Transparency in Object-Oriented Grid Database Systems

Krzysztof Kaczmarski[1], Piotr Habela[3], Hanna Kozankiewicz[2], Krzysztof Stencel[4], and Kazimierz Subieta[2,3]

1 Warsaw Univesity of Technology k.kaczmarski@mini.pw.edu.pl
2 Institute of Computer Science, Polish Academy of Sciences
{hanka, subieta}@ipipan.waw.pl
3 Polish-Japanese Institute of Information Technology habela@pjwstk.edu.pl
4 Warsaw University stencel@mimuw.edu.pl

Abstract. The paper presents various transparency issues that have to be considered during development of object-oriented Grid applications based on virtual repositories. Higher-level transparencies, such as location, heterogeneity, fragmentation, replication, redundancy, indexing and service provider transparency assure new information processing culture greatly supporting the development, operation and maintenance of Grid database applications. The paper discusses some requirements for a virtual repository that is a kernel of a Grid database and a general architecture of such systems. The architecture is based on object-oriented updatable database views that serve, in particular, as adapters of local servers and integrators/mediators on the level of a global virtual repository. Finally, some issues of the development of Grid database applications are presented.

1 Introduction

In distributed businesses digital data is scattered across different file systems, content repositories, databases, legacy applications and web sites, making it difficult to discover, reuse and integrate into content enabled processes [2,8]. In situation when global client software may need to access thousands of distributed resources, their complexity may undermine feasibility of business software goals. In addition, conceptual simplification has direct impact on various vital business factors, such as the cost, time and manpower necessary for software manufacturing, the quality of service (reliability, stability, effectiveness, etc.), the quality and size of software source and documentation, the software maintenance/change cost and others [7].

The mentioned effective data usage problems might be greatly simplified by transparent integration of resources and hiding technical details. Higher forms of transparency assure new information processing culture, where the client of a system may concentrate his/her effort on just services and abstract from technical and business peculiarities of particular service providers and contracts with them. Transparency has various forms, in particular:

- hardware, operating system, communication/transport protocol, file system and database management system transparency – users have no need and no possibility to involve these features into their programs,
- data/service location and access transparency – users need not care for the geographical location of data and services,
- concurrency transparency – users can access resources simultaneously and need not know about the existence of other users,
- heterogeneity transparency – users do not see local data structures and local system implementation but operate on higher and common form of information
- scaling transparency – servers, data and services may be added or removed without impact on the applications and the users,
- fragmentation transparency – users need not be aware that data is partitioned; the fragments are integrated automatically,
- replication transparency – users need not be aware that data is replicated; replicas may be transparently added or removed to improve the efficiency of processing,
- redundancy transparency – users are isolated from redundant data that may exist among all local systems,
- site or connection failure transparency – most users can still work after some of the nodes or communication links are broken,
- migration transparency – data and services can be moved without any impact on the applications and users,
- optimization transparency – indexing, query cashing and rewriting, pipelining, decomposition and other optimizations are done on the level invisible for users,
- service provider transparency – users are interested in just services, service providers are hidden or are shifted to a secondary scene.

Typical approaches to Grid technologies consider some limited forms of transparency, from the above list. The current tendency is, however, to achieve advanced transparency forms that will simplify and unify the access of the user of global applications to the entire resources of the organization. Full heterogeneity, fragmentation, replication, redundancy, migration, optimization and service providers transparencies are considered in the context of virtual data/service repositories rather than in classical grid-oriented literature. However, achieving all the advanced forms of transparency is a big challenge for developers. Usually existing systems do not solve it to satisfactory degree. Some of transparency forms, although conceivable to achieve, may result in compromising the performance of applications, thus tradeoffs between performance and transparency might be required.

Usually, systems integrating heterogeneous resources propose kind of a middleware solution that could be a basis for achieving many forms of transparency. The main goal is to create a new layer of abstraction, which could seamlessly integrate any technology - data and service, and perform additional administrative operations in a way invisible for end-users. Fig. 1 presents the general approach

to transparency that is already implemented in many systems (CORBA, RMI, WebServices, etc). It is usually realized by an architecture employing three or more layers.

Transparency achieved by virtual repositories has two organizational aspects. The first one could be called functional and focuses on generating virtual data reflecting real, concrete resources in a form acceptable for clients. This must be done according to certain business goals. The design is created by an involved consortium (called a Virtual Organization [3]). Complexity of this task may vary in different applications. Thus, business analysts, Grid designers and managers play a key role here. The second aspect may be called operational (or administrative) and it must assure proper operation of the virtual repository. This part of the system is independent of the chosen data structures and business goals. It is responsible for automatic performing repository reconfiguration, meta-database modification, indexing, caching and other operations in dynamically changing environment.

