Open-Gate: An Efficient Middleware System for Heterogeneous Distributed Databases

Naglaa M. Reda
Faculty of Science
Ain Shams University

Fayed F. M. Ghaleb
Faculty of Science
Ain Shams University

ABSTRACT
Middleware has become an essential component for almost every distributed database system. It uses wrappers when integration is achieved for heterogeneity. Different middleware systems have been produced aiming for a better performance. In this paper a new middleware system for heterogeneous distributed databases (HDDBs) called Open-Gate is proposed. Its main objective is to provide an efficient system with the characteristics of autonomy, scalability, reliability, and high performance. In addition, it can handle a huge number of users overcoming the bottleneck problem or loss of user's queries. Experimental results show that, the proposed system achieved high performance compared to other systems.

General Terms
Database, Distributed computing.

Keywords
Heterogeneous distributed database; middleware; wrapper.

1. INTRODUCTION
Currently middleware is an essential component for almost any type of distributed environment and networked applications. Starting from the hardware infrastructure and run-time support all the way up to the applications, middleware solutions provide endless possibilities to support applications requirements both functional (i.e. the main processes of the system that will generate the required outputs) and non-functional (i.e. those that define how the system will perform in terms of overall performance, reliability, scalability, etc.).

Database middleware systems [1], [2] are used to integrate heterogeneous data sources dispersed over a computer network. They impose a global data schema on top of the individual schemas used by each source. This mechanism provides user applications with a uniform view and access interface to the data sets stored by each data source. The translation of the data items to the global schema is performed by either a wrapper or database gateway. Wrappers are used when integration is achieved through a mediator system, such as COIN [3], TSIMMIS [4], DISCO [5], Garlic [6], MOCHA [7], SQMD [8], AMIS [9], OGSA-DAI [10], and OGSA-DQP [11]. Unfortunately, most mediators suffer from the bottleneck problem. This paper proposes a new middleware system Open-Gate that takes advantage of the benefits provided by using the wrapper (IWRAP) established in [12] to handle this problem properly, in addition to accomplishing other important features.

The rest of this paper is organized as follows: Section 2 discusses the needed system environment. Section 3 describes the architecture of the proposed middleware system. Section 4 presents a detailed discussion on the system's storages. The system components including the manager, the global integrator, the query processor and the data collector are discussed in Section 5. The performance evaluation is summarized in section 6. The main characteristics of the system are introduced in Section 7. Conclusions are given in Section 8.

2. THE SYSTEM ENVIRONMENT
Since Open-Gate needs a special environment to work with. In the following the elements that suites the system is discussed.

2.1. Users Interfaces
Generally, interfaces for users of the system to access data in HDDBS must be provided. Mostly, the interfaces include GUI [13] for users to pose queries against the database schema and visualize their results. It also includes APIs [14] for programmers to develop complicated applications. Queries are formulated in a generic form and evaluated by the system. So the Open-Gate system parses the issued queries and provides it with the distribution information from the integrated schema, accesses the data sources, gets requests throughout IWRAP, and convert them into the format that can be displayed to the users. Open-Gate provides users with two privileges. First, they are not concerned of the location of the data and what is done in the middleware system. Second, they don't have to learn a new query language. In addition, their queries will never be lost. And that is why it is called Open-Gate.

2.2. Data Sources
Open-Gate is developed for heterogeneous distributed data sources that have different operating systems, DBMSs, or query languages. Each data source must have the IWRAP wrapper. IWRAP acts as an interface for the middleware that interacts with each data source and fetches data result. It also provides other especial services such as query translation and schema integration which improve the system overall performance.

3. THE PROPOSED OPEN-GATE ARCHITECTURE
The Open-Gate's main goal is to provide users with an efficient, reliable, scalable, and autonomous HDDB middleware system with high performance. This is achieved by the new proposed middleware system introduced in the following. It consists of four different components.

