An architecture for efficient, flexible enterprise system integration

John Grundy[1, 2], Jun Bai[1], John Blackham[3], John Hosking[1] and Robert Amor[1]

[1]Department of Computer Science
[2]Department of Electrical and Electronic Engineering
University of Auckland
Private Bag 92019, Auckland, New Zealand
{john-g, john, trebor}@cs.auckland.ac.nz

[3]XSol Ltd, Level 2, 149 Parnell Road
Parnell, Auckland, New Zealand
Phone: +64-9-302-9684
ajb@xsol.com

Abstract

Integrating complex enterprise systems is challenging and reliability and performance of the integrated systems can be problematic when using typical solutions of distributed transactions or on-demand message-based querying. We describe a data-oriented approach for enterprise system integration that uses information brokering. A broker application isolates the interactions between a local enterprise application server and a wide variety of remote systems, performing data acquisition, storage, transformation, update and status management. We describe a prototype book brokering system developed using Java 2 Enterprise Edition, CORBA, Java Messaging Service and Web Services and using our integration strategy. We outline the architecture, design and implementation of this prototype and summarise our experiences with this information integration technique.

1. Introduction

Enterprise systems integration has become important in many domains, both intra-organisational, focusing on supporting Enterprise Applications Integration [15, 21], and inter-organisational, focusing on supporting Business-to-Business E-commerce [1, 18]. Systems need to be integrated in order for users to have easier access to data and information processing facilities, for data to be more easily kept consistent, for multiple organizations to more efficiently interoperate, and for new and legacy applications to co-exist and complement one another [18, 22]. However integrating enterprise systems is challenging, due to varying degrees of support for integration, a huge range of integration technologies, the complexity of translating between different system data structures and business processes, and ensuring integrated system reliability and performance is acceptable.

There have been many approaches developed to integrating enterprise systems. These range from file and document exchange [5], database integration and federation [3, 6, 14, 20], message and document translation [19, 15, 9, 10, 17], distributed object interaction [22, 2, 6], to workflow and business process management [8, 21]. Most data-oriented approaches limit the range of technologies that can be integrated and only work on low-level database interaction. Message-oriented approaches typically don't manage persistent data and require considerable effort to maintain as business processes change. Distributed object transactions typically require considerable over-head to map different object representations and often have a poor impact on system performance and reliability.

Our approach uses the concept of an information broker that mediates interaction between multiple enterprise systems, replicating data that needs to be shared from one system to another to provide very efficient access to it. The broker isolates the different system technologies required for data access and update and seamlessly translates data to and from a local enterprise system's data formats. The broker is notified of updates to replicated data and performs data translations into appropriate target remote system formats, applying appropriate remote system data update operations. Data inconsistencies, concurrent access and update and update failure are all handled by the information broker initiating appropriate compensation business process to correct. We describe a prototype system built using this approach to efficiently and effectively integrate a J2EE business-to-customer web enterprise application with several remote applications with very different integration technologies.

2. Motivation

Consider the enterprise system integration scenario in Figure 1. In this example, a J2EE-implemented "book broker" with JSP-implemented web interfaces and EJB-implemented business logic provides a one-stop portal for accessing the products of a group of book publishers [3, 4]. The broker obtains books from the publishers and formulates a catalogue. The customer purchasing system is populated with the catalogue and allows customers to buy books from any publisher with a single order.

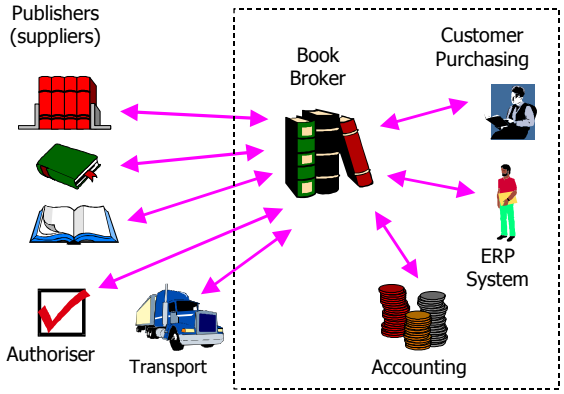

Figure 1. Book broker scenario.

