Scientific Journal of Riga Technical University
Computer Science. Information Technology and Management Science
2011
Volume 49

An Approach to Parallelization of Remote Data Integration Tasks

Janis Kampars[1], Janis Grabis[2], [1-2]*Riga Technical University*

***Abstract* – Data integration from autonomous, remote data sources is complicated by the data source heterogeneity, lack of methodological support and appropriate data integration systems. To solve this problem, the On-demand Remote Data Integration Architecture (ORDIA) is defined, which promotes maintenance and allows minimizing data integration time. A data integration task parallelization algorithm is the key part of this architecture. A detailed description of this algorithm is provided, and its performance is evaluated by experimental comparison with other data integration solutions.**

***Keywords* – data as a service, data integration, web services**

I. INTRODUCTION

Companies make various decisions on a regular basis. The accuracy of the decision depends upon adequacy of the available data, number of alternatives and decision-making model [1]. Data combination from different sources and acquisition of a single comprehensible view on all of the combined data is called data integration [2]. Data integration for analytical purposes is called business intelligence data integration [3].

The necessary data can be located both inside and outside the organization. Traditionally external data are first copied into a local centralized storage (e.g. data warehouse), however such solutions are inefficient and hard to implement [4]. The necessary infrastructure has to be provided, and the data must be updated regularly. The evolution of web service related technologies has provided an alternative approach – Data as a Service (DaaS) [5] – [7]. In DaaS, data sources are heterogeneous, remote web services. DaaS allows one to retrieve only the necessary data when it is required by the decision-making process, so data retrieval and combination from DaaS based data sources is referred to as on-demand data integration.

Data integration from remote, autonomous sources is complicated by heterogeneous nature of data sources, lack of suitable integration solutions and methodological support. The time needed for data integration must be minimized as it constitutes a substantial part of the overall decision-making process. For simple decision making problems data integration time exceeds decision-modelling time [8]. Individual data integration task parallelism can be achieved by multithreading and parallel programming, however interdependencies among data integration tasks must be considered.

The objective of the paper is to propose an approach to parallelization of remote data integration tasks, as well as experimentally evaluate it. The rest of the paper is organized as follows. Section 2 reviews related work and identifies main challenges which complicate data integration from heterogeneous sources. The architecture is defined in Section 3; the developed data integration task parallelization algorithm is summarized in Section 4. The results of the experimental evaluation of the data integration task parallelization algorithm are provided in Section 5. Section 6 concludes the paper.

II. STATE OF THE ART

Traditional data integration solutions like data warehouses or virtual databases are not suitable for dynamically changing data source interfaces and user requirements [19]. Constantly changing environment makes maintenance of such systems complex and expensive [20]. Data integration from DaaS-based sources can be performed using ESB (Enterprise Service Bus) [21], [22] however ESB is mostly used for enterprise application integration and transactional data integration. ETL (Extract, Transform, Load) systems are not suitable for data retrieval from remote web services, because they support XML and related specifications [9], [10] only partly; nevertheless, XML is widely used in web services.

There are several researches that focus on data integration from remote web services. Jun et al [4] have proposed a dynamic data integration model based on SOA. It consists of four layers – uniform data accessor, XML view and SOA mapper, processing engine of XML data integration view and XML data integration view. Masseroli et al [23] integrate bioinformatics web services by using Search Computing Based Data Integration Architecture. Abiteboul et al [24] define Active XML (AXML) peer-to-peer integration architecture. Its central elements are AXML documents containing web service invocations, SOAP web services and AXML peers, providing AXML document processing services. Frehner et al [25] use a virtual database to integrate distributed spatial data which is accessed via web services based on WFS (Web Feature Service) specification. Virtualization approach is also used in [26]. Data sources are integrated by creating Unique Virtual View consisting of data ontology, global light service ontology and a set of mappings connecting elements of the two ontologies. Fujun et al [19] have defined a Service Oriented Data Integration Architecture, which performs on demand data integration from heterogeneous, autonomous data sources. The web services integrated must support SOAP protocol, and their metadata must be expressed in WSDL or DAML-S language. There are researches that concentrate on using XML for data integration purposes [10], [27], however XML retrieval via web services is not considered.

Some of the mentioned researches are based on the assumption that all web services will be SOAP based, there will be sufficient metadata available and there will be an option to modify web services to fit the data integration system. Unfortunately, in practice these assumptions often do not hold.

