 DOI: 10.1515/umcsinfo-2015-0001

Heterogeneous Data Integration Architecture-Challenging Integration Issues

Michal Chromiak[1*], Marcin Grabowiecki

[1] *Institute of Computer Science, Maria Curie-Sklodowska University, pl. M. Curie-Sklodowskiej 5, 20-031 Lublin, Poland*

Abstract – As of today, most of the data processing systems have to deal with large amount of data originated in numerous sources. Data sources almost always differ regarding its purpose of existence. Thus model, data processing engine and technology differ intensely. Due to current trend for systems fusion there is a growing demand for data to be present in common way regardless of its legacy. Many systems has been devised as an answer to such integration needs. However, the present data integration systems mostly are dedicated solutions that brings constraints and issues when considered in general. In this paper we will focus on the present solutions for data integration, their flaws originating in their architecture or design concepts and present an abstract and general approach that could be introduced as an answer to existing issues. The system integration is considered out of scope for this paper, we will focus particularly on efficient data integration.

Keywords: *grid integration model, heterogeneous integration, distributed architecture, data integration, big data, distributed transaction, warehouse, ETL, OLAP*

1 Data Integration Role

Integrating distributed data and service resources is an ultimate goal of many current technologies, including distributed and federated databases, brokers based on the CORBA standard [**1**], Sun's RMI, P2P technologies, grid technologies, Web Services [**2**], Sun's JINI [**3**], virtual repositories [**4**], metacomputing federations [**5, 6**] and perhaps others. The distribution of resources has desirable features such as autonomic maintenance and administration of local data and services, unlimited scalability due to many servers, avoiding global failures, supporting security and privacy, etc. On the other hand, there is a need for global processing of distributed resources that treats them as a centralized repository with resource location and implementation transparency. Such integration is needed especially when considering domains (e.g. spatial) with large amounts of data that is required for further analysis [**7**] and optimisation [**8**], but is originated from many sources. Distributed resources are often developed independently (with no central management) thus with the high probability they are heterogeneous, that is, incompatible concerning, in particular, local database schemas, naming of resource entities, coding of values and access methods. There are methods to deal with heterogeneity, in particular, federated databases, brokers based on the CORBA standard and virtual repositories. The integrated data context is essential in terms of understanding the domain. The more the data is isolated from colligated data, the less informative it is. In other words, data Integration plays a significant role, as the data puzzle is meaningful only while it is considered in particular context. Therefore the integration is so crucial for modern systems. Present information systems must deal with large amount of data. Thus, the big data concepts are becoming particularly important. The data itself in most cases is scattered, uses different models or query engines, is stored in different locations and administered by independent administrators. In scope of data integration this situation is undesirable. At present, despite of many dedicated solutions for data integrations (see section 3) or global caching [**9**], most of them are not considered general enough to provide an abstract layer that could unify all of the data integration on ground of general solution.

The rest of the paper is organized as follows. We focus on problems during data integration and common models that address them in Section 2, existing data integrations solutions and their issues in Section 3. We also propose an abstract approach that would solve the issues present in Section 4. Section 5 concludes.

2 The Models of Integration (Enterprise Design Patterns)

Integration may continue using multiple techniques utilizing their each of their particular advantages where needed. However, it is believed that asynchronous messaging technique plays an increasingly important role among other styles. Apart from asynchronous messaging, there are other approaches that solve the same problem,

*mchromiak@umcs.pl

 DOI: 10.1515/umcsinfo-2015-0001

each has its distinct advantages and disadvantage. In general approaches to integration can be considered in order of evolution:

- shared file - one party writes to a file and other reads from it. File contact details, format and access timing must be agreed between parties using it.
- shared database - one database is a data source and store for many applications
- Remote Method Invocation - one party exposes to the world its interface to inner procedures that can be later executed form outside of this party. The communication is real-time and synchronous.
- asynchronous messaging - one party publishes messages[1] to a common message channel and the other parties can read those messages at later time from the channel. However, the channel and the message format must be agreed.