The broker receives orders, validates them (including authorising payment from a bank) and sends (possibly multiple) orders to publishers. The publishers notify the broker of delivery details and the broker provides the customer system with tracking information from courier company delivery systems. The broker may also support information exchange between the customer system and ERP/accounting systems that the organization runs .

Key requirements of such a book purchasing broker for integrating these diverse enterprise systems include:

- Minimal (ideally no) modification should be necessary to the various enterprise systems
- The performance of the customer purchasing portal must be very good, without delays caused by accessing remote applications and databases across wide area networks
- Integration should be done in a consistent manner with no bespoke solutions for different technologies
- Systems should not have direct knowledge of each other and integration support be isolated
- Data and business process integration both need to be supported in the integration framework
- The integration approach should be scalable, reliable and provide secure control over business data.

Existing solutions that could be used in this scenario include the book broker querying and updating publisher data via remote database queries [6], the broker using on-demand web service queries [15, 18], EDI and XML messages [5, 19] or remote object accesses within distributed transactions [1, 2, 8, 22]. All of these approaches introduce major performance overheads if the interaction with the remote systems is done by the customer purchasing portal while the customer waits. Similarly, if a publisher system is temporarily unavailable then the customer can not access its data. All of these technologies may be provided by different publishers, requiring different bespoke integration solutions to be implemented by the customer purchasing system for each publisher. A virtual or heterogeneous database system [11, 13, 14, 20] could be used as a portal to the remote data, mapping data formats and database operations. However, again performance would suffer greatly if queries and updates are passed to the remote systems directly, and different integration technologies like messaging and distributed object APIs may not be supported. Various EAI and B2B solutions have been developed to overcome some of these problems, such as asynchronous messaging and XML document exchange [7, 9, 10], the J2EE Connector Architecture, and Business Process and Data Integration [8, 17, 21]. All are useful in some situations but many have the same performance impact e.g. the need to query several systems and assemble a result before customers can use it.

3. Our Approach

We have been developing an integration conceptual model which solves many such system integration problems by the use of three key techniques: the replication (copying) of data to be shared between systems; the optimistic access and update of this replicated data by other distributed systems; and the resolution of data or business process inconsistencies that may be introduced using a long running transaction protocol. Our approach for the book broker system is illustrated in Figure 2.

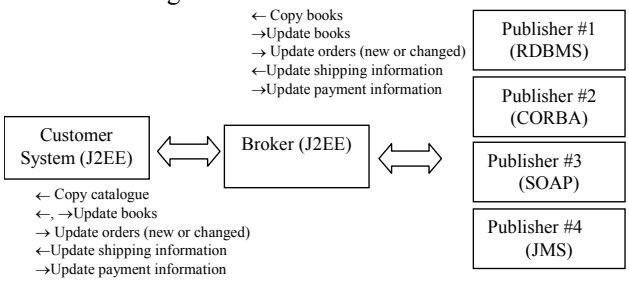

Figure 2. System integration via replicated datasets.

A broker is used to integrate publishers of books and a customer book purchasing system. The broker copies books (and authors, sales figures, delivery cost information etc) from publishers to the customer system, translating data formats as appropriate. The broker also copies orders for books generated by customers to publishers (not shown here), and book delivery tracking information from publishers to the customer system (not shown). The customer system accesses and updates such replicated data as if it is its own i.e. it "looks and feels" like customer enterprise system data. Updates may be made to some replicated data e.g. creating or updating orders, updating book sales committed, etc. Such updates are detected and translated by the broker into publisher system updates. Similarly, as data is changed in the

publisher systems the changes are detected and applied to data replicated in the customer systems e.g. number available, cost etc. Sometimes updates to replicated data are rejected by remote systems and we use a form of the Business Transaction Protocol (BTP) [16] to resolve this by having the broker detect this and ask the customer system to run "compensation" business transactions to correct any data inconsistencies.