There are many factors complicating data integration from remote, heterogeneous data sources. There is a large variety of protocols and standards used in web services [11] – [13]. Web service interfaces are dynamically changing [7]. The use of the Internet as the medium for data transfer in conjunction with web service interface variability leads to a high possibility of errors during the data integration process. In cases when there are a large number of functionally equivalent web services, non-functional requirements like average response time and percentage of dropped requests can become the most important factors. That is why the quality of service and data must be considered, however the available information can be incomplete and devious [14]. Although the possible non-functional quality attributes are defined [15], [16], automatic data source (web service) discovery based on non-functional requirements is complicated and not supported by the leading web service register technologies [16] – [18]. There is a lack of well-defined documents about web service and their corresponding data usage permission, intellectual property rights and legal issues [6].

III. ARCHITECTURE

To provide effective remote data integration, the On-demand Remote Data Integration Architecture (ORDIA) is proposed. This architecture enables execution of multi-step remote data integration processes according to decision-making requirements. The architecture is based on the abstraction approach – abstract data retrieval operations and their corresponding input, output and error message data models are defined. Web service methods are mapped to abstract data retrieval operations by using abstraction layers.

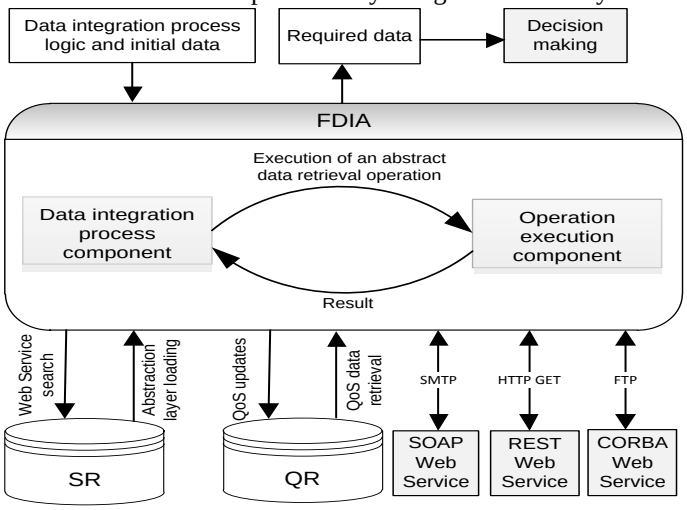

Fig. 1. On-demand remote source data integration architecture

The data integration process is defined using blocks and their configuration parameters. It is formed by interexchange of data integration tasks among blocks. A block is the basic functional unit of the data integration process; it provides transformations, loops, conditional expression evaluation and execution of abstract data retrieval operations. The data integration process logic is completely separated from the web service access logic which is encapsulated in the web service abstraction layers. In case of web service interface variance, the corresponding web service abstraction layer must be altered. New alternative web services are added by creating their abstraction layers and mapping them to specific abstract data retrieval operations. In both cases changes in data integration process logic are not required, thus promoting maintenance.

The architecture (Fig. 1) consists of Service Register (SR), Quality Data Repository (QR) and Functional Data Integration Adapter (FDIA). SR stores information about abstract data retrieval operations, their data models and abstraction layers, which provide web service mapping to abstract data retrieval operations. The data models are defined using XML schema.

The quality of web services and their data is stored in QR. Quality measures stored in QR are monitored and updated during the data integration process. QR allows effective load balancing and web service selection based on non-functional requirements. The main part of the architecture is FDIA, which receives XML based initial data, data integration process logic as an input and returns the required data. The data integration process logic can represent complex individual data integration task interdependencies, and it contains:

Minimal allowable quality measures for the used web services.

Weights of quality measures.

A set of data integration tasks forming the data integration process.

Task interdependencies and correct execution order.

Maximum number of web services used for load balancing within a single abstract data retrieval operation.

At the beginning of the data integration process, the initial data are copied into the temporary data document. The temporary document contains input data for abstract data retrieval operations and is iteratively updated until it contains all of the necessary data. FDIA is divided into Data Integration Process (DIPC) and Operation Execution (OEC) components.

DIPC is responsible for monitoring of data integration process execution, input data transformation according to specific abstract operation input data models, realization of loops, evaluation of conditional expressions and data aggregation. To solve data integration task interdependencies, a block mediator (BM) class is defined. Data integration tasks created by blocks are sent to BM, which verifies if the task can be further sent to its recipients for execution. The block receives a data integration task only when there are no blocking task dependencies. The possible data integration task states are defined as idle, created, sent, processing, finished and error.