3 The Collation of Issues With Data Integration Solutions

Current research in the field of database integration is focused around detailed areas without general approach that would be a complete (i.e. independent, extensible by design) and tested concept. The solutions that we propose in this paper bring the flexibility and abstraction layer enabling unrestricted design for networking and schema customization. The existing research in the domain of integration of heterogeneous databases has provided a diverse array of solutions. The main heterogeneous DB integration issues that are being answered in this research field has been briefly classified in the past 1995 in [**10**] and [**11**]. However, some attempts has take place even earlier in 1983 [**12**]. At present, the main area of interest regarding the integration techniques is basing on XML solutions. It is believed, as mentioned in [**13**], that the XML has become the indisputable standard both for data exchange and content management, and moreover that is about to become the lingua franca of data interchange. This point of view can be justified by immense number of research papers, like [**14, 15, 16**] and many more, trying to utilize the XML technology as the tool for database scheme and data representation. The XML has also become part of many commercial products supported by giants of the software industry like Microsoft [**17**], Oracle [**18**] or IBM.

Another aspect of significance of the integration research should be considered in the field of enterprise. Since 1992 [**19**] the concept of data warehouse has been proposed, and the database vendors have rushed to implement the functionalities for constructing data warehouses. That made on-line analytical processing (OLAP) emerge as a technology for decision support systems. However, problems may arise in building a data warehouse with pre-existing data, since it has various types of heterogeneity. This integration scheme for data warehouse however had to focus on some conflicts, namely value-to-value, attribute-to-attribute and table-to-table. These conflicts are not exclusive, they may occur in any pair of relations at the same time. Such heterogeneity occurs frequently in two distinct pre-existing databases, when different databases are designed by different designers or driven by different assumptions. This conflicts are resolved only partially or are considered out of scope.

The idea of the presented projects is aimed to solve all of the issues and prepare monolithic solution. While in case of published achievements the results can be accessed and classified freely there is still a considerable amount of closed, enterprise solutions that are mainly dedicated for commercial DBMS. In this paper we try to challenge and solve problems we further elaborate in following sections.
OLAP OLAP is a decision supporting software that gathers data in multidimensional structures (hypercubes). It also enables the analysis (statistical, sale, financial, etc.) of collected data. OLAP solutions are populated with data from heterogeneous sources i.e. multiple vendor and models (databases, pliki, services, etc.). Data acquisition and transformation is done by ETL tools. The design concept of OLAP is to collect analyzed data fast and effectively. Unfortunately there is no formally unified query language for OLAP implementations1. There are three types of OLAP systems - ROLAP (relational), MOLAP (multidimensional) and HOLAP (hybrid). The indisputable advantage of OLAP is fast analysis of historical data, collected periodically from data sources and aggregated in form of data storing structures.

However, it must not be forgotten that data acquisition and transformation is a tedious and time consuming process. This utilize hardware resources intensively regardless of actual needs of analytical process client. The fact of often unnecessary data collecting from data sources bring overhead to networking and contributing systems when considering big data. Another problem would be the fact that the OLAP analytical processes almost always are based on outdated data. This is due to continuous data changes in contributory systems and periodical data acquisitions to OLAP. Therefore OLAP client is reported with historical results. It is not a problem in all systems but it is a serious flaw when

[1] Message is data structure - such as a string, a byte array, a record, or an object. It can be interpreted simply as data, as the description of a command to be invoked on the receiver, or as the description of an event that occurred in the sender

 DOI: 10.1515/umcsinfo-2015-0001

considering current data requests.

ETL Modern enterprise solutions, mostly base on ETL tools. Modern information systems often require input from multiple data sources (databases, xml files, csv files, web services, etc.). What is more, the integration of data brings possibilities of data transformation and analysis. Therefore, the data needs to be extracted, transformed and loaded to the destination system which usually is a data warehouse. ETL (Extract Transform Load) meets those needs. There are numerous ETL solutions originating in enterprise[2] and open source[3] vendors, all sharing the same model. The "Extract" phase is challenging as needs to handle multiple data sources. The databases themselves are different regarding their model: relational, object, document, graph, column-oriented, etc. Only relational databases include many SQL dialects and what is more the NewSQL is coming. Finally, the web applications can sometimes be also the data source (web spidering). The "Transform" phase modifies the extracted data. Simple tasks such as selection of specific columns, column joining, data filtration (selecting data from formula based ranges), formula based data transformations, aggregation, table joining, sorting, etc. Some more complicated tasks are also pivoting or disaggregation of repeating columns into a separate detail table. At this stage the data types are processed (eg. data interpretation from string values). The last phase is data "Load" to data warehouse. Depending on strategy, new data can be periodically overwritten or new data is written while providing timestamp of data aggregation.