4. A Supporting Architecture

The architecture of our approach for the book broker J2EE example application is outlined in Figure 3. A broker server acts as an intermediary between publisher and customer order management systems. The broker provides a temporary information repository and a set of interfaces to other systems. These interfaces perform data queries and updates on these external systems via a variety of communication technologies, ranging from RDBMS APIs and XML document mark-up to CORBA, RMI and message-oriented middleware. Some interfaces are generic and can be tailored to interact with different systems e.g. the RDBMS and XML document interfaces whereas others are specific to a particular system e.g. CORBA and RMI interfaces. All interface components map the external system data representation into an XML-encoded broker data format and vice versa.

Data is replicated from external systems e.g. a publisher and stored by the broker. Systems requiring access to data, such as the customer system, has the data translated into its own information model by the broker and its database updated via its interface component to the broker. Updates to this data by the customer system are detected by the broker's customer system interface component and copied back into the broker. The broker then translates this updated data into appropriate publisher system data updates via the appropriate publisher system's interface component. The broker implements a "long transaction" model across this replicated dataset, detecting when concurrent access/update to the dataset is made or when a system rejects updates to the dataset made by another system, and resolving these problems by invoking appropriate "compensation" business processes.

As can be seen in Figure 3 a range of communication technologies are supported by our integration approach. The broker uses an XML-based representation of datasets as documents, with information describing entity key correspondence, changes made to document elements, and document status (e.g. committed, pending commit, rejected). CORBA, RMI and DCOM-based interface components typically provide synchronous, remote procedure call APIs. Most require a hand-coded implementation of the interface that maps the broker XML dataset encoding to and from the remote system's business process data via sequences of API calls. SOAP and XML document exchange support asynchronous messaging and XML document encoding of remote system data. Java Messaging Service (JMS) provides a message-oriented information exchange protocol using either XML-encoded messages or binary-format messages. Database APIs provide synchronous table row query, insert, update and delete. Interactions with external systems are transactional i.e. the updating of a broker dataset may result in many remote system API calls which are carried out as one transaction.

Our approach makes as much use as possible of high-level business processes and subscribe/notify architectures in the integrated systems to provide this data-oriented systems integration. Some systems provide high-level APIs which allow the broker to enact remote system business processes, be informed immediately of data updates in the remote systems and to invoke remote system business processes to resolve data inconsistencies that may arise as part of its long transaction model. However, some systems provide only low-level, data query and update e.g. RDBMS interface, some XML document data encodings and some message-based systems. Some integration technologies support the remote system "pushing" data change notifications to the broker e.g. some CORBA, SOAP and messaging systems.

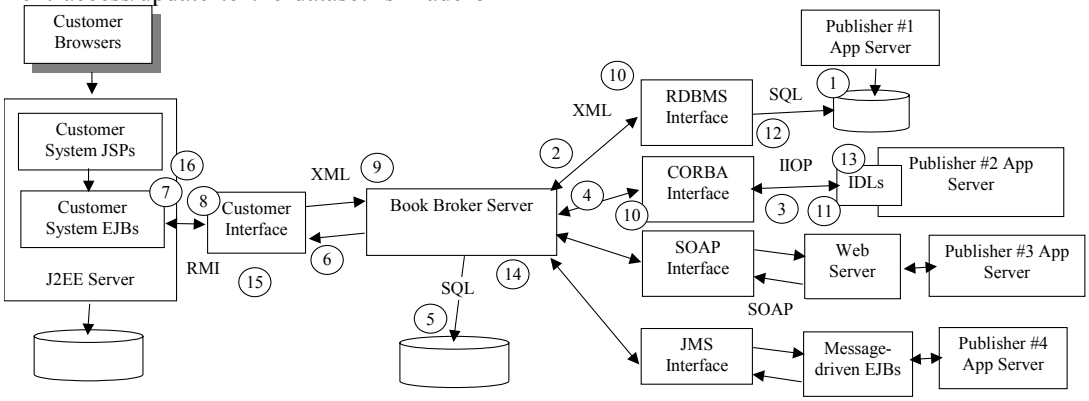

Figure 3. Architecture of our integrated system.