The architecture contains three different block types:

Simple Block (SB) – used to create loops and evaluate conditional expressions.

Transformation Block (TB) – document transformation services.

Operation Block (OB) – abstract data retrieval operation execution.

There are two possible relations between blocks:

Succession – block A is a successor of block B if it receives data integration tasks formed in block B.

Dependency – block A depends on block B if it can process incoming data integration task only if block B has finished processing the related data integration tasks.

When an abstract data retrieval operation is executed in OB, OEC is called asynchronously to search for suitable web services in SR and to execute a specific web service method through the abstraction layer. This process also contains error recovery, load balancing and update of web service related quality data.

IV. Data Integration Task Parallelization Approach

Data integration task parallelization can be acquired using parallel programming and multithreading, however in order to provide correct and timely execution of data integration tasks, their relations must be considered. Some data integration tasks depend upon the result of other data integration tasks and cannot be executed prior to receiving the required information. Data integration task progress must be monitored, and block relations have to be taken into account. The possible data integration task states are as follows:

Idle – BM is unable to send data integration task to its recipient due to a blocking dependency. Some of the data required for data integration task execution is still unavailable.

Created – BM has received data integration task generated by a parent block and has created the corresponding data integration tasks for its successors.

Sent – data integration task is sent to the recipient.

Processing – block has read the data integration task and is processing it.

Finished – block has finished processing the incoming data integration task. The output data integration task is created and sent to the BM.

Error – there has been an error in the process of data integration task execution. The data required for data integration task execution cannot be retrieved.

Data integration task parallelization is illustrated in Fig. 2. Block B and C are the first level successors of block A. Block E depends upon block D. Block A receives an incoming data integration task with XPath address "/S", adds its relative XPath address to the XPath address of the incoming data integration task and searches for corresponding elements in the temporary data XML document. The element processing logic is defined by the block class type (SB, TB, OB) and its configuration parameters. In the example shown in Fig. 2, two elements are found and two data integration tasks are sent to the successors of the block A via BM class.

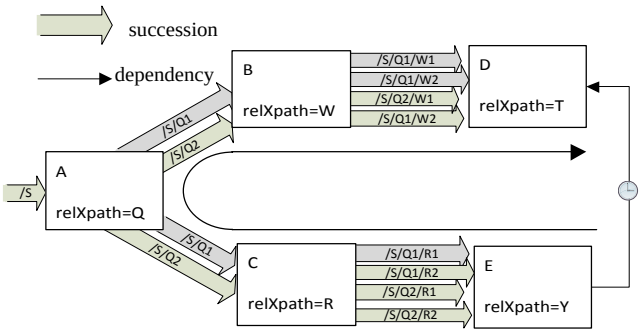

Fig. 2. Data integration task interdependency detection

The incoming data integration tasks are processed, and output data integration tasks are generated in blocks B and C by analogy. In order to send data integration task to block E, data integration task interdependencies must be considered. The data integration task interdependency validation algorithm implemented in BM is defined as follows:

A data integration task is generated in the predecessor of block E (corresponding XPath address - "`/S/Q1/R1`") and sent to BM. This task is further referred to as dependent task. BM has to verify if the dependent task can be sent to the recipient.

The common predecessor of block D and E is found. It is a block, whose successors (not necessarily the first level) are both of the dependent blocks D and E. There must be no other successor of the common predecessor, whose successors are block D and E. The common predecessor of block D and E is block A.

While navigating from block E to the common predecessor of both blocks (block A), the data integration task which initiated the creation of the dependent task is identified. It is a data integration task with a matching XPath address beginning. In this case the dependent task with XPath address "`/S/Q1/R1`" was initiated by the data integration task with XPath address "`/S/Q1`" sent by block A.

While navigating from the common predecessor of block E and D (block A) to block D, data integration tasks initiated by the data integration task identified in Step 3 are found. These are data integration tasks whose XPath address beginning matches XPath address found in Step 3 ("`/S/Q1`", "`/S/Q1/W1`", "`S/Q1/W2`"), and they are further referred to as blocking tasks. The states of the blocking tasks are verified:

If all of the blocking tasks have the state "Finished", the dependent data integration task can be sent to its recipient. The data integration task interdependency check algorithm exits.

If at least one of the blocking tasks has the state "Error", the state of the dependent task is changed to "Error", and it cannot be sent to the recipient. A search is performed to find other data integration tasks depending on the dependent task. If any of them are found, their status is changed to "Error" as well, and the search is continued to check if there are more data integration tasks influenced by the error. The data integration task interdependency check algorithm exits.