The ETL plays a great role for the specific architecture that it represent. However, ETL is not general or abstract and moreover, has some quite significant issues. Let us focus on the conceptual problems of ETL. The fundamental issue with ETL is that the model, where the data collecting is periodical, what brings the danger that when the data goes to data warehouse it is already outdated or will soon become such. ETL ignores fact that the data sources keep changing on regular basis. Collecting data takes place only at particular time moments i.e. once a day, once a week, etc. In result data warehouse most of the time contains historical data. For applications where the data can be historical this is not an issue. However, when the online data processing is required ETL is unacceptable.

Another architectural concept of ETL is unavoidable data pulls, regardless whether the data are to be viewed or analyzed or not. This can generate redundant network chatter and increase data source systems load, often unnecessarily. One needs to remember that often data load is transactional and therefore reliable and consistent but also requires large resource request per transaction.

[2]Oracle Data Integrator, SQL Server Transformation Services, IBM InforSphere DataStorage

[3]Talend Open Studio, Pentaho Data Integration

4 Proposed Abstract Integration Solutions

Answering the need of a general and abstract architecture for data integration, we introduce some of the techniques that we have used to overcome the issues mentioned in previous section, while still preserving the key functionalities of exemplary solutions. The proposed architecture depict in the Figure 1 is based on processing metadata instead of the actual data itself. The architecture brings the following advantages by design:

(1) efficiency - metadata instead of real data
(2) manageable - easy to manage due to only structures manipulation
(3) traffic optimization - less network connections
(4) reliability - less data to transfer; not relying on network reliability
(5) optimized - uses only native queries; brings the most of native data source optimizations
(6) no distributed transactions

Below we discuss the reasons for applying the following assumptions and justify their usage in the architecture depicted in Figure 1.

4.1 Solution Proposal for the ETL issues

The ETL issues discussed above can be overcome thanks to a general architecture particularly design thanks to its data-less model. The key is to provide the data that are up to date with the source state. To achieve this i.e. obtain the data same as the data stored in data source, we have devised an architecture that provides not the data itself but the fast access method for the requested data. The Fast Access Method (FAM) would be the data source native query, that would pull the data in the fastest possible way the data source can provide. Such query would differ across different models. For instance, in relational model the fastest method for reaching the data is calling them by their primary key, in object database it would be the OID (object ID), etc. Basing on this idea, we utilize a dedicated metamodel [**20**] that is based not on the data itself but on the complex metadata that also includes the FAM. The metamodel for representing the integration patterns is presented in [**21**] and is out of scope of this paper, however, it enables the means to obtain a pure native query (FAM) from a metadata model to pull the actual data from the data source. Such data are not only current but are also pulled effectively. For example. Let us assume the goal of extracting about a billion of