- The manager: it plays the role of controlling. It manages the work of the other components composing the Open-Gate and the storages. It is also the basic component that deals with users and coordinates between their requests.
- The global integrator: its main job is to collect the local schemas from each IWRAP and then integrates them forming a global schema. Any change in any local schema will replicated by this component to form a new global schema.
- The query processor: it accepts queries from the manager. Each query that enter this component must be subjected to some processing, translating, fragmenting, optimizing, and deciding for each fragment what IWRAP it should be sent to.

• The data collector: it is responsible for composing the final data result that must be sent to the requesting user. This is done by collecting the partial data resulting from every IWRAP and emerging them.

Also, the system needs two storages. The storage queue is used to store queries. And the data cache storage is used to store data results. The architecture showing the interaction between the system components, the storages, the users, and the wrappers is presented in fig. 1.

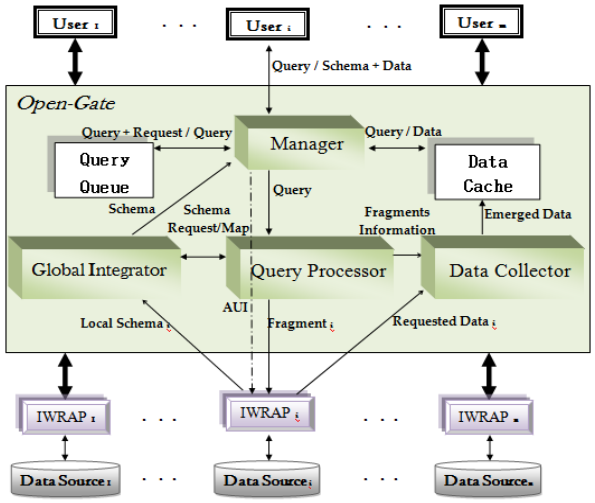

Fig. 1: Open-Gate architecture

4. OPEN-GATE STORAGE

Open-Gate has a new feature that doesn't exist in other systems. It preserves queries that have been processed with their resulted data. So if any user poses one of these queries, the system will directly replay the stored data without reprocessing. Another feature of Open-Gate is storing queries in a queue for keeping them from lost and preserving their ordering. In the following, an explanation of the storages needed to accomplish these features is presented.

4.1. The Query Queue

The query queue (QU) is a repository for queries that are sent by users. It is used by the system to make a control on all queries sent by users simultaneously and preserves them from lost. It organizes queries depending on the principle of "First in First out", but it gives a priority to the administrator. This is done by saving administrator's queries marked by the manager component in a separate repository to distinguish them from the ordinary ones. QU Interacts only with the manager component. It receives queries and verifies for their existence in its repository SQ. If a query doesn't exist then QU save it after giving it an ID number specifying its user so that each user can get his requested query. On the other hand, if a query exists in its repository, it simply leaves it to take its role for processing and adds the new user's ID to its IDs list. Marked queries are saved in another repository SQ* in its role. Whenever the manger component asks for a query to be processed, QU gives it the first query in SQ* if it exists, otherwise it returns the first query in SQ. These services are done by the following steps.

• Whenever a query Q is sent from the manager component (MG), QU accepts it.
• If the query Q is marked then QU inserts it in its role at the end of the administrators' repository SQ*.
• For not controlled query Q, it verifies if Q doesn't exist in the repository SQ, then it saves Q in its role at the end of the repository SQ with its ID number. Otherwise QU saves the ID of the new user in the list L containing other user's IDs that poses Q.
• As MG sends a request, QU replies the first query existing in its administrators' repository SQ* if it is not empty, otherwise QU returns the query at SQ front.

The proposed algorithm for achieving these services is shown in Algorithm 1.

ALGORITHM 1 Query queue
Input: Query Q, user' ID ID, request R
Output: Query Q
1: Receive Q from MG
2: **if** Q =Q* **then**
3: Enqueue (Q, SQ*)
4: **else**
5: **if** Q Not in SQ **then**
6: Enqueue (Q , SQ)
7: Insert (ID , L)
8: **end if**
9: **end if**
10: Receive R from MG
11: **if** not Empty (SQ*) **then**
12: Dequeue (Q, SQ*)
13: **else**
14: Dequeue (Q, SQ)
15: Send Q to MG
16: **end if**

4.2. The Data Cache

The data cache storage (DCH) is a huge memory for storing queries and its data result; where each query entered the mediator is stored in DCH with its user's IDs list L. Open-Gate uses it for two main purposes. First, it works as a store for all processed queries with their data so that when MG requests one of them, it simply passes its saved data.

