An Integration Model of Distributed Objects in a Virtual Repository Implementation basing on The ODRA OODBMS

Kamil Kuliberda\textsuperscript{1,3}, Michal Meina\textsuperscript{2}, Jacek Wislicki\textsuperscript{1,3}, Radoslaw Adamus\textsuperscript{1,3}, Tomasz Marek Kowalski\textsuperscript{1,3}

\textsuperscript{1} Technical University of Lodz, Lodz, Poland
\textsuperscript{2} Nicolaus Copernicus University, Torun, Poland
\textsuperscript{3} Polish-Japanese Institute of Information Technology, Warsaw, Poland

\{kamil, jacenty, tkowals, radamus\}@kis.p.lodz.pl, mich@mat.umk.pl

Abstract. The paper deals with integration of distributed fragmented collections of data, being the basis for virtual repositories in the data grid or P2P architecture. The core of the described architecture is based on the Stack-Based Query Language (SBQL) and virtual updateable SBQL views. Our virtual repository transparently integrates distributed, heterogeneous and fragmented data producing conceptually and semantically coherent result. In the background the system is based on the P2P architecture. We provide an example of data integration procedures for horizontal data fragmentation including object and relational data resources. The procedures are implemented under the integrator prototype covered behind P2P middlelayer.

1 Introduction

One of the current trends in software engineering is to provide technology and tools for development of applications for data intensive processing in distributed environment. The roots for this tendency lay in the business requirements (such as globalization and wide use of Internet) that applications must fulfill. Nowadays we can observe grow of computer technology connected to the development of distributed applications for processing a large amount of data. Such new solutions must conform to existent systems in such manner, that, both - old and new systems have to cooperate with each other. In a contemporary business models there are lot of associated services, where certain of them are important towards to globalization issue. The problem concerns a distribution of data locations and their forms. Because an OO data model is simple for organizing, there is also needed a simple solution for a transparent integration of the data model into one common and global data schema. Such a solution will be presented from scratch to prototype in the paper.

The paper deals with the data grid architecture for data-intensive grid solution

\textsuperscript{1} This work is supported by European Commission under the 6\textsuperscript{th} FP project e-Gov Bus, IST-4-026727-ST.
which covers technical aspects of data processing in a virtual repository (VR) which consecutively composes a complete technology for transparent processing of different forms of data. These proposals rely on Stack Based Architecture (SBA) and SBQL query language (implemented for ODRA database management system). The VR also contains a peer-to-peer transport platform (TP) - a non-limited communication middlelayer organized in a centralized architecture as a core for a grid management. Another mechanism used is object-to-relational wrapper used for integrating legacy relational data resources. The VR also deals with some aspects of higher forms of distribution transparency and some common infrastructures build on top of the grid, including the trust infrastructure (security, privacy, licensing, payments), web services, distributed transactions, workflow management, etc. [1].

2 Related Work

A data grid concept has many realizations in several projects where integration of distributed resources plays a principal role. Well known approach founded by SRB [20] covers low-level access to distributed resources and provide specific mechanisms for a logical identification of data and their metamodel distribution through specialized MCAT servers (which stands for model catalogue) based on a hierarchical approach. Data integration was based on porting API to wide range of programming languages on different platforms instead of integrating the data model itself. Development of SRB brought his successor called iRods [21]. The foundation of this approach is Rule Oriented Programming. The iRods provides an Adaptive Middleware (AM) [22] – a model of data processing where users arrange their needs of data access and integration without programming skills. The administrative functionality (rule catalogues, priorities, code reusable, etc.) gives flexibility among weak data structures and data process. Finally, it is presented as workflow-type rules which allow requesting the iRods servers to perform a sequence of operations (called micro-services) on behalf of the user. At the one hand it offers a stable and elegant tool for creation data processing, but at the second hand, an underlay model which supports micro-services mechanisms is a simply C-language functions mapped (in a specific way) into external rules. Mentioned AM solution from iRods does not fulfill taxonomy presented by Sadjadi in [22], because Rule Oriented Programming is to simple to use it as a programming language.