If none of the blocking tasks have the status "Error", but not all of the tasks have the status "Finished", the state of the dependent task is changed to "Idle". It cannot be sent to the recipient due to the blocking dependency.

The blocking tasks are monitored (it is also checked for any new blocking tasks created). If the state has changed for any of those tasks, Step 4 is repeated.

V. EVALUATION

Experiments have been performed on ORDIA prototype, sequential data integration solution (implemented in C#) and commercial ETL system (Microsoft SQL Server 2008 Integration Services) to evaluate the effect of data integration task parallelization on the total data integration time T_K. Calculation of the mathematical expression (1) is chosen as an example, web services performing mathematical operations are used to simulate remote data sources.

$$Y = \frac{\sqrt{x_1^2+x_2^3+x_3^4}-(x_4*x_5*x_6)^2}{x_7+x_8+x_9} \quad (1)$$

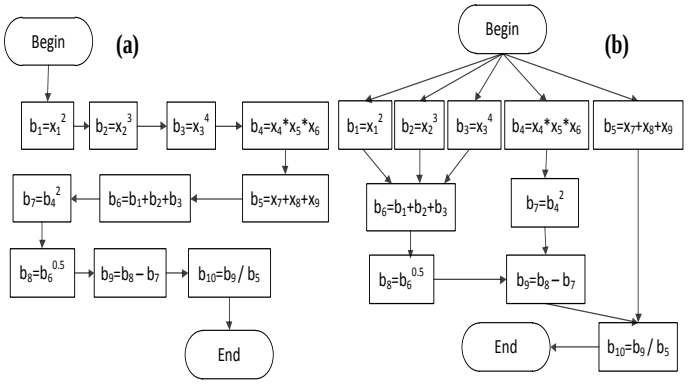

Fig. 3. Mathematical operation performance order: (a) sequential, (b) parallel

Calculation of the expression is repeated several times $N_A \in \{1, 3, 5\}$, and the variability of the T_K is monitored. The maximum number of simultaneous requests performed on a data source L_V varies during experiments ($L_V=\infty$ and $L_V=1$) in the ORDIA prototype and ETL. Experiments are repeated 10 times to reduce experimental error. The order of actions needed to solve (1) in the case of parallel and sequential data retrieval scenario is shown in Fig. 3. The web services named R1, R2, R3, R4 and R5 are implemented in PHP using the REST architectural style. Web service R1 is used for aggregation, R2 performs subtraction, R3 allows multiplication, R4 performs division, R5 allows getting power or root of a number. Web service request and response times are registered to allow evaluating the web service load and T_K in each of the scenarios.

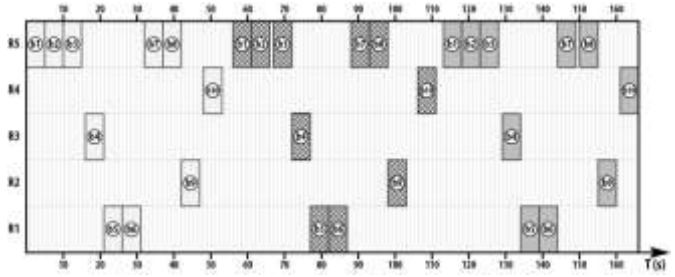

Fig. 4. Web service load and T_K in case of the sequential data integration task execution, $N_A=3$

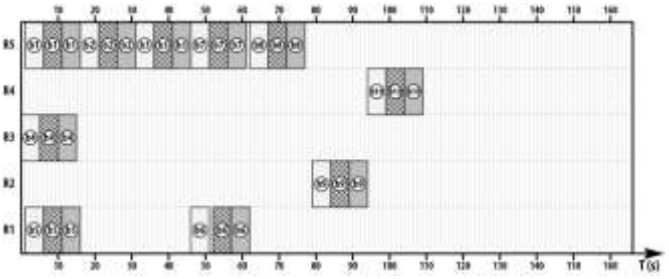

Fig. 5. Web service load and T_K in case of the parallel data integration task execution using ETL, $N_A=3$, $L_V=1$