Second, it works as a mediator between the data collector component (DTC) and the manager. As the collected data result is received from DTC, saved with its current processed query, and passed to MG by DCH. This work is accomplished by the following steps. The DCH steps are given in algorithm 2.

ALGORITHM 2 Data cache
Input: Query Q, user' ID ID, data result DR
Output: data result - DR
1: Receive Q from MG
2: **if** Q = _ **then**
3: Clear DCH
4: **else**
5: **if** Q 2 DCH **then**
6: Get DR from DCH
7: **else**
8: Save Q in DCH
9: Insert (ID, L)
10: Receive DR from DTC
11: Save DR with its Q
12: **end if**
13: Send DR to MG
14: **end if**

• When DCH receives a query from MG, it tastes the query if it is empty then it clears its repository.
• Otherwise, it searches in its repository for this query, If the query exists in DCH's repository then its data is sent to MG

to be passed to its suitable user, otherwise it is saved in the respiratory, and inserts the user's ID in L.

- After the query is processed, and the data is collected by DTC, DCH receives this final data result from DTC.
- Finally, DCH saves this data result with its query and sends it to MG for sending it to its requester.

5. OPEN-GATE COMPONENETS

As mentioned above, Open-Gate is composed of four main components: the manager; the global integrator; the query processor; and the data collector. This section presents a detailed study of them including their algorithms.

5.1. The Manager

The manager component MG is the principle component of Open-Gate. It manages the overall work of the system. For users, they interact with the system through MG using GUI or API interfaces to pass their queries and get their results. Also any change happen to the schema is reflected to them. As for administrators, they send their control queries to MG to be marked and processed directly. While for the global integrated schema (GS), it is always sent from the global integrator component (GI) to provide MG with the up-to-date copy that support users' interfaces with it. Also for AUI, it is sent to each IWRAP by MG with an ID for each user to control the whole process. And for storages, MG clears the data cache storage DCH if it receives an empty query. Then it cheeks-up DCH for each entered query, if it exits then MG returns the data result to the user request, and stores the query with its data in DCH. Otherwise MG sends this query to the query queue storage QU to stand in line, save a copy in DCH, and add the user's ID to its ID's list L. Finally for query processing, MG gets queries from QU one by one for processing, sends them simultaneously to the query processing component (QP) for fetching the desired data, and gets the collected data result from the data cache DCH to send them to the user.

The following steps show how MG does his work properly. These steps are illustrated in an algorithm 3.

- It receives GS from GI component for providing a copy of its global schema to the users' interfaces.
- It sends to every IWRAP on each data source, AUIs of users, and provides each user with an ID number.
- If it receives a new GS, then it sends an empty query to DCH indicating that the GS has changed, and sends a copy of GS to user's interface.
- When it gets a query Q, it checks if Q is from the administrator then it is marked, and sent to QU.
- If the query is not from admin. It sends a copy of Q to DCH with the user's ID.
- If a data result (DR) is received from DCH, then it returns DR to the users, otherwise, it sends the query to take its role in QU.
- Then it sends a request R to QU, gets a query Q' from QU and sends it to QP for processing and fetching data result from data sources.
- Finally, it gets data result from DCH to be sent to all users whose ID is in L. Then disconnect.