DOI: 10.1515/umcsinfo-2015-0001

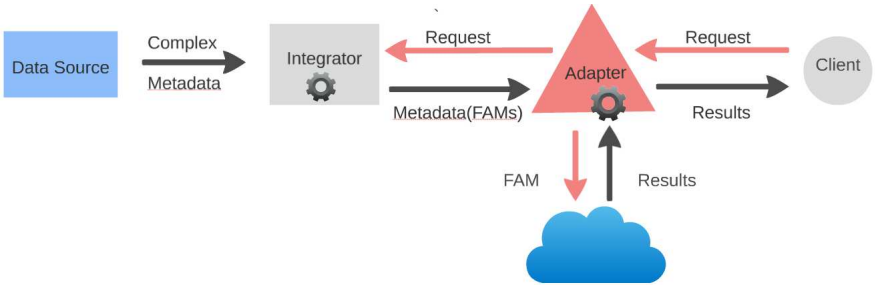

FIGURE 1. Proposed architecture for the heterogeneous data integration grid

records using not optimized query. This would bring major latency issues and could even cause system crash, which is not acceptable. A native query basing on the best row ID (primary key, object ID, etc.) is the most effective way to query arbitrary data source. This is due to the native nature of query. The model utilizing native query can use all of the optimization techniques that the data source provide. This is done transparently as the native optimization methods (i.e. indexing) are managed by the database engine that processes each native query. The general architecture actually has no knowledge of the actual optimization happening at the data source side. The native query is simply send to the datasource and the result is the requested data. Such an approach results in up-to-date and reliable data on demand. This is particularly important in online applications that require current data state. The metamodels advantage over the data model for i.e. ETL, is that it is only metadata about the data. No data dependent constraints are necessary. In fact such issues are going to be considered and handled at the target data source with its best optimization of native query engine and without any effort at global integration scale.

4.2 Solution Proposal to OLAP issues

As the OLAP systems are based on ETL populating the OLAP with source data, the downsides of ETL discussed above have become also part of OLAP systems. Due to this inherent drawback from ETL on ground of current data acquisition, the proposed solution is solving the issue by delivering up-to-date content that is pulled on demand.

4.3 No distributed transactions

The discussed architecture, due to metadata utilization, can also store simple updates apart from selecting queries. However, this requires further elaboration. In case of proposed data integration solution, the data stored at data sources can be updated with extra records. This brings us to a point when a data replication in a distributed grid environment must be handled. Let us assume then, that the system is going to add one record that needs to be replicated in three integrated databases controlling the same schema. The integrator sends native updating request to all three sources. Now, the problem is when one of the three sources is not able to update its state and thus cause potential inconsistency in integrated grid. In regular distributed system, in such condition, one would have to apply one of the following solutions. First approach is a distributed two phase commit which means all actions being a part of the same transaction. This implies tying up all of the systems in voting process until the unsuccessful update on failed source is going to be handled or until remaining two sources will rollback the distributed update. Both cases bring major inefficiency forcing entire system to halt until the problem is resolved. Another, more advanced solution, would be dividing the distributed transaction into distinct transactions but in an asynchronous queue. This way you can send the database update message by enqueuing the update message in one transaction and forget about it. This is actually not really distributed due to the fact that close location of the queue broker and the message sender is required. This means that such solution requires the queue broker and the database to be collocated, in the same datacenter with fast connectivity and high availability. The queuing system takes care of everything by utilizing reliable messaging. The target systems then processes update message locally in transaction from the queue. So each dequeue of message and update to a data source is a separate transaction. Now, in case when one of the data sources will not make an update from the queue, there is no way of instantaneous rollback on remaining sources, that has already accomplished this task with success. A compensation process is required in such case. This is done by sending message from the system that failed to process the transaction to the remaining systems, either to rollback or to handle this situation in other arbitrary way. The big advantage of this solutions is that transactions happen quickly, asynchronously and do not tie up entire system over one transaction. On the other hand it is not really distributed due to collocation requirement and while intense system load this can bring major system slow down due to need of compensation even with fast connectivity.

In our solution a transaction is not required. In the exemplary case, when two out of three data sources accept the update and the third one does not, we simply change the metamodel of integrator to store access methods pointing only to the first two parties where the update has succeeded. Now, in case of asking integrator for all updated records, only the data sources that has made updates with success (i.e. data source one and two) will be made available to pull data. The third data source will be considered only while querying for data that it has managed to store until this update message. This can be done due to metamodel presented in [**20**].

4.4 Architecture open to optimization

The key aspect of integration relies on networking. High availability is clue to integrate resources efficiently. Considering intensive networking traffic the optimization of retrieving replicated data is crucial to effective and responsive communication. Therefore, we have prepared architecture for replaceable load balancing algorithms [**22**] covering access to replicated data. The architecture at the level of integrator is also prepared for applying optimizations that has already been used for ORM solutions like Hibernate [**23, 24, 25, 26**]. The results of the mentioned optimization are published in separate paper. In [**27**] the authors has proven that exemplary optimization technique basing on query rewriting and order dependencies can be applied to arbitrary data source without need to interfere with the database engine and thus regardless of its origin. This is due to effective design of the proposed architecture and its flexibility and agility of appliance.