We briefly describe how integration of these distributed enterprise systems is achieved in such an architecture, demonstrating the range of integration technologies and system interactions supported. A RDBMS publisher interface component issues a query to obtain book data (1) which it translates into an XML document and sends to the broker (2). A SOAP-based interface is pushed changed book data messages (3) and it sends the changed book data (possibly after querying the publisher system for it) to the broker (4). The broker may temporarily store this data (5) and translates the remote system book data formats into the customer system book format, sending it to the customer system interface (6). The customer system's book data is updated via its API (7) (in this example via RMI calls to EJBs in its application server). Changes to book data e.g. number of items ordered are detected by the customer system interface (8), data and change information encoded and sent to the broker (9). The changed data is sent to the publisher interfaces by the broker (10) and is translated into SOAP messages sent to publisher #2 (11) and RDBMS database updates for publisher #1 (12). If the remote systems reject these updates e.g. database transaction fails (12) or notification message (13), or the broker detects concurrent access to the data (e.g. new changes from publishers time-stamped earlier than data change set from customer system) (14), message(s) must be sent to the customer system (15). The customer system runs appropriate business process(es) to resolve the inconsistency (compensation transactions) e.g. undoes the effects of its "invalid" book updates; updates orders for the book to satisfy violated publisher system constraints e.g. back-orders or cancels orders for out-of-stock order items; or informs the customer and asks them to e.g. modify their order due to inability to fulfil it as it stands.

5. Example Implementation

In order to provide a suitable book brokering system solution we have developed a design for the broker comprised of a number of components, illustrated in Figure 4. Remote system interfaces source or sink data in publisher systems using a wide variety of technologies (including both push- and pull-based querying, message subscription and API calling). When replicating publisher system data these remote interfaces translate their data sets obtained via database queries, file reads, message extraction and API calls into a common XML dataset format. This dataset contains data, data update operation and remote system keying information. Datasets are transformed into a canonical book broker information structure and then book broker data tables and “key tables” (mappings of system key information) are updated. Customer system data is updated when appropriate book broker data has been changed by this replication process. The book broker dataset is transformed into the customer system dataset and then the customer system remote interface applies data updates to the customer system via its API. When the customer system changes publisher system data that has been copied into its database, the broker detects these changes, translates its dataset into the remote publisher system data format and has the publisher system remote interface apply appropriate update operations to the publisher system (e.g. database updates, message construction or API calls).