ALGORITHM 3 Manager

Input: Query Q, user' ID ID, global schema GS, data result DR
Output: Global schema GS, query Q, data result DR

```
1: Receive GS from GI
2: Connect DTS via IWRAPs
3: Send AUI to IWRAPs
4: if Receive GS from GI then
5:     Send Q = _ to DCH
6:     Send GS to user interface
7: end if
8: Receive Q from user with ID
9: if ID = administrator's ID then
10:    Mark Q=Q*
11:    Send Q to QU
12: else
13:    Send Q and ID to DCH
14: end if
15: if Receive DR from DCH then
16:    Go to step 23
17: else
18:    Send Q to QU
19: end if
20: Send R to QU
21: Receive Q' from QU
22: Send Q' to QP
23: Receive DR from DCH
24: for all id in L do
25:    Send DR to user ID
26: end for
27: Disconnect DTS
```

5.2. The Global Integrator

The global integrator component (GI) includes the global conceptual schema which comes from integrating all local schemas distributed over sites participating in the distributed system. GI includes the information of data distribution such as address and specifications of vertical and horizontal fragmentations (schema map) for allocating it at each site in the system. It also includes the access interface information for each data source that enables the users to transparently access the data sources no matter if they are stored remotely or locally or can be accessed by means of SQL or APIs. With this mechanism, users serviced by the middleware system are provided with a uniform view and access interface to the data stored on each data source. When GI gets the local schemas from each data source throughout its wrapper IWRAP, it gives an ID for each wrapper to connect each one with its schema and its information. After receiving all local schemas, GI merges these schemas to form the global schema. Any changes in any local schema are transformed to GI to change the global schema. Whenever a data source is disconnected or added it will be reflected on the global schema by GI. The new global schema is always sent to MG to provide interfaces with the new global schema to make users up to date with data sources. Also data sources' information and location of data are sent to QP. GI accomplishes its service by the following steps and Algorithm 4 shows how this is done.

- It receives the local schema of each data source.
- It saves in its database all information and localization of data for each data source.
- It ingrates these local schemas forming a global schema.
- It sends the global schema to MG.
- When any change happen in any local schema, GI well update the global schema, and send it to MG.

• When it receives a schema request (SR) from QP, it sends a schema map (SM).

ALGORITHM 4 Global integrator

Input: Local schemas LS, no. of wrappers n, Schema request SR
Output: Global schema GS, schema map SM

1: **for** i:=1 to n **do**
2: Get LS i from IWRAPi
3: **end for**
4: Integrate LS i:1→ n to GS
5: Send GS to MG
6: **if** change LS i:1→ n **then**
7: Receive updated LSi
8: GOTO step 4
9: **end if**
10: Receive SR from QP
11: Send SM to QP

5.3. The Query Processer

The query processer component (QP) is the middle-tier component that controls the execution of all the queries and commands received from the user throughout the manager component MG. The main goal of QP is decomposing the query into fragments and finding the corresponding location for each data item which can handle the request. The choice of the best fragmentation and execution for the query is done by applying special optimization techniques. Fragmentation process is also guided by information about the data sources and the schema map which are requested from the global integrator component GI. This information is also sent to the data collector component DTC as a base to the process of recomposing results.

As all query processing component; QP provides the services: query parsing; query optimization, and query execution. When a query is entered from client as standard SQL query or user's familiar query language, QP parses it to an algebraic expression, and then it provides an optimal query plan that will be executed efficiently and gets the wanted data from its data sources. Algorithm 5 represents the QP's work.

The main tasks of QP are illustrated in the following steps.

• It receives a query from MG component for processing.
• It requests the global integrator component GI to get the last copy of schema map SM to be used for choosing the optimal fragmentation for the query.
• It processes the query by translating the query to an algebraic form, fragmenting the query, and specifying the suitable location for each fragment.
• It sends query fragments to its specific IWRAP.
• Finally, it sends the fragmentation information to the data collector component DTC.

ALGORITHM 5 The query processer algorithm

Input: Schema map SM, query Q
Output: Schema request SR, query fragments QF, no. of wrappers n, fragments information FI

1: Receive Q from MG
2: Send SR to GI
3: Receive SM from GI
4: Process Q
5: **for** i:=1 to n **do**
6: Send QFi to IWRAPi
7: **end for**
8: Send FI to DTC

5.4. The Data Collector

The data collector DTC plays a serious role in the Open- Gate system. Its main aim is to collect the data result from all IWRAP wrappers, merge them to form one data result, and send this combined data result to DTC, which in turn sends these data to the requested users throughout MG.