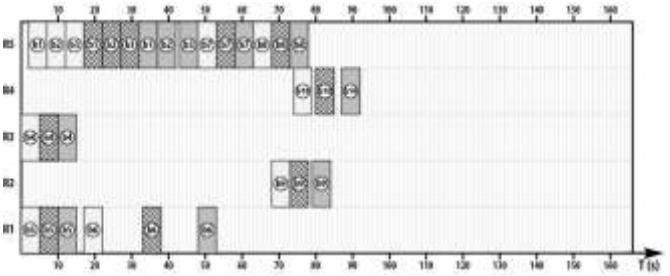

Fig. 6. Web service load and T_K in case of the parallel data integration task execution using ORDIA prototype, $N_A=3$, $L_V=1$

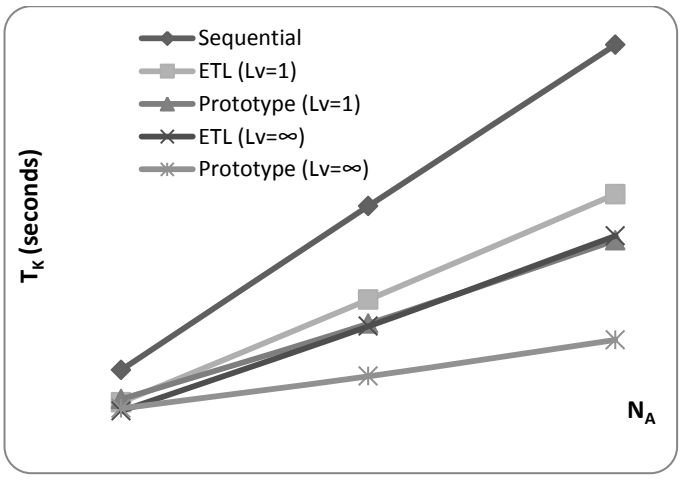

Fig. 7. Effect of T_K on N_A in each of the scenarios

The observed web service load in case of the sequential data integration task execution is shown in Fig. 4. Fig. 5 illustrates web service load when using ETL. Fig. 6 visualizes parallel data integration task execution in the ORDIA prototype. The small blocks shown in Fig. 4, Fig. 5 and Fig. 6 represent web service request and processing time; they are coded according to mathematical operation designation in Fig. 3. The effect of N_A on T_K for each of the scenarios is visualized in Fig. 7.

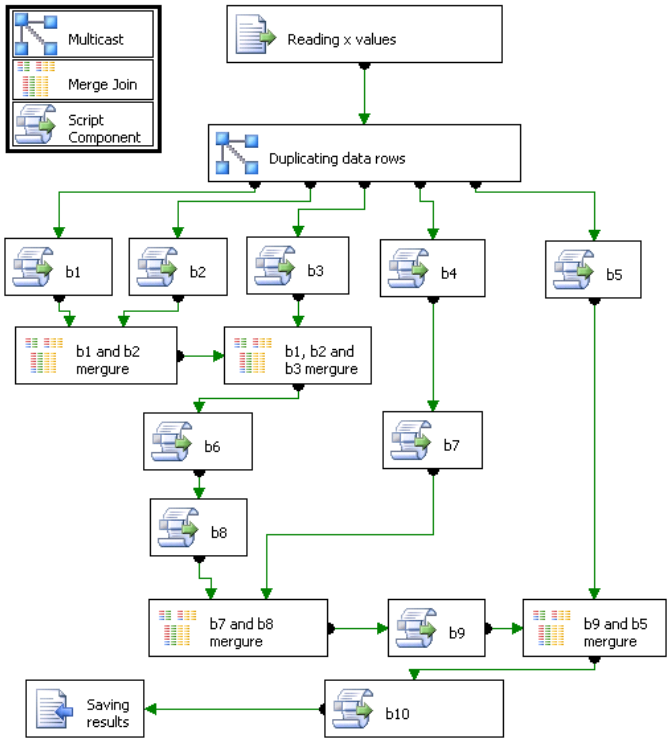

Fig. 8. Data integration process visualization in ETL, $L_V=\infty$

In case of the unlimited concurrent web service requests, the data integration process implemented in ETL is illustrated in Fig. 8. Script Components performing web service invocations are coded according to Fig. 3.

In case of the ORDIA prototype, L_V can be easily modified and adoption to potential changes in web service interfaces is implemented in web service abstraction layers without affecting the data integration process logic, thus promoting maintenance.

The most inefficient is sequential data integration. Comparison of parallel data integration processes implemented in ETL and ORDIA prototype shows the advantage of the developed data integration task parallelization algorithm. It becomes more obvious when the amount of the data to be retrieved increases (larger N_A values). Unlimited number of simultaneous requests to a single web service allows considerable reduction of T_A when using the ORDIA prototype. However the difference is less notable when ETL is used.