Another class of grid’s systems are projects like Globus Toolkit 4 (GT4) [23] and Unicore [24]. Their goal is a creation an environment which is able to utilize many heterogeneous systems and provide to the users simple solutions and libraries permitting on a use all of the environment components as well. In GT4 users can describe a task (which can deal with distributed data resources including a data fragmentation) and then it can be executed in the (grid) environment. For distributed locations of resources there we still encounter a problem of data integration. The GT4 developers have implemented gridFTP service which supports a calling distributed resources through its logical naming. This solution has a limitation in a form of distributed data type control – in the GT4 some data instances may be processed together even if they have incompatible type and/or structures. It may bring to
unexpected errors.

Commercial relational database systems (RDBMS-s) like Oracle-10G offer a flexible execution of distributed queries, but limited by a data model and languages not sufficient for distributed queries, in which programming suffers from inflexibility and complexity. Because current commercial trends use SQL-like RDBMS-s as basic data storages, also grid systems which are built on top of them, use DB engines as back-end layers. A leading standard of such solutions - OGSA-DAI [10] implements interesting data access and integration model. In OGSA-DAI, a data access to resources is placed on a lower-layer and combine web services to make a (parallel) communication. According to this, an integration mechanism for data is built on top of data access layer. Such a model allows the users to plug into the platform an independent and already existing solutions and technologies and then integrate them with the platform on the fly. It is also worth mentioning, that the prototype includes few simple libraries designed to create workflows. Workflow of the OGSA-DAI is composed of a set of sequentially executed activities (like generic SQLQueryActivity or TupleSplitActivity), that have many combined inputs and outputs. The drawbacks of the OGSA can be enumerated as; (1) the problem of efficient integration of data from databases which can be achieved only by classical DAO. Such a ways on accessing a data and the platform components may be a cause of countless undefined problems, (2) an optimization in OGSA-DAI is a tough problem. In the workflow definition every unit (activity) is actually a web service based on SOAP over HTTP. Flow execution per unit is highly costly. (3) There is only a programming tool (a library written in java) for workflow creation, but it is rather useless as a business tool because developers didn't share any modeling tool for a workflow creation, so finally a design engineering is impossible.

In next chapters we explain our approach to above issues and how we have provided a good and efficient way of dealing with these topics by developing the ODRA, SBQL and cooperated solutions, like P2P and Object-to-Relational Wrapper.

3 Details of Virtual Repository

. Processing in a virtual repository comprises some complex issues. One of them is updating virtual data seen through a VR, the state of research on this problem is open. Similar problems concern security and global transaction infrastructures that are built over of a VR. Another problem concerns performance issues, in particular, optimization of global queries expressed in a VR query language.
Problems described above raise an issue of developing generic methodologies, environments, tools and languages that support a quick development of a grid with a virtual repository aiming at a particular application integration goal. This problem can be solved with an architectural idea of the grid components presented in Figure 1 and with developing concrete technical solutions concerning particular components:

- developing a canonical model and a schema according to global user requirements (several directives like a business contract, a standard, law regulations, etc.). The model and the schema should be implemented in a corresponding language having both human and machine interpretations,
- developing local models and schemata of providers for participating local data and service resources in terms of a canonical model and a schema, i.e. showing how particular local providers contribute to the global schema,
- developing an integration schema exploiting a local schema and a global schema which shows dependencies between local resource providers and the global view and dependencies between the local providers (redundancies, replications, etc.).
- developing a communication middle-layer which performs on easy connection of VR participants’ and where data may be transported transparently without any limitation (e.g. NAT, firewalls, etc.) between all connected units.
- developing a tool for transparent processing of remote objects,
- developing assumptions concerning export wrappers for particular providers sharing their resources.