While merging the partials data received from each wrapper data results for one fragmented query, DTC depends on the information of fragmentations that is sent from QP. It also uses some of union operations and clustering algorithm [1]. This is done by the following steps. The data collector algorithm is presented in Algorithm 6.

ALGORITHM 6 Data collector

Input: Fragment information FI, no. of wrappers n, partial data results PDR
Output: data result DR

1: **if** FI **then**
2: Receive FI from QP
3: **end if**
4: **for** i:=1 to n **do**
5: Receive PDRi from IWRAPi
6: **end for**
7: Merge PDRi to DR
8: Send DR to DCH

• When there is fragment information FI, DTC receives it from QP.
• Then, it receives the partials data results from each IWRAP that has a data.
• As DTC has all parts, it merges them to form the final data result for this query.
• Finally, it sends the final data result to DCH.

6. PERFORMANCE EVALUATION

The performance of the introduced middleware system Open-Gate has been evaluated practically. All of its components have been built by the Java 6 using Eclipse Galileo version [15]. IWRAP was installed on six deferent data sources using (Linux-Redhat11 and windows-XP) operating systems. Their random access memory is 2 GB; their hard disk storage is 80 GB, with speed 2000/3000 MHZ. The amount of data Distributed over the six data sources was about 70 GB and the origin middleware system was AMIS [9]. They include data sets describing information of employees, products, stores, and agents for companies. Some of these data sources use the object oriented database system Oracle while the rest uses the relational database system SQL-server. As a case study, the chosen standard type of the integrated schema was the object oriented. Also the entered queries were relational ones including different functions such as select, project, join, and aggregate functions. Various queries have been experimentally tested. For comparison, the processing time has been calculated for the two systems as the number of data sources increases from 1 to 6. Figure 1 shows their average performance variation.

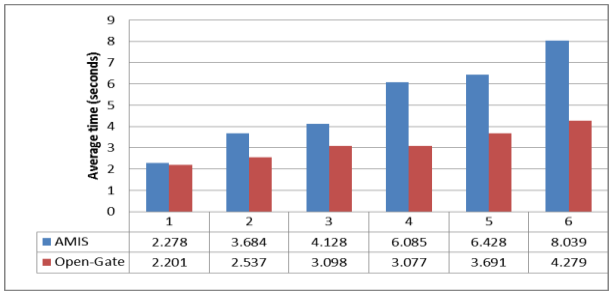

Fig 1: Performance comparison

Performance measurements prove that the Open-Gate average performance is much better whenever a new data source is added. This is because of two reasons. First, it takes advantage of the computer power of each data source by partitioning the schema integration or query translation computations over all DTSs. Second, it doesn't suffer from network bandwidth as the AMIS does since it eliminates most of communications between the wrapper and DTSs.

7. OPEN-GATE CHARACTERSTICS

The Open-Gate system has the characteristics that distinguish it from other systems. It accomplishes the most important promises of DDBS and overcomes some of its challenges. The following subsections discuss how it does so.

7.1. Autonomy

Autonomy [1], [2] refers to the capacity of a component to choose its own design and operational model. An autonomous component is able to define the data that it wants to share with other components, define the representation and naming of the data elements, determine the way that it views the data and combines existing data, decide when to enter or leave a federation, specify whether to communicate with other components, execute local operations without interference from external operations, add new data and withdraw access to any shared data.

Open-Gate is an autonomous system as it achieves the above privileges. These privileges are provided by the GI component. Since it gives the data source administrator the whole responsibility to decide which data will be participating in the global schema as he sends the local schema. Thus it frees the data sources from concerning about its private data and how to keep them far away from users accessing.

7.2. Scalability

The most crucial challenge that the design of an efficient middleware in a distributed environment face is the development of a distributed architecture for managing large amounts of data stored in geographically distributed resources. The efficiency of the system should not decline when the amount of stored data and the number of data sources increase. The system should scale well, providing reliable and concrete services [16].