Both of these aspects accomplished by the system simultaneously may guarantee proper integration of distributed heterogeneous databases. If such a system offers transparency on a satisfactory level, we call it a Grid Database. In this paper, we discuss basic issues of Grid databases such as common (canonical) data model that is to be introduced on the Grid global level, the architecture and general design phases of a Grid database creation. These issues are currently investigated in our prototype implementation ODRA (Object Database for Rapid Application development), which is based on the Stack-Based Approach to object-oriented query/programming language, the language SBQL and updatable views defined in SBQL. ODRA is assumed to be a platform for enhanced Web Services and advanced Grid applications.

The rest of the paper is as follows. Section 2 discusses some issues of a virtual repository - a kernel of Grid database applications. Section 3 is devoted to the architecture, based on the concept of object-oriented updatable database views. Section 4 presents generalities of the creation of a Grid database. Section 5 concludes.

2 Virtual Repository in a Grid Database

A big advantage of virtual repositories is that data and services need not to be copied, replicated and maintained on the global applications side: they live on their autonomous sites and are locally supplied, stored, processed and maintained. In many businesses copying data (in any form) is not allowed due to local security policies. Virtuality of data, however, requires automatic mechanisms allowing one to access the data as easily as if they resided inside one machine. This property is extremely difficult to achieve if virtual data are to be updated and mappings from stored to virtual data are a bit more sophisticated.

Many virtual repositories are already implemented and operating. Usually, they are proprietary solutions; no general standard or conceptual frame is ob-

served. The repositories differ in end-user or programmer interface, security capabilities, forms of transparency and other aspects.

A little step forward was done by Web Services [12] but their capabilities in the field of transparently distributed systems are limited, what was already observed by the OGSA standard [3]. In fact, they achieve only basic forms of transparency. Global applications based on distributed, heterogeneous and redundant data/service resources databases require technical features much beyond the current capabilities of Web Services.

2.1 Common Data Model – Functional Aspect

Transparency can also be considered as a higher abstraction level over distributed, heterogeneous and redundant data and services. From the functional point of view, the abstraction means a common canonical data model, which is universal, simple and minimal. From among many approaches (data models) that can be considered as candidates to build canonical models we have chosen the Stack-Based Approach (SBA) and a corresponding query language SBQL [8, 10]. SBA introduces (as the main notions): complex objects, classes, methods, inheritance, dynamic object roles, encapsulation and other concepts of the object-orientedness. Each object has an internal identifier, external name and may contain a value, a link or a set of objects. The major difference between SBA and other object-oriented database models is that it is based on a fully-fledged query/programming language with precisely defined formal operational semantics. A programmer writes queries and updating statements based on queries, which may manipulate objects in repositories: create new objects, remove, update and insert into other objects, call their methods, create and assign new roles, etc. Queries have all well-known capabilities plus some more advanced like transitive closure. One of the most important features of the SBA data model is virtual objects (updateable views), which from the programmer's point of view are undistinguishable from stored objects. Virtual objects do not exist in any real data storage but are determined by view definitions. A view definition establishes two kinds of mappings: (1) from stored objects into virtual ones; (2) from any operations on virtual objects (reading, updating, deleting, inserting, etc.) into operations on stored objects. This mechanism of updatable views makes it possible to achieve many transparency forms mentioned in the introduction, in particular, location, fragmentation, indexing, replication and redundancy transparency.

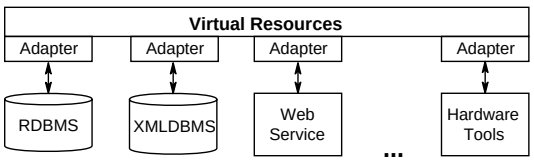

Fig. 1. General approach to transparency

Mappings of stored objects into virtual ones are also known from other approaches, for example, SQL views in relational databases. In all such solutions, however, the programmers must face the problem of reverse mappings if one would like to update information delivered by a virtual object [6, 11]. Some systems simply forbid such possibilities. Other, like Oracle and MS SQL Server, require so called INSTEAD_OF triggers. Our updatable object views define five generic operations, which must be defined for each virtual object kind in order to achieve full transparency between stored and virtual objects. These are: on_update, on_insert, on_delete, on_retrieve and on_create (on_create_pointer). They cover all generic operations that may be performed on objects. Explaining all subtleties of this data model is beyond the scope of this paper; please refer to [5, 6, 10] and other references to SBA and SBQL.