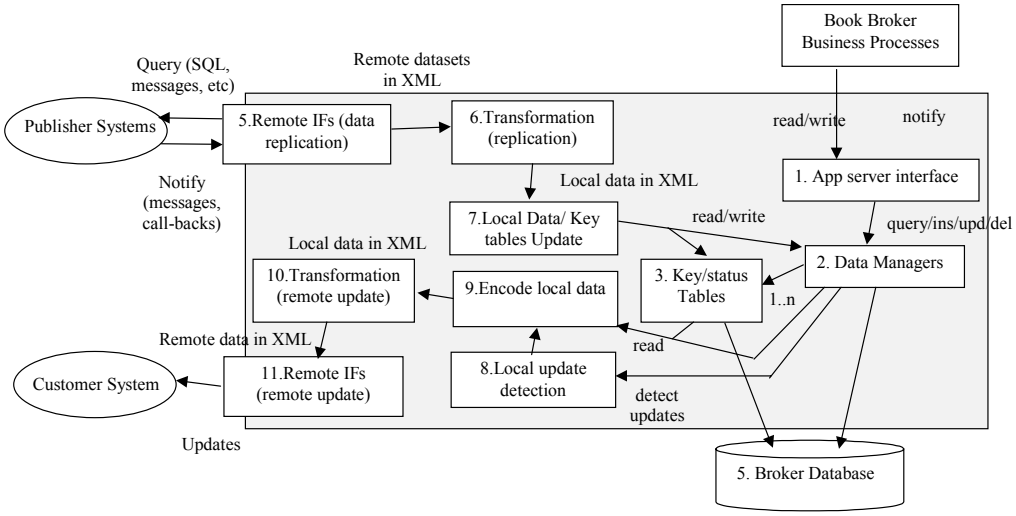

Figure 4. Broker architectural components.