Although the literature contains many works concerning the above issues and problems, the field is rather far from a complex and universal solution. Our research based on generic object-oriented database model with updatable views focused on to designing well defined grid mechanisms creates a big chance to receive significant theoretical and practical results much beyond the current state of art.

The technical aspects of realization of the described idea assume an existence of
several cooperating technologies nearby. The principal aspect is a good design of a VR platform corresponding with two other aspects responsible for data resources maintenance (wrappers) and a business data exchange including a grid security and its management (a TP). The concept concerns existing individual software modules with interconnectivity mechanisms enabling a restricted and specified participation of data processing. It has an original and general tendency. The modules may be developed and implemented according to the idea of a project’s architect.

Our concept assumes the following strategies:

• a virtual repository – physically available with applications based on the previously described architecture. Parts of a repository are: client, provider application and management application.

• a transport platform [6] – determines independent software environment responsible for free distributed transaction processing. The platform particularly should grant an unlimited physical access to a grid network for clients and resource providers (units) and an assurance of a well formed protocol for an information interchange. It is based on a centrally managed peer-to-peer network infrastructure. All vital aspects can be developed using a multiprotocol, fully programmable P2P platform of the JXTA project [12]. Because of the space limitation the details of the transport platform implementation will be omitted.

• import/export adapters and wrappers – mechanisms supporting a grid architecture to import and export local resources which may contribute to a virtual repository. They are software modules enabling a resource provider’s services exploitation. Each of grid clients and providers will be equipped with user selected or corresponding to a grid contribution schema modules which can discover local data and thanks to views mechanisms grant access as a part of a virtual repository [4].

3.1 SBA and SBQL in ODRA Platform

Integration of the distributed data requires sophisticated and flexible platform with an adequate set of features including:

• generic data model that is able to express wide range of possible models of data sources

• powerful query/programming language with full computational power of programming languages seamlessly integrated with a high-level query languages capabilities.

• virtual views with full update functionality that allow to: wrap local resources to common data model; build global applications through integration of local resources; customize global assets to the needs of a particular client applications.

All abovementioned features are hardly to achieve using current state-of-the-art platforms, programming and/or query languages and common data models. Fortunately the methodology called Stack Based Approach (SBA) to query/programming languages [13, 14] is able to equip the SBA-based platforms with all the tools to work out required solution. SBA extends the database programming with all the popular object-oriented mechanisms and introduce new, previously unknown like e.g. dynamic object roles and virtual updatable views.

The query language that is based on SBA is called Stack Based Query Language
(SBQL) [13, 14] that is new self-contained query/programming language where there is no distinction between traditional programming language expressions and declarative constructs (queries) for data processing. SBQL is defined on very general data store model based on object relativism and full internal identification principles.

SBQL also supports imperative constructs and mechanisms, e.g.: control structures, procedures, classes, interfaces, modules. One of the most important features of SBQL are virtual updatable object views [18]. The entirely new property of SBQL views is its full updatability. View designer/programmer has full control over the update operations that are performed on the virtual objects defined by the view. There are no restrictions concerning the type of operation that is allowed. The view definition can contain nested view definition, procedures, variables, etc.

The advantages of SBA and SBQL have been implemented in a project called ODRA (Object Database for Rapid Application development) [19]. The aim of it is to design an object-oriented application development platform – the tool for future database application programmers. Because the ODRA features are still under development and continuous extension below we summarize only the most important ones:

- The core of ODRA environment is the SBQL language defined for a general data store model based on the object relativism principle. They are no dangling pointers and null values.
- Queries are treated the same way as expressions in popular programming languages. The design is based on the compositionality principle (e.g. select-from-where syntactic sugar is avoided).
- Names occurring in queries are bound using environment stack (ENVS), a structure well known from popular programming languages implementations.
- Operators are divided for the sake of its association with the ENVS: algebraic (e.g. +, -, auxiliary name) do not use ENVS; and non-algebraic (navigation, where, quantifiers, join, etc.) that use it in a similar way to procedures (opens new ENVS section and execute against new ENVS state).
- ODRA introduces the concept of updatable object views that allows performing transparent update operation on virtual data. The view is the core for data and application integration using ODRA [4].
- ODRA implementation includes strong query optimization techniques based on query rewrite (procedures and views rewrite), query modification (independent sub-queries, removing dead sub-queries) [9] and indices.
- ODRA is able to plug existing data sources through wrappers (e.g. relational data) and filters (e.g. XML and RDF). Next such a data can be queried with SBQL queries.

All abovementioned features of ODRA environment predestine the platform to be an excellent tool for implementation of virtual repositories for data grid applications.

3.2 Distributed Data Representation through Data Grid in Virtual Repository

The presented proposal for processing distributed and heterogeneous resources assumes different approach to work with data than similar solutions [9, 10, 15, 16]. Our solution represents a data grid architecture (consistent with MDA architecture)
which is described in details in [5, 6]. There we claim that neither data nor services can be copied, replicated and maintained in the centralized server. They are live to be supplied, stored, processed and maintained on their autonomous sites [5, 6, 7, 8]. The external (and existing) resources should be easily pluggable into the system as well as users can appear and disappear unexpectedly.

A user as well as a data provider (see Figure 1 at resources area) may plug into a VR and use its resources according to his or her requirements, availability of the resources and assigned privileges. The goal of our research is to design a platform where all users and providers are able to access multiple distributed resources and to work on the ground of a **global schema** (at client side) for all the accessible data and services. A virtual repository should present a middleware supplying a fully transparent access to distributed, heterogeneous and fragmented resources from its clients [1, 5, 6].

Looking on the system at the participants side (together - clients and data providers) there are two kinds of data schemata. The first one, a **contributory schema**, is the description of a local resource acceptable for the VR. A virtual repository can deal only with the data that is exported by the decision of a local administrator. Another reason for limited access to local resources is some **consortium agreement** which is established for the VR. The agreement has certain business goals and need not to accept any data from any provider [5, 6]. As a second schema we claim **global schema**, which is a description of a global data and services available for clients.

The basic assignment to solve for the VR is a transformation of local provider’s data through contributory schema into a global schema. The transformation can perform more sophisticated homogenization of data and integration of fragmented collections. This is done by updatable views that are able to perform any data transformation. Our views have the full algorithmic power of programming languages, thus are much more powerful than e.g. SQL views.

A responsibility of managing grid contents through access permissions, discovering data and resources, controlling location of resources and indexing whole grid attributes are basic tasks of a global infrastructure. The design and implementation challenge is a method of combining and enabling non-limited both-way processing of clients’ and data providers’ contents which participate in VR’s global virtual store [5].

Our data grid architecture provides a views system (containing contributory, integration and global view, see in Figure 1) available to every VR’s participant. Moreover, each data provider possesses a view which transforms its local share into an acceptable contribution, a **contributory view**. Providers may also use extended wrappers to existing DBMS systems (see sub-chapter 3.4 and [7, 8]). Similarly, a client uses a **global view** to consume needed resources in a form acceptable for his or her applications. Between those views is placed an **integration view** which is a core mechanism for a transparent resources integration. 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 automatic semantic-based analysis. The problem is how to allow transparent plugging in new resources and how to incorporate them into existing and working views. This question is discussed in section 3.3.
3.3 Wrapping External Data Models into Virtual Repository

A wide presentation of wrapping problems is discussed in previous works [7, 8]. Now, we present some implementation aspects of mentioned wrapper modules. A grid resource (an ODRA engine) denotes any data resource providing an interface capable of executing SBQL queries and returning SBQL result objects as their results. A nature of such a resource is irrelevant, as only the mentioned capability is important. In the simplest case, where a resource is an ODRA database, its interface has a direct access to a data store and it is similar to an ODRA database engine (DBMS). However, as our grid aims to integrate existing business resources, whose models are mainly relational ones, an interface becomes much more complicated, as there is no directly available data store – SBQL result objects must be created dynamically basing on results returned from SQL relational queries evaluated directly in a local RDBMS.

