wilson B.: JXTA Book, http://www.brendonwilson.com/projects/jxta/
16. eGov-Bus, http://www.egov-bus.org