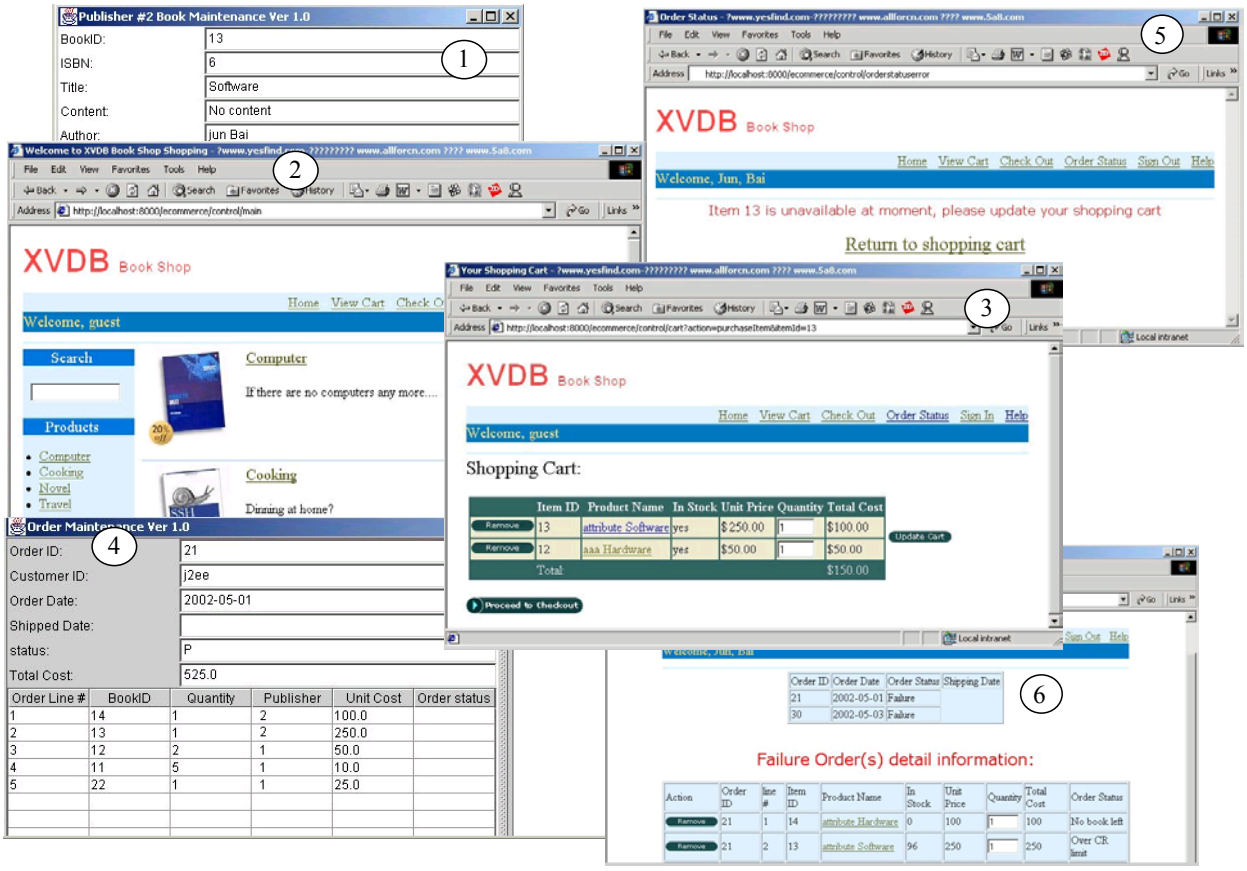

Figure 5. Example of customer system, book broker and publisher system interfaces.

We have built a prototype book broker that integrates a J2EE-implemented customer ordering system, a J2EE-implemented book broker, and four publisher systems, one providing a database API, one SOAP XML messaging, one CORBA IDLs and one Java Messaging Server messages. The customer on-line ordering system was developed as part of a previous project and is a generic on-line E-tailing system that we configured to provide book catalogue management, searching, ordering and tracking of book orders. No code changes were made to this system to integrate it with the publisher systems via the book broker server. Our book broker server's functions are modelled on those of a bespoke book broker application developed as part of a previous project [4]. Figure 5 shows an example of screen dumps from the running prototype.

After replicating book data from the publisher systems, an example shown in (1), the book broker formulates a canonical catalogue from all publishers' book information, copying this into the customer on-line purchasing system. Subsequent publisher system updates to catalogue items detected by the broker are made as updates on the customer catalogue data. The customer system provides numerous web-based interfaces, including book search (2) and a shopping cart metaphor used to assemble orders for books (3). Upon "checking out" the new order is detected by the book broker server and its data copied. The book broker splits this into an order for each publisher whose books have been ordered, with an example shown from one publisher system (4). Updates to publisher orders e.g. shipping information are detected by the broker and copied to the customer system.

Problems may occur due to our integration strategy's adoption of asynchronous, optimistic data exchange. These include: data being used in a system's transaction is modified by its "owning" system and updated in the broker; updates made by a system to remote data can be rejected by that remote system; and data updates can "cross" between the publisher and customer systems i.e. be made concurrently. The book broker server implements a "long transaction" model, using a form of the Business Transaction Protocol, to resolve

inconsistencies that may occur due to these problems. The book broker may take several approaches to resolving inconsistencies where the book record is in use in a customer system transaction: inform the user after check-out (or during ordering) of changed values or unavailability of the old book information (5) or may delay the update of the book information and send an order to the publisher using previous information (possibly having this rejected as e.g. insufficient payment provided). Rejected orders e.g. invalid data (old data used), insufficient payment, customer over credit limit or book(s) no longer available are detected by the broker and have a status associated with the order and/or order lines indicating reasons for failure. This information is presented to the customer when appropriate (6). Automatic resolution strategies can be employed by the broker or customer system in some situations e.g. invalid/out of stock book order lines deleted; back-orders automatically created for out-of-stock books, or email messages sent to the customers.

6. Discussion

The integration architecture provided by our book broker server allows for a very diverse range of publisher systems to be integrated with the on-line customer purchasing system in a seamless manner with none of the integrated systems aware of the others with which they are integrated. We began by integrating the customer purchasing system via its Enterprise JavaBean APIs and a publisher system via its database API. We subsequently added the CORBA, SOAP and JMS-implemented publishers. The addition of further publisher systems did not impact the operation of the existing integrated systems or require modification to their integration interfaces. Building some integration interfaces was quite straightforward e.g. the relational database and SOAP interfaces, but others were quite complex e.g. the EJB and CORBA interfaces, due to the need to handle distributed transactions across these interfaces and a sometimes long sequence of remote object calls. Some interfaces were remotely deployed as "remote integration agents" (e.g. for some database and JMS systems) on the remote system host to provide better performance and security.