Such cases (the most common in our grid's real-life application) force introducing additional middleware, a wrapper (showed as the O-R Wrapper) in Figure 1 as a client-server solution. This kind of architecture was applied for a few reasons. One of them are simplicity of implementation and portability. A standard ODRA database can be extended with as many wrappers as needed (e.g. for relational or semistructured data stores) and plugged into any resource model without any lost of its primary performance. Furthermore, a wrapper server can be developed independently, providing a communication protocol to its client. Of course, an ODRA database with a wrapper's client can work on a separate machine.

A query evaluation process in our data grid environment is depicted in Figure 2. One of the global grid applications sends a query (arrow 1). This query is expressed with SBQL, as it refers to the business object oriented model available to grid users. According to the global schema and its information on data fragmentation, replication and physical location (obtained from integration schemata), the query is sent to appropriate resources. In Figure 2 this stage is realized with arrows 2, 2a and 2b.

The partial query aiming at our relational resource is further processed with a resource's ODRA interface. First, the interface performs a query optimization. Apart from efficient SBQL optimization rules applied at any grid resource's interface, here we can also transform queries so that powerful native SQL optimizers can work and amounts of data retrieved from the RDBMS are acceptably small. Relational optimization information (indices, cardinalities, primary-foreign key relationships, etc.) is provided by the wrapper server's resource model (arrow 3) and appropriate SBQL query syntax tree transformations are performed. Appropriate tree branches (responsible for such SQL queries) are substituted with calls to execute immediately procedures with optimizable SQL queries.
Once syntax tree transformations are finished, the interface starts a regular SBQL query evaluation. Whenever it finds an `execute immediately` procedure, its SQL query is sent to the server via the client (arrows 4, the client passed SQL queries without any modification). The server executes SQL queries as a resource client (JDBC connection), arrow 5, and their results, arrow 6, are encapsulated and sent to the client (arrow 7). Subsequently, the client creates SBQL result objects from results returned from the server (it cannot be accomplished at the resource site, which is another crucial reason for a client-server architecture) and puts them on regular SBQL stacks for further evaluation (arrow 8). In the preferable case (which is not always possible), results returned from the server are supplied with TIDs (tuple identifiers), which enables parametrizing SQL queries within the SBQL syntax tree with intermediate results of SBQL subqueries. Having finished its evaluation, the interface sends it
“partial result” upwards (arrow 9), where it is combined with results returned from other resources (arrows 9a and 9b) and the global query result is composed (depending on fragmentation types, redundancies and replication). This result is returned to the global application (arrow 10).

In a next chapter we present integration and wrapping example for data set which is possible in real-life events.

4 Integration and Data Wrapping by Example

Here we present little complicated example of retrieving particular data objects through our grid solution – a VR for external RDBMS resources which are available through the wrapper modules. Because there is no space for detailed description of all cases and its interrelationships we will be focused only on basic problems concerning the integration.

At the beginning we introduce how we create ad use integration of distributed resources, later will show an example how we obtain data from external DBMS-s.

As an example we use following objects’ schema in our VR (see Figure 3).

![Virtual repository global data schema](image)

Fig. 3. Virtual repository global data schema.

We also assume that schema from figure 3 is a representation of a global schema available for grid contributors. In such case, local data from providers’ resources need to be contributed according to above objects’ structure including dependencies between a particular objects. Obviously, the structure of local data in providers’ databases may differs, then a local administrator must conform the local schema to contribution schema by local implementation of an individual contribution view.