While experimenting with ETL, several problems have been observed:

There are no ready-made blocks suitable for data retrieval from web services, so the Script Component block has been adapted.

The data integration process logic includes web service access logic (implemented in Script Component blocks). In case of the web service interface modification, the data integration process logic has to be altered.

The data integration process is bind to physical web service implementation. In case of an error, it is hard to implement load balancing and web service substitution, and it would complicate the data integration process logic even more.

There is no convenient way to set the maximum number of concurrent requests performed for a single web service. Simultaneous requests can only be limited by performing manipulations with ETL blocks (in this case Merge Join blocks have been used). To change L_V the data integration process logic must be altered. Only two possible L_V values can be ensured – $L_V=1$ and $L_V=\infty$.

As it has already been identified in literature [9], [10], ETL is not suitable for XML data processing. During experiments x_i values have been retrieved from CSV file and results have also been saved in the CSV format.

VI. Conclusions

Although the on-demand data integration from remote heterogeneous sources is a promising approach which provides no need for data storage infrastructure and regular data updates, it is more complex to implement than the data retrieval from well-known, local data sources. It is complicated by web service discovery, interface variability, use of semi-structured, hierarchical data format, high error risk, ambiguous terms of use and necessity for load balancing. There is a lack of suitable software and methodological support.

This paper proposes the ORDIA system, which is based on XML and related specifications and allows implementing the on-demand data integration from remote, heterogeneous web services. The components of the architecture and the data integration task parallelization algorithm are defined.

To approve the significance of the data integration task parallelization and the effectiveness of the developed algorithm, tests have been performed using the sequential data integration solution, ORDIA prototype and commercial ETL system (Microsoft SQL Server 2008 Integration Services). It has been acknowledged that data integration task parallelisation allows significant reduction of the total data integration time T_K. It has also been confirmed that the data integration task parallelisation algorithm implemented in the ORDIA prototype is more effective than the one implemented in ETL when retrieving data from remote web services.

REFERENCES

[1] V. L. Sauter, *Decision Support Systems for Business Intelligence*. USA: Wiley, 2011, pp. 3-4.

[2] M. Casters, *et al.*, *Pentaho Kettle Solutions: Building Open Source ETL Solutions with Pentaho Data Integration* Canada: Wiley, 2010, pp. 18.

[3] A. D. Giordano, *Data Integration Blueprint and Modeling: Techniques for a Scalable and Sustainable Architecture*. USA: IBM Press, 2011, pp. 7-8.

[4] J. Wang, *et al.*, "A dynamic data integration model based on SOA," presented at the Second ISECS International Colloquium on Computing, Communication, Control, and Management (CCCM), Sanya, China, 2009

[5] A. Dan, *et al.*, "Information as a Service: Modeling and Realization," presented at the International Workshop on Systems Development in SOA Environments (SDSOA), Minneapolis, Minnesota, USA, 2007.

[6] H. L. Truong and S. Dustdar, "On analyzing and specifying concerns for data as a service," presented at the 2009 IEEE Asia-Pacific Services Computing Conference (APSCC), Biopolis, Singapore, 2009.

[7] H. L. Truong and S. Dustdar, "On evaluating and publishing data concerns for data as a service," presented at the IEEE Asia-Pacific Services Computing Conference, Hangzhou, China, 2010.

[8] J. Kampars and J. Grabis, "Spatial Data Integration Approach with Application in Facility Location," presented at the 16th International Conference on Information and Software Technologies, Kaunas, Lithuania, 2010.

[9] Y. Guohua and W. Jingting, "The Design and Implementation of XML Semi-structured Data Extraction and Loading into the Data Warehouse," presented at the International Forum on Information Technology and Applications (IFITA), Guangzhou, China, 2010.

[10] M. Bhide, *et al.*, "XPEDIA: XML processing for data integration," *Proceedings of the VLDB Endowment*, vol. 2, pp. 1330-1341, 2009.

[11] J. Kopecký, *et al.*, "hRESTS: An HTML Microformat for Describing RESTful Web Services," presented at the IEEE/WIC/ACM International Conference, Sydney, Australia, 2008.

[12] C. Pautasso, *et al.*, "RESTful web services vs. "Big" web services: Making the right architectural decision," presented at the 17th International Conference on World Wide Web 2008, Beijing, China, 2008.