5 Conclusions and Future Work

Existing solutions that has been widely used tend to solve specific problems not even trying to generalize due to their constrained applications. In this paper we have presented the ways that we believe can challenge the problems that the existing solutions do not aim to solve. On ground of an abstract model, that is out of scope for this paper, our solutions can be combined into fully abstract and general architecture for heterogeneous data integration. In future papers we will introduce the entire architecture that has been devised to challenge the general problems of data integration in heterogeneous and distributed environment.

References

[1] http://www.omg.org/technology/documents/corba_spec_catalog.htm, 2010.

[2] http://en.wikipedia.org/wiki/list_of_web_service_specifications, 2010.

[3] http://www.jini.org/wiki/jini_architecture_specification, 2009.

[4] K. Kuliberda, R. Adamus, J. Wislicki, K. Kaczmarski, T. M. Kowalski, and K. Subieta. A generic proposal for a transparent integration of distributed data by an autonomous layer in a virtual repository. *an International Journal (MAGS)*, 3(4):393–410, 2007.

[5] M. W. Sobolewski. Computing and metacomputing intergrid. *Proc. 10th International Conference on Enterprise Information Systems, Barcelona, Spain*, 2008.

[6] M. W. Sobolewski. Federated collaborations with exertions. pages 127–132, 2008.

[7] R. Grycuk et al. Content-based image indexing by data clustering and inverse document frequency. *Communications in Computer and Information Science*, pages 374–383, 2014.

[8] M. Lupa and A Piórkowski. Spatial query optimization based on transformation of constraints man-machine interactions. 242, 2014.

[9] P. Leszczynski and K. Stencel. Update propagator for joint scalable storage. *Fundam. Inform.*, 119(3-4), 2012.

[10] P. Hepner. Integrating heterogeneous databases: An overview, school of computing and mathematics. *Deakin University, Geelong, Victoria, Australia*, 1995.

[11] V. D. Gilgor and G. L. Luckenbaugh. Interconnecting heterogeneous database management system. *IEEE Comp. Soc. Press*, 17(1), 1983.

[12] S. E. Madnick. A taxonomy for classifying commercial approaches to information integration in heterogeneous environments. *Database Engineering - Special Issues on Database Connectivity*, 13(2), 1999.

[13] V. Rajeswari et al. Heterogeneous database integration for web applications. *International Journal on Computer Science and Engineering*, 1(13):227–234, 2009.

[14] S. C. Tseng Frank. Heterogeneous database integration using xml.

[15] S. Wei-Jung and H. Minng-Ying. An interactive tool based on xml technology for data exchange between heterogeneous erp systems. *Journal of CIIE*, 22(4):273–278, 2005.

[16] P. Rodriguez-Gianolli and J. Mylopoulos. A semantic approach to xml-based data integration conceptual modeling.

[17] Microsoft TechNet. http://technet.microsoft.com/en-us/library/ms151835.aspx.

[18] J. Basu Nirav Chanchani. Heterogeneous xml-based data integration.

[19] B. Inmon. Building the data warehouse.

[20] M. Chromiak and K. Stencel. A data model for heterogeneous data integration architecture. 424, 2014.

[21] M. Chromiak and K. Stencel. The linkup data structure for heterogeneous data integration platform. 7709:263–274, 2012.

[22] et al. A. Piorkowski. Load balancing for heterogeneous web servers. 79:189–198, 2010.

[23] P. Wisniewski and K. Stencel. Query rewriting based on metagranular aggregation. pages 457–468, 2013.

[24] M. Gawarkiewicz and P. Wisniewski. Partial aggregation using hibernate. pages 90–99, 2011.

[25] A. Boniewicz et al. On materializing paths for faster recursive querying. pages 105–112, 2013.

[26] M. Burzanska et al. Recursive queries using object relational mapping.

[27] M.Chromiak et al. Exploiting order dependencies on primary keys for optimization. 2014.