Please notice, that a creation of a grid data model is strongly dependent from principles of a grid consortium. Thus, its every aspect must be agreed, for e.g. if there is assumption that a contribution schema has specific structure (object naming), then providers during contribution process must prepare appropriate contribution views. As example; if we assume that in whichever source the objects Employee will be named as Emp and a grid consortium intends that in contributory schema all objects must have an specified name EmployeeContrib, and if there are all overloaded
operations for managing virtual objects like on_retrieve, on_delete, on_new, on_update, then administrators of contributed resources must prepare a view which should look like:

```plaintext
view EmployeeContribDef {  
  // declaration of virtual objects provided by the view  
  virtual EmployeeContrib: record |idContrib: integer; nameContrib: string; surnameContrib: string; sexContrib: string; salaryContrib: real; infoContrib: string; birthDateContrib: date; worksInContrib: DepartmentContrib| [0..*];  
  //seed is a parameter which unambiguously defines a virtual object  
  seed : record |e: Emp; |[0..*] | return (Emp) as e; }  
  
  //operator which defines a view behaviour for retrieval virtual objects  
  on_retrieve { return e.(id as idContrib, name as nameContrib, surname as surnameContrib, sex as sexContrib, salary as salaryContrib, info as infoContrib, birthdate as birthDateContrib,  
    //definition for virtual pointer worksIn  
    (DepartmentContrib where idContrib = e.worksIn.Dept.id) as worksInContrib); }  
  
  //operator which defines a view behaviour for deletion virtual objects  
  on_delete { delete e; }  
  
  //operator which defines a view behaviour for updating virtual objects  
  on_update { e := value.(idContrib as id, nameContrib as name, surnameContrib as surname, sexContrib as sex, salaryContrib as salary, infoContrib as info, birthDateContrib as birthdate,  
    //definition for virtual pointer worksIn  
    ref(Department where id = worksInContrib.idContrib) as worksIn); }  
  
  //operator which defines a view behaviour for creation a new virtual object  
  on_new { create permanent Emp.(value.(idContrib as id, nameContrib as name, surnameContrib as surname, sexContrib as sex, salaryContrib as salary, infoContrib as info, birthDateContrib as birthdate,  
    //definition for virtual pointer worksIn  
    ref(Department where id = worksInContrib.idContrib) as worksIn)); }  
  
  //define the subview for access and management of ‘id’ subobjects of Emp  
  view idContribDef {  
    virtual idContrib: integer;  
    seed: record | _id: Emp.id; | return e.id as _id; }  
  on_retrieve { return _id; }  
  on_update { _id := value; }  
}  
  
  //define the subview for access and management of ‘name’ subobjects of Emp  
  view nameContribDef {  
    virtual nameContrib: string;  
    seed: record | _name: Emp.name; | return e.name as _name; }  
  on_retrieve { return _name; }  
  on_update { _name := value; }  
}  
```

This step is absolutely required in our grid solution, because through a next step, during process of auto-plugging-in the current contribution into the grid, mentioned (in chapter 3.3) view generator, will automatically produce an integration view which will consist of a set of dblinks with designation to the contribution view’s objects.
The form of the integration view will be following for the current example (we assume that our contribution - as a dblink form, has the name ‘WarsawPeer’):

```plaintext
view EmployeeIntegrDef {
    virtual EmployeeIntegr: record { idIntegr: integer; nameIntegr: string; surnameIntegr: string; sexIntegr: string; salaryIntegr: real; infoIntegr: string; birthDateIntegr: date; worksInIntegr: DepartmentIntegr; }[0..*] ;
    seed : record { e: mylink.EmployeeContrib; }[0..*] { return (WarsawPeer.EmployeeContrib union KrakowPeer.EmployeeContrib) as e; }
    on_retrieve { return e.(idContrib as idIntegr, nameContrib as nameIntegr, surnameContrib as surnameIntegr, sexContrib as sexIntegr, salaryContrib as salaryIntegr, infoContrib as infoIntegr, birthDateContrib as birthDateIntegr, (DepartmentIntegr where idIntegr = e.worksInContrib.DepartmentContrib.idContrib) as worksInIntegr ); } on_delete { delete e; }
    on_update { e := value.(idIntegr as idContrib, nameContrib as nameIntegr, surnameContrib as surnameIntegr, sexContrib as sexIntegr, salaryContrib as salaryContrib, infoContrib as infoContrib, birthDateContrib as birthDateContrib, ref{"WarsawPeer.DepartmentContrib union KrakowPeer.DepartmentContrib} where idContrib = worksInContrib.idContrib) as worksInContrib); }
    on_new { create permanent EmployeeContrib(value.(idIntegr as idContrib, nameContrib as nameIntegr, surnameContrib as surnameIntegr, sexContrib as sexContrib, salaryContrib as salaryContrib, infoContrib as infoContrib, birthDateContrib as birthDateContrib, ref{"DepartmentContrib where idContrib = worksInContrib.idContrib} as worksInContrib)); }
}
```