We have run several performance tests on our book broker system, analysing the time taken to replicate data and perform remote updates of data. These operations normally run asynchronously but the additional loading on both the broker and integrated systems of interaction with the integration interface components can become significant. We found with our prototype that the broker could acquire thousands of books, even those represented using complex structures, within a few seconds, even with moderately loaded publisher systems to query and customer system to update. Order querying from the customer system and order update in publisher systems requires significantly more processing as much business logic and database updates need to be performed. However, the number of orders processed is far less than book information transfer and we found performance perfectly adequate. For example, Figure 6 shows results of one of the performance tests we conducted to gauge broker performance under various loading conditions. In this example, we are timing the construction of orders in the customer system and their export to the broker when the customer system is both unloaded and when more and more concurrent transactions with less and less time between transaction requests are made. This shows that with heavy (over 90% CPU) loading of the customer system the broker can receive orders in reasonable time.

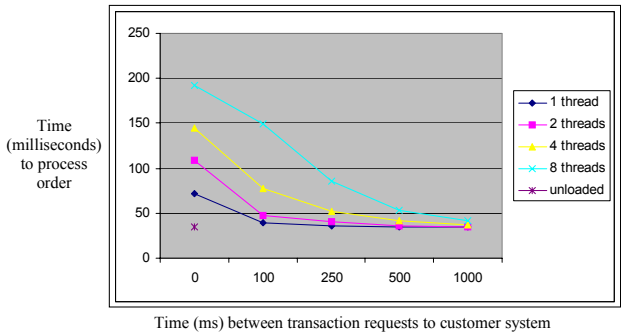

Figure 6. Performance of broker under loading.

Our approach to resolving inconsistencies that occur because of the asynchronous data replication/update approach has worked well in this application domain. Most of the time inconsistent data usage between systems does not occur as data changed in one system is updated via the broker in another system before it is accessed or further updated. Some inconsistencies turn out not to matter in practice. For example, when updating the number of books sold in the customer system, this change is passed through the broker to the book publisher system as a relative update (+ or – change) to the publisher's book attribute. Even if this value is different to the number of books sold in the publisher (as publisher direct sales may have changed it), the relative update almost always works. In this scenario, even if the change is rejected by the publisher system, the current value in the publisher system need only be replicated back into the customer system (as it is the definitive value) and will be properly updated by the publisher system. Even where inconsistency resolution is very difficult, the systems can inform the user and let them manually start further business transactions.

Compared to the bespoke book broker prototype our system was modelled on [4], our approach has numerous advantages. Both data acquisition and update are

supported using a homogeneous approach and new systems can be seamlessly added with a much reduced development effort. Multiple threading and process can be used to provide additional scalability and performance. Data inconsistency is handled using business transactions after detection by the broker. We are currently implementing a generic integration broker application and integration agents that can be used for a wide range of system integration problems. This will include configuration tools allowing data model and data operation correspondences to be specified declaratively and some remote integration agents to be declaratively configured. We are also experimenting with presenting users with richer information about broker-moderated remote data e.g. showing them additional status information from the broker such as "data committed in local system but not remote system" or "previous remote data change rejected".

7. Summary

We have developed an information brokering system that allows a wide range of systems to be integrated in a seamless fashion. Integration interface components isolate remote system communication technologies and data models. Interaction can be at data read/write or higher level messages or API calls corresponding to business process enactment. A broker server replicates and updates information between integrated systems. An optimistic long-transaction model is used to allow high performance, reliability and scalability of the approach. Existing enterprise system business processes are used to solve data inconsistencies that may arise. We have successfully prototyped a system using this approach to integration.