Similarly to normal objects, virtual ones are identified by virtual identifiers. Their construction allows the system to distinct them from normal identifiers and use dedicated methods, while a programmer is not aware of their virtual nature.

2.2 Operational Aspect of Transparency in Grids

Apart from establishing a common data model, which is used in a virtual repository, a Grid Database must also implement features assuring proper operation of the whole system. The general idea is to make such features as invisible as possible for the users. This allows the system to achieve a higher level of transparency, which is the main benefit for administrators, programmers and end users.

However, some solutions, even if very useful, may be very expensive while other may be too complex. A consortium planning to create a virtual repository based on a Grid Database should always carefully analyze possible advantages and drawbacks, finding a reasonable tradeoff between higher forms of transparency, satisfactory performance, programmers/user efficiency and the costs of development, operation and maintenance. Simplified, abstract, encapsulated and better organized data structures usually require additional operations performed by the system behind them. Many stages of data transformations on the way to a client may produce additional and sometimes unacceptable performance overhead. There is obviously the need for automatic optimizers [4], in particular, parallelization of computations [1], indexing techniques, query rewriting mechanisms and special physical data organizations. There may be also the need for special ,,golden rules" for programmers (c.f. Oracle SQL) encouraging or discouraging using some features of the system and its languages.

3 Grid Database Architecture

In our approach, a Grid Database consists of independent but cooperating database systems, which share data: publish and process objects. Clients use its virtual repository, which transparently integrates data and accomplishes a higher

abstraction level. The repository is based on two kinds of specialized views: contributory views defined on local servers and global views defined on the global level. Fig. 2 on the left presents possible (nested) architectures of a Grid database. Local servers (smallest rings) can be integrated into the first-level Grid (larger rings); then these Grids can be integrated into the global Grid (the biggest ring). Because virtual objects delivered by virtual repositories are not distinguishable from stored objects, the number of nesting levels of the Grid is (theoretically) unlimited. Each Grid node has the same conceptual architecture. If only desirable for the consortium, data integration process may also be divided into several stages. Fig. 2 on the right presents the architecture of a Virtual Repository Layer realized by stateless Global Views (GV) supported by a Grid Coordinator and a resources' Adapters Layer implemented by Contributory Views (CV).

A contributory view (sometimes called wrapper, mediator or adapter) is a view by means of which a node shares its own resources with the others. Its main task is to hide heterogeneity of local database systems (object, relational, etc.) within the consortium, by transforming local data models into unified data model specific for the consortium and to control access rights and hide data, which should not be published.

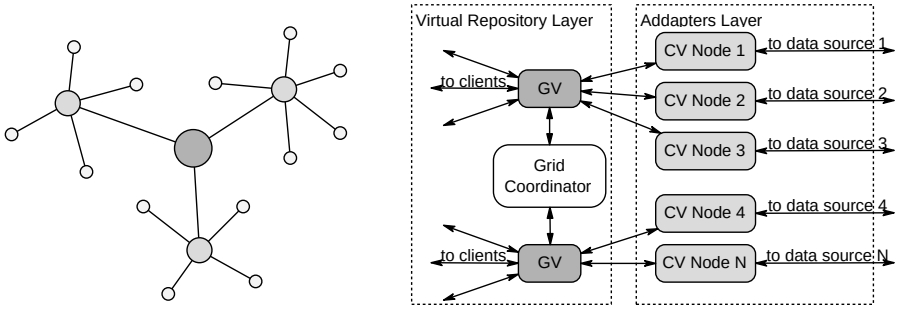

Fig. 2. Architecture of a Grid database

A Grid view (sometimes called mediator, integrator or fuse) is a view delivering virtual objects to users. Through this view, a user sees Grid resources adapted to his/her particular needs. All data transformations, optimizations and additional operations are hidden behind this view. Users see only resulting virtual objects created according to the established consortium's schema. Virtual objects created by the view do not exist in any concrete repository place, but are materialized on the fly, when needed upon distributed resources. Some optimizations (notably query modification [9]) cause that in fact some virtual objects may be never materialized.

A Grid coordinator plays additional role of a global coordination and global information administration. Generally, it is dedicated to: support system-wide optimizations (like query caching, indexing, query decomposition); control global

access rights (which probably should not be decentralized due to security requirements) and meta-database coordination (sending additional update information to distributed Global Views). There are many ways to organize this global coordination to be feasible and optimal for all users. It may be centralized, clustered or distributed according to the particular cost, time, complexity and other constraints.