view idIntegrDef {
    virtual idIntegr: integer;
    seed : record { _id: mylink.EmployeeContrib.idContrib; } { return e.idContrib as _id; }
    on_retrieve { return _id; }
    on_update { _id := value; }
}

view nameIntegrDef {
    virtual nameIntegr: string;
    seed : record { _name: mylink.EmployeeContrib.nameContrib; } { return e.nameContrib as _name; }
    on_retrieve { return _name; }
    on_update { _name := value; }
}
```

The above view source shows rules how our integration model works. (1) Virtual object from integration view processes virtual objects from contribution view which are explicitly defined according to grid’s consortium model. (2) Integration works only for a horizontal fragmentation of data – it means, contribution schema in all contributed resources must be the same. (3) Remote objects are delivered through dblink objects. (4) Information about composition of remote objects is stored and combined inside seed parameter of the view (see the grey-marked line in the integration view listing). There is composition of two resources connected by union operator. As a result we obtain a bag of remote objects from both sources. If
necessary we can extend or change this definition by adding or removing any source (a dblink and attendant object). The definition will be changed automatically by a view generator’s action. (5) Similar situation must be performed for objects which are dependent on each other. This situation shows second grey-marked line in the integration view – inside on_update operation. There is definition how to update virtual pointer named worksIn if someone calls updating query on grid/global schema.

On top of the integration view inside each of grid’s contributor must be placed global view which determines a direct interface to grid’s objects for clients and data providers. Its structure is similar to contribution and integration view, but virtual objects’ definitions calls in static way only virtual objects from an integration view. We do not show the view body example, because of limited space of the paper.

Integration with Object-to-Relational Wrapper is more complicated. We assume that relational schemata (see Figure 4) are automatically wrapped to simple internal object-oriented schemata where a table corresponds to a complex object, while a column corresponds to its simple sub-object with a corresponding primitive data type. The applied naming convention uses original table and column names, except logical relations (in current model) employees.id and cars.owner_id columns (where an employee can own a car). These simple object-oriented models are then enveloped with administrator-designed views (regarded as contributory schema views) realising virtual pointers responsible for primary-foreign key pairs, the views can also control data access (e.g. disallowance of updating primary keys and virtual pointers). The code of the views used in the example (agreeable with contribution information) is below:

```sql
view EmployeeContribDef {
  virtual EmployeeContrib: record {idContrib: integer; nameContrib: string; surnameContrib: string; sexContrib: string; salaryContrib: real; infoContrib: string; birthDateContrib: date; worksInContrib: DepartmentContrib; }[0..*] ;