8. References

1. Alonso G, Fiedler U, Hagen C, Lazcano A, Schuldt H, Weiler N. WISE: business to business e-commerce. Proceedings Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises. RIDE-VE'99. IEEE CS Press, 1999, pp.132-9. Los Alamitos, CA, USA.
2. Aleksy M, Schader M, Tapper C. Interoperability and interchangeability of middleware components in a three-tier CORBA-environment-state of the art. Proceedings Third International Enterprise Distributed Object Computing. Conference, IEEE CS Press. 1999, pp.204-13. Piscataway, NJ, USA.
3. Blackham, J., Grundeman, P., Grundy, J.C., Hosking, J.G. and Mugridge, W.B., Supporting Pervasive Business via Virtual Database Aggregation, In Proceedings of Evolve'2001 – Pervasive Business, Sydney, Australia, March 15-16 2001, DSTC Press.
4. Bloomfield, D., Amor, R. and Grooman, M. The Evolving CONNET Gateway to European Construction Resources, Proceedings of the CIB W102 conference, Melbourne, Australia, 26-27 March 2001.
5. Cheung, D., Lee, S.D., Lee, T., Song, W., Tan, C.J. Distributed and scalable XML document processing architecture for E-commerce systems. In *Proceedings of the Second International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*. IEEE, 2000, pp.152-157.
6. Emmerich, W., CORBA and ODBMSs in Viewpoint Development Environment Architectures, in Proceedings of the 4th International Conference on Object-Oriented Information Systems, Springer Verlag, 1997, pp. 347-360.
7. ebXML Group, ebXML documentation, http://www.ebxml.org/.
8. eXcelon Corp, eXcelon B2B Integration Server White Paper, www.exceloncorp.com.
9. Goulde, M.A. Microsoft's BizTalk Framework adds messaging to XML. *E-Business Strategies & Solutions*, Sept. 1999, pp.10-14.
10. Grundy, J.C., Mugridge, W.B., Hosking, J.G. and Kendall, P. Generating EDI Message Translations from Visual Specifications, In Proceedings of the 2001 IEEE Automated Software Engineering Conference, San Diego, CA, 26-28 Nov 2001, IEEE CS Press.
11. Gupta, A. Harinarayan, V. Rajaraman, A. Virtual database technology, Proceedings of the 1998 14th International Conference on Data Engineering, 23-27 Feb 1998, 297 – 301.
12. IBM Corp, *MQ Series Integrator*, www.ibm.com.
13. Idenhen, K., Introducing OpenLink Virtuoso: Universal Data Access Without Boundaries, White paper, www.openlinksw.com.
14. Lim, E.P. and Chiang, R.H.L. The integration of relationship instances from heterogeneous databases. Decision Support Systems, vol.29, no.2, Aug. 2000, pp.153-67. Publisher: Elsevier, Netherlands.
15. Morgenthal, J.P. XML: The New Integration Frontier, *EAI Journal*, Feb. 2001, www.eaijournal.com.
16. Oasis Group, OASIS Business Transaction Protocol 1.0, June 2002, www.oasis-open.org.
17. OrderWare Inc, OrderWare Data Transformation Manager White Paper, June 2002, www.orderware.com.
18. Peyret, H. Mission-critical Web Services: Plan for Long-running Transactions, Giga Information Group, 2002.
19. Swatman, P.M.C., Swatman, P.A., Fowler, D.C. A model of EDI integration and strategic business reengineering. *Journal of Strategic Information Systems*, vol.3, no.1, March, 1994, pp.41-60.
20. Uchoa, E.M.A. and Melo, R.N. HEROS: a framework for heterogeneous database systems integration. Database and Expert Systems Applications. 10th International Conference, DEXA'99, Lecture Notes in Computer Science Vol.1677, Springer-Verlag. 1999, pp.656-67. Berlin, Germany.
21. Vitria Technolgy Inc, Vitria BusinessWare White Paper, www.vitria.com.
22. Wu, E. A CORBA-based architecture for integrating distributed and heterogeneous databases. Proceedings Fifth IEEE International Conference on Engineering of Complex Computer Systems, IEEE CS Press, 1999, pp.143-52. Los Alamitos, CA, USA.