Some middleware systems like DISCO [5] scale well. It provides special features for all users to deal with the problems of scale. For the application programmer and end user, it provides a new semantics for query processing to ease dealing with unavailable data sources during query evaluation. For the DBA, it models data sources as objects which permit powerful modeling capability. In addition, DISCO supports type transformations to ease the incorporation of new data sources into a mediator, and provides a flexible wrapper interface to ease the construction of wrappers. Also SQMD [8] focuses on the issue of data scalability with the software architecture and virtualization technology. But other systems such as MOCHA [7] don't achieve scalability. As a new site is added, it must manage system-wide interactions.

On the other hand, the proposed Open-Gate system handles the problem of scalability properly. Since the GI component always gets local schemas from each data source. So when a new data source is added to the system, it instantly sends its local schema to GI after installing IWRAP on it. And GI by his turn integrates this new schema with other schemas and directly composes the updated global schema. This new global schema is directly sent to the manager who simply passes it to all users.

7.3. Reliability

High reliability is critical for distributed systems. However, achieving it at a reasonable cost can be a challenge, especially in large-scale. Even when the devices are fairly reliable, failures are frequent in such systems due to the large number of devices they employ. Some researchers concentrate their efforts to achieve high reliability like in [17].

The proposed system handles this challenge directly. When a data source failed, it is instantly recognized. As the GI component regularly inquires about any changes happening in any of the data sources. If a data source failed then GI directly omitted its local schema from the global schema, passes the new global schema to the manger MG, to inform the users.

7.4. High Performance

In the last few years efforts have been made in enhancing the performance of DDBSs by concentrating on the key challenges in DDBs performance which are data allocation and high communication cost. Several kinds of distributed approaches have been implemented that reduce the amount of data transferred during the run time, speeds up the database system by maximizing the degree of parallel execution, minimize the communication cost, and increase data availability and integrity by replication of fragments where possible [18].

Open-Gate handles these challenges perfectly by installing IWRAP on each data source to minimize the communication cost and speed up the system as explained in [12].

Another factor that badly affects the system performance is growing population of users which is not comparable to the processing done. This situation causes significant congestion around the communication network leading to the bottleneck problem [19]. Open-Gate handles this situation perfectly. First, the burden of query translation has been distributed over the IWRAPs of the data sources. This reduces the load of processing on the QP component and speeds up the processing time which in turn leads to a higher performance compared to other systems like AMIS [9] as proved experimentally in previous section. Second, the use of QU component decreases the load on the QP component by organizing queries. Also a better performance is expected due to the data cache storage DCH, since it replies data results to users directly for its stored queries. Thus it saves the time of processing.

8. CONCLUSION

In this paper, Open-Gate a new middleware system for efficient management and integration of heterogeneous distributed data resources was proposed. The introduced architecture uses the most promising wrapper IWRAP with each data source to minimize the communication cost. It also avoids the centralization of query translation by distributing this burden over the multiple IWRAPs. This technique alleviates the congestion bottleneck and speed up the system.

In addition, the system preserves processed queries with their resulted data in data cache storage so as not to reprocess any of them when possessed by other users. This new strategy makes the system serves more users at less time. It also preserves queries from lost by using queue storage.

Experiments prove that Open-Gate average performance is much higher compared to AMIS system. It also scales very well as the number of data sources increase. Moreover, analysis of other existing systems indicates that the proposed architecture provides a reliable, scalable and autonomous infrastructure that meets the evolving user's requirements.

In the future, it is planned to extend Open-Gate system to become a full-scale comprehensive DDBMS that implements the functionality and techniques proposed in recent DDB research, and emerge it as a commercially viable product. As well as testing it on valuable and huge databases such as those of Bioinformatics.

9. REFERENCES

[1] M. Tamer Özsu, "Principles of distributed database systems", 3rd ed., Prentice Hall, USA, Jul. 2007.

[2] Ling Liu, M. Tamer Özsu, "Encyclopedia of Database Systems", Springer US, USA, Sep. 2009.