  seed : record [e: employees; ][0..*] { return employees as e; }
```

![Fig. 4. Relational schema wrapped into integration example.](image-url)
The grey-marked lines in above listing shows how we cover a relational schema and weave it into contribution schema. A physical mapping (data traversing) between schemata is made by the wrapper application. Its construction (see chapter 3.4) permits on processing data form relational resources within the grid and the virtual repository without any limitations.

Please to be careful, because above code of views’ examples are not showed entirety. A code of Employee objects in contribution and integration views does not include some sub-views’ and virtual pointers’ definitions. Also definitions of Department, Location, Car, Model and Make are omitted.

Continuing our integration example, an usage of (global) object schema from Figure 3, a grid clients can process from contributed OO-resources and relational-resources following examples of SBQL queries:

- Retrieve surnames of employees and names of departments of employees named Nowak

(Employee as e join e.worksIn.Department as d).(e.surname, d.name);
• Set salaries of all employees who earn less than 1000 to 1200
  (Employee where salary < 1000).salary := 1200;

• Retrieve cars owned by employees named Nowak
  (Car as c where c.isOwnedBy in (Employee as e where e.surname = "Nowak").(e.id)).c;

5 Conclusions and Future Work

The paper presents a solution to a transparent integration of distributed data in a virtual repository mechanism and a data grid architecture. The solution utilizes a consistent combination of several technologies such as; SBA object-oriented database as basic system for design a data grid architecture; the SBA’s query language SBQL as an interface for processing distributed data; virtual updatable views as a complex structure for a distributed data virtualization into a global schema; transport platform (P2P networks developed on the ground of JXTA as a background layer for unlimited communication); and an object-oriented to relational wrapper.

A preliminary implementation solves a very important issue of independence between technical aspects of distributed data retrieving through the wrappers, managing through set of views (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 solution 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 “proof-of-concept” implementation was already used in two EU projects:
– e-Gov Bus (EC 6-th FP, IST-26727) where it plays the role of the virtual repository software
– VIDE(EC 6-th FP, IST 033606 STP) devoted to MDA tools, where the part of the platform are used as model execution engine and the legacy data integrator.

Currently we are working on extending the presented idea to achieve better flexibility for a real data models. This will include solutions to managing the data replicas and redundancies residing inside distributed resources.

References

computing, high-performance and Distributed Applications (GADA’06), Springer 2006
LNCS 4276, pp.1290-1304.

Repository Supporting Integration of Pluginable Resources. 17th DEXA 2006 and 2nd
International Workshop on Data Management in Global Data Repositories (GRep) 2006,
Proc. in IEEE Computer Society

Databases in the Data Grid Architecture. OTM Workshops 2005, Springer LNCS 3762,
2005, pp.367-376

Semistructured Data in a Data Grid Architecture. 9th International Conference on Business

EDUTELLA, a P2P networking infrastructure based on RDF. Proc. Intl. World Wide Web
Conference, 2002.

10. Open Grid Services Architecture, Data Access and Integration Documentation,
http://www.ogsadai.org.uk

Warszawa 2000

12. Project JXTA Community: http://www.jxta.org

13. Subieta K.: Stack-Based Approach (SBA) and Stack-Based Query Language (SBQL).
http://www.ipipan.waw.pl/~subieta, Description of SBA and SBQL, 2006

Polish-Japanese Institute of Information Technology, 2004 (in Polish)

15. Tatarinov I., Ives Z., Madhavan J., Haley A., Suciu D., Dalvi N., Dong X., Kadiyska Y.,
Miklau G., Mork P.: The Piazza Peer Data Management Project, ACM SIGMOD Record,
32(3), 2003

SIGMOD Conference 2004: 539-550


~subieta/ -> Finished PhD-s -> Hanna Kozankiewicz

19. Lentner M., Subieta K., ODRA: A Next Generation Object-Oriented Environment for
Rapid Database Application Development, 2006,


Cyberinfrastructure for Large-Scale and Long-Term Data Management, Fall AGU 2007,

CSE-03-35, Michigan State University, East Lansing, Michigan, December 2003

23. GT4, open source software toolkit used for building Grid systems and applications,
http://www.globus.org/


http://www.omg.org/docs/omg/03–06–01.pdf