[13] D. Jagannadham, *et al.*, "Java2 distributed application development (Socket, RMI, Servlet, CORBA) approaches, XML-RPC and web services functional analysis and performance comparison," presented at the International Symposium on Communications and Information Technologies (ISCIT), Sydney, Australia, 2007.

[14] D. Grosu and A. T. Chronopoulos, "A Truthful Mechanism for Fair Load Balancing in Distributed Systems," presented at the Network Computing and Applications (NCA), Cambridge, MA, USA, 2003.

[15] W3C. *QoS for Web Services: Requirements and Possible Approaches*. Available: http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/ [Accessed: Sept. 11, 2011]

[16] S. Ran, "A model for web services discovery with QoS," *ACM SIGecom Exchanges*, vol. 4, pp. 1-10, 2003.

[17] E. M. Maximilien and M. P. Singh, "A framework and ontology for dynamic web, services selection," *IEEE Internet Computing*, vol. 8, pp. 84-93, 2004.

[18] Y.-j. Mou, *et al.*, "Interactive Web service choice-making based on extended QoS model," presented at the The Fifth International Conference on Computer and Information Technology (CIT), Shanghai, China, 2005.

[19] Z. Fujun, "Dynamic Data Integration Using Web Services," presented at the IEEE International Conference on Web Services, San Diego, California, USA, 2004.

[20] J. Lacasta, *et al.*, "A Web Ontology Service to facilitate interoperability within a Spatial Data Infrastructure: Applicability to discovery," *Data and Knowledge Engineering*, vol. 63, pp. 947-971, 2007.

[21] R. Abrahiem, "A new generation of middleware solutions for a near-real-time data warehousing architecture," presented at the IEEE International Conference on Electro/Information Technology (EIT), Chicago, IL, USA, 2007.

[22] H.-p. Si, *et al.*, "Efficient implementation of data integration and sharing of crop germplasm resources investigation," presented at the International Conference on Computer Application and System Modeling (ICCASM), Taiyuan, China, 2010.

[23] M. Masseroli, *et al.*, "Bio Search Computing: Bioinformatics web service integration for data-driven answering of complex Life Science questions," *Procedia Computer Science*, vol. 4, pp. 1082-1091, 2011.

[24] S. Abiteboul, *et al.*, "Active XML: Peer-to-Peer Data and Web Services Integration," presented at the 28th International Conference on Very Large Databases (VLDB), San Francisco, USA, 2002.

[25] M. Frehner and M. Brändli, "Virtual database: Spatial analysis in a Web-based data management system for distributed ecological data," *Environmental Modelling & Software*, vol. 21, pp. 1544-1554, 2006.

[26] M. Palmonari, *et al.*, "Aggregated search of data and services," *Information Systems*, vol. 36, pp. 134-150, 2011.

[27] F. Xiong, *et al.*, "Research and implementation of heterogeneous data integration based on XML," presented at the 9th International Conference on Electronic Measurement and Instruments (ICEMI), Beijing, China, 2009

Janis Kampars has received his B. Sc. Ing. (in 2004) and Mg. Sc. Ing. (in 2006) degrees in Information Technology from Riga Technical University, Latvia. He is currently working on his Doctoral Thesis under supervision of Dr. Janis Grabis in the field of on-demand data integration from remote, heterogeneous sources. The defence of the Thesis is planned to be held at the end of the year 2011. Since 2004 he has been working at the Faculty of Computer Science and Information Technology, Riga Technical University. The current positions are lecturer and researcher. His areas of interest are application integration, data integration and business intelligence.

Janis Grabis is a Professor at the Faculty of Computer Science and Information Technology, Riga Technical University, Latvia. He obtained his Doctoral degree from Riga Technical University in 2001 and worked as a Research Associate at the College of Engineering and Computer Science, University of Michigan-Dearborn. He has published articles in major academic journals including OMEGA, European Journal of Operational Management, International Journal of Production Research, Human Systems Management and others. He has been a guest-editor for two top academic journals and a member of the program committee of several academic conferences. His research interests are enterprise applications, project management and system optimization.