4 The Process of a Grid Database Creation

The two mentioned aspects, functional and operational, reflect in the whole process of Grid database modeling and construction. The first one influences Updatable Object Views modeling and implementation, while the second one focuses on configuration of a Grid Coordinator. Especially the functional aspect is dependent on certain business strategy and cannot be accidentally solved. For this reason, we are skeptical about the feasibility of ad-hoc (or dynamic) integration solutions (except for perhaps very simple data structures) especially in the context of higher forms of transparency. Thus, for a serious project, a full development cycle is to be initiated by a consortium and precise rules obliging every participant are required. We assume the following Grid construction scenario:

1. Strategic phase, when the decision on creating a Grid is made and an initial analysis of the required content and potential participants are performed.
2. Analysis phase, when the existing resources are elaborated and confronted with the information requirements for the integrated service. The issues such as data heterogeneity, redundancy or incompleteness should be identified.
3. Design phase, which results in the precise definition of the global virtual schema and contributory schemes for each participant. The task of transforming local resources to the agreed contributory schema is delegated to an adapter of a particular node. On the other hand, the specification of mapping the contributions into the global schema needs to be specified as an integral part of the Grid design.
4. Finalization phase, when participants sign the final agreement formalizing the obligations coming from contributory schemes' specifications. Each participant gets unique identification and authorization codes.
5. Implementation phase, when the necessary data adaptation described by contributory schemas and global schema are implemented. Depending on the heterogeneity between a given node's data and global schema, such node may require either appropriate wrapper or may require in-depth restructuring of its original design.

The resulting specifications determine the required form of data provided both by the global virtual repository (the Grid itself), as well as by each of the participants. The task of adjusting local data into the form required by the Grid can be distributed between participants (that is, the administrators and developers of local systems) and the integrator (that is, the developer of the system serving

global, integrated data view). Although different ways of balancing this responsibility are possible, we assume that in order to simplify the integration itself, local sources should be obliged to adapt their contribution as far as possible, based on the knowledge of their local resources.

5 Conclusions

Updatable views allow us to reconsider architecture and transparency in distributed databases. Fully operable virtual objects present new quality among many approaches to virtual repositories. Unlimited composition of Contributory Views and Global Views allows us to achieve full transparency of data location, heterogeneity, fragmentation, indexing, replication and redundancy. More forms of transparency are possible if the system is provided with additional global coordination. That means migration, failure and optimizations transparency. Independence of local databases and distributed query evaluation based on additional meta-information provides concurrency transparency and scaling transparency (in case of horizontal fragmentation).

The presented architecture is currently being implemented in an experimental system ODRA, an object-oriented database management system devoted to rapid development of web and Grid applications.

References

1. E. Bertino: Query Decomposition in an Object-Oriented Database System Distributed on a Local Area Network. RIDE-DOM 1995: 2-9
2. S. Ceri, G. Pelagatti: Distributed Databases: Principles and Systems, 1984
3. I. Foster, C. Kesselman.: The Grid 2: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 2003
4. D. Kossmann: The State of the art in distributed query processing. ACM Comput. Surv. 32(4): 422-469 (2000)
5. H. Kozankiewicz, J. Leszczyowski, K. Subieta.: Updateable XML Views. Proc. of Advances in Databases and Information Systems (ADBIS), Springer LNCS 2798, pp. 385-399, Dresden, Germany, 2003
6. M. Lenzerini.: Data integration: a theoretical perspective. Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. Madison, Wisconsin. 2002
7. A.Y. Levy, A. Rajaraman and J. J. Ordille.: Querying Heterogeneous Information Sources Using Source Descriptions. Proceedings of the 22nd VLDB Conference, Mumbai (Bombay), India, 1996
8. M. Tamer zsu, Patrick Valduriez: Principles of Distributed Database Systems, Second Edition Prentice-Hall 1999
9. K. Subieta, J. Podzie.: Object Views and Query Modification. In: J. Barzdins, A. Caplinskas (eds.) Databases and Information Systems. Kluwer., pp. 3-14, 2001
10. K.Subieta. Theory and Construction of Object-Oriented Query Languages. Editors of the PJIIT, 2004, 522 pages, ISBN 83-89244-28-4 (in Polish)
11. I. Tatarinov et al.: The Piazza Peer Data Management Project. ACM SIGMOD Record, 32(3), 2003.
12. W3C Consortium: Web Services Documents [http://www.w3.org/2002/ws]