[3] H. Zhu, S.E. Madnick, "Context interchange as a scalable solution to interoperating amongst heterogeneous dynamic services", in 3rd Workshop on eBusiness (WEB), Washington, DC, 150-161, Dec. 2004.

[4] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A.Rajaraman, Y. Sagiv, J.D. Ullman, V. Vassalos, and J.Widom, "The TSIMMIS Approach to Mediation: Data Models and Languages", in Journal of Intelligent Information Systems, vol. 8(2):117-132, Mar. 1997.

[5] Eric Freudenthal, Vijay Karamcheti, "DisCo: Middleware for Securely Deploying Decomposable Services in Partly Trusted Environments", in proceedings of 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04), 494-503, Japan, Mar. 2004.

[6] M. Tork Roth, et al., "The Garlic Project", in proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, 557-558, Montreal, Jun. 1996.

[7] M. Rodriguez-Martinez and N. Roussopoulos, "MOCHA: a self-extensible database middleware system for distributed data sources", in proceedings of the 2000 ACM SIGMOD international conference on Management of data, Texas, Jun. 2000.

[8] Kim, K., R. Guha, M. E. Pierce, G. C. Fox, D. J. Wild, and K. E. Gilbert, "SQMD: Architecture for Scalable, Distributed Database System built on Virtual Private Servers", in proceedings of the 4th IEEE International Conference on eScience, 658-665, Indianapolis, Dec. 2008.

[9] Don Libes, Edward J. Barkmeyer, et al., "The AMIS Approach to Systems Integration: An Overview", National Institute of Standards and Technologies, Manufacturing Systems Integration Division, May 2004.

[10] Xuhong Liu, Yunmei Shi, Yabin Xu, Yingai Tian, Fuheng Liu, "Heterogeneous Database Integration of EPR System Based on OGSA-DAI", in High Performance Computing and Applications LNCS, 5938: 257-263, Mar. 2010.

[11] Helen X. Xiang, "Integrated Queries over a Heterogeneously Distributed Scientific Database using OGSA-DQP", in proceedings of the 6th IEEE Joint International Information Technology and Artificial Intelligence Conference (ITAIC), 421-425, Chongqing, Agu. 2011.

[12] Abdullah F. A. Sebai, Naglaa M. Reda, Fayed F. M. Ghaleb, "IWRAP: An intelligent wrapper for distributed heterogeneous database systems", in Egyptian Computer Science Journal, 34(5):79-90, Sep. 2010.

[13] Konstantinos Liakos, Albert Burger, and Richard Baldock, "A Scalable Mediator Approach to Process Large Biomedical 3-D Images", in IEEE Transactions on Information Technology in Biomedicine, 8(3):354-359, Sept. 2004.

[14] Arvind S. Krishna, Aniruddha S. Gokhale, Douglas C. Schmidt, "Context-specific middleware specialization techniques for optimizing software product-line architectures", in proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems, 40(4): 205-218, NY, USA, Oct. 2006.

[15] Java developer, at http://www.eclipse.org

[16] A. Asiki, K. Doka, I. Konstantinou, A. Zissimos, D. Tsoumakos, N. Koziris, and P. Tsanakas, "A grid middleware for data management exploiting peer-to-peer techniques", in Future Generation Computer System, 25(4): 426-435, Apr. 2009.

[17] R. Bachwani, L. Gryz, R. Bianchini, C. Dubnicki, "Dynamically Quantifying and Improving the Reliability of Distributed Storage Systems", in proceedings of the 27th International Symposium on Reliable Distributed Systems (SRDS), 85-94, Naples, Oct. 2008.

[18] I. O. Hababeh, M. Ramachandran, N. Bowring, "A high-performance computing method for data allocation in distributed database systems", in Journal of Supercomputing, 39(1):3-18, Jan. 2007.

[19] Saumitra M. Das, Himabindu Pucha, Y. Charlie Hu, "Mitigating the gateway bottleneck via transparent cooperative caching in wireless mesh networks", in Journal of Ad Hoc Networks, 5(6):680-703, Aug. 2007.