Jānis Kampars, Jānis Grabis. Attālu avotu datu integrācijas uzdevumu paralelizācijas pieeja

Uzņēmumi ikdienā pieņem dažādus lēmumus. Nepieciešamie dati var atrasties ārpus uzņēmuma. Datu savākšana un pārveidošana analīzei piemērotā formā tiek saukta par biznesa intelekta datu integrāciju. Lai analizētu ārējos datus, tradicionāli tiek izveidota to pilna lokāla kopija, tomēr tam ir nepieciešama atbilstoša infrastruktūra un regulāra datu atjaunošana. Eksistē alternatīva pieeja – dati kā pakalpojums (Data as a Service), kurā datu avoti ir attālas, heterogēnas tīmekļa pakalpes. Līdz ar to ir iespējams iegūt tikai nepieciešamos datus tad, kad tas ir nepieciešams (pieprasījuma datu integrācija). Atkrīt jautājumi, kas saistīti ar infrastruktūras izveidi liela datu apjoma glabāšanai un regulāru datu atjaunošanu. Tradicionālie datu un lietojumprogrammatūras integrācijas rīki nav piemēroti datu integrācijai no ārējām, heterogēnām tīmekļa pakalpēm. Integrācijas procesu sarežģī vairāki faktori, piemēram, datu avotos izmantoto protokolu un standartu

dažādība, daļēji strukturēta formāta izmantošana un datu avotu interfeisu mainība. Šajā pētījumā tiek definēta attālu avotu pieprasījuma datu integrācijas sistēmas arhitektūra, kas balstās uz abstrakcijas pieeju un ļauj pilnībā nodalīt datu integrācijas procesu no tīmekļa pakalpju piekļuves loģikas. Īpaša vērība tiek pievērsta kopējā datu integrācijas laika minimizēšanai un datu integrācijas uzdevumu paralelizācijai. Tiek definēts algoritms, kas nodrošina pareizu un savlaicīgu datu integrācijas uzdevumu izpildi. Lai novērtētu algoritma efektivitāti, sistēmas prototips tiek praktiski salīdzināts ar komerciālu ETL sistēmu (Microsoft SQL Server 2008 Integration Services) un secīgu datu integrācijas risinājumu. Iegūtie rezultāti apstiprina datu integrācijas uzdevumu paralelizācijas nozīmi, kā arī to, ka arhitektūrā īstenotais algoritms ļauj nodrošināt mazāku datu integrācijas laiku nekā ETL sistēma. Starpība starp datu integrācijas laiku ETL sistēmā un arhitektūras prototipā pieaug, palielinot izgūstamo datu apjomu un noņemot limitu vienlaicīgi izpildāmo pieprasījumu skaitam. Ir apstiprināts pieņēmums, ka ETL sistēmas nav piemērotas datu integrācijai no attālām tīmekļa pakalpēm.

Янис Кампарс, Янис Грабис. Метод параллелизации задач интеграции данных из удалённых источников

В своей повседневной деятельности компании регулярно принимают различные решения. Необходимые данные могут находиться за пределами компании. Извлечение данных и их трансформация в целях анализа называется интеграцией данных бизнес-аналитики. Для анализа внешних данных традиционно делают полную локальную копию, для чего требуются соответствующая инфраструктура и регулярные обновления данных. Существует альтернативный подход - данные как сервис (Data as a Service), в котором источниками данных являются удалённые гетерогенные веб-сервисы. Таким образом, можно получить только необходимые данные тогда, когда это необходимо. В этом случае проблемы, которые связаны с инфраструктурой для хранения больших объемов данных и регулярного обновления данных, отсутствуют. Традиционные решения для интеграции приложений и данных не подходят для интеграции данных с удалённых гетерогенных веб-сервисов. Интеграцию осложняют такие факторы, как использование различных стандартов и протоколов в источниках данных, полуструктурированный формат и изменчивость интерфейса. В этой работе определена архитектура для интеграции данных с удалённых гетерогенных веб-сервисов. Архитектура основана на подходе абстракции и позволяет отделить логику доступа веб-сервисов от логики процесса интеграции данных. Особое внимание уделено минимизации общего времени интеграции данных и параллелизации задач интеграции данных. Разработан алгоритм, который обеспечивает точное и своевременное выполнение задач интеграции данных. Для оценки эффективности алгоритма прототип системы практически сравнен с коммерческой ETL системой (Microsoft SQL Server 2008 Integration Services) и последовательным решением интеграции данных. Полученные результаты подтверждают значение параллелизации задач интеграции данных. Алгоритм, реализованный в прототипе архитектуры, обеспечивает более быструю интеграцию данных, чем ETL система. Разница во времени интеграции в ETL и в прототипе растет с увеличением объема извлекаемых данных и с удалением предела одновременных запросов веб-сервисам. В результате работы установлено, что ETL-системы не подходят для интеграции данных с удалённых веб-сервисов.