Mobile Agent Technology in Support of Sales Order Processing in the Virtual Enterprise

T.Papaioannou and J.Edwards
MSI Research Institute
Department of Manufacturing Engineering
Loughborough University
Loughborough, Leicestershire, LE11 3TU, UK
Tel: +44 1509 228250 Fax: +44 1509 267725
Email T.Papaioannou@lboro.ac.uk or J.M.Edwards@lboro.ac.uk

Abstract

The flexible integration of a range of disparate IT applications is a key requirement for today's global enterprises. The virtual enterprise, formed by a collection of collaborating companies for short term, high return, one off projects provides perhaps the most extreme example of this need. It is unlikely that any collaborators in a virtual enterprise will have similar networks or software, but the requirement exists for them to inter-operate.

This paper proposes the notion that Mobile Agent Technology can be a significant aid to enterprise agility, particularly where distribution of information is a feature, as in virtual enterprises. The implications of using Java and its facilities for database connectivity (JDBC) together with mobile agent environments are discussed before a model to fulfil the requirements of the manufacturing Sales/Order process is proposed. The model used in the process has been produced with data collected from an industrial case study.

The subsequent decomposition of the Agent types involved reveals an abstract pattern for database query using agent technology. Finally, the tools required by an enterprise wishing to utilise agent technology are discussed.

Keywords

Virtual Enterprise, Mobile Agents, System Integration, Agile Systems

---

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35390-6_58

L. M. Camarinha-Matos et al. (eds.), *Intelligent Systems for Manufacturing*

1 INTRODUCTION

Access to an organisation's store of information is a critical factor in developing business systems. Whether client/server applications or monolithic, host-based systems, the information stored in an organisation's database is its lifeblood. The ability to effectively manage, manipulate, and distribute this information was once viewed as a provider of competitive advantage [Wright, 1996]. Today it is simply a base requirement for corporate existence within the global market place.

Globalisation and the advances in Information Technology (IT) systems have led to the emergence of the Virtual Enterprise [Ball, 1997]. This type of enterprise is made up of a number of co-operating companies who are generally physically distributed but work together to meet some market demand. As the operations of the enterprise are distributed geographically, so must the information systems that support them [SSA, 1995]. Typically, these relationships are short-term collaborations on one off, high return projects where time to becoming effective in the market is of the essence. Management of information remains a base requirement of these virtual enterprises, who are therefore charged with the integration of several separate IT systems to form an operational system in as short a time as possible. One of the key issues is then, how to handle system component distribution.

2 DISTRIBUTION

The main protagonists offering solutions to the problems of distribution are the Object Management Group (OMG) with their Common Object Request Broker Architecture (CORBA) and Microsoft with their OLE/DCOM product. The key difference being that CORBA is a specification (met by several proprietary commercial products) and DCOM is a single vendor product. When dealing with distribution these technologies are being challenged by the relative simplicity of Internet technology.

2.1 CORBA

CORBA is the product of a consortium of over eight hundred companies, known as the Object Management Group (OMG). The OMG has approached the problem of handling the interaction of distributed components by creating interface specifications, and ***not*** code. Distributed components of the system are able to describe their interfaces using the Interface Definition Language (IDL) and subsequently interoperate through the underlying Object Request Broker (ORB). It is the ORB that provides the communication backbone through which all of the components are able to interact. The OMG contend that components communicating via an ORB do not need to be aware of the mechanisms used in that communication, and in fact are able to discover each other at run time allowing extensive flexibility and configurability. However, the reality is that in the currently available commercial ORB's all that can be discovered at runtime are a component's methods and arguments as described by its IDL definition. This is insufficient for the realisation of systems based on loosely coupled components using late binding.

2.2 OLE/DCOM

OLE and the **Distributed Component Object Model** (DCOM) are Microsoft's attempt to provide a software architecture that allows applications to be assembled from binary software components and interoperate across a network. DCOM is the underlying architecture that forms the foundation for higher-level software services normally provided by OLE. OLE is a unified environment of object-based services with the ability to both customise those services and arbitrarily extend the architecture through custom services, with the overall purpose of enabling integration between components. OLE services span various aspects of commonly needed system functionality, including compound documents, custom controls, inter-application scripting, data transfer, and other software interactions.

2.3 Internet Technology

Internet technology has proved to be effective for connecting a mix of different types of computers and computer networks, while also providing location independence to information. Further, analysts predict that the next five years will see the evolutionary convergence of the Internet, Intranets and traditional IT models such as client/server and peer to peer [Ball, 1997]. It would thus appear to be an appropriate approach for building integrated, distributed IT systems to support the virtual enterprise where worldwide distribution is an important consideration. However serious problems with this distribution technology remain.

2.4 Distribution Problems

The saturation of network bandwidth, especially when part of the network in question is the Internet, means that remote database access as required by the virtual enterprise model could mean ineffective IT support for the business. As well as data timeliness, factors such as, data integrity and security are also a concern when dealing with the Internet.

Mobile Agent technology may be able to overcome this set of problems through local interaction [Clements, 1997]. It is equally applicable to the problem of geographically distributed information sources, since mobile agent systems are inherently distributed. This paper proposes an agent-based model for supporting the Sales/Order process within a virtual enterprise having first discussed the implications of Java and its facilities for database connectivity known as JDBC*. The subsequent decomposition of the agent types involved identifies key agent types of this model and proposes an abstract pattern for database query using agent technology.

3 MOBILE AGENT TECHNOLOGY

It is the features of local interaction, reduced network loading, server flexibility and application autonomy which are supported by mobile agent technology that can help to provide a level of agility above that provided by more conventional IT technology [Clements, 1997].

3.1 Java and Mobile Agent Environments

The agents discussed in this paper can be classified in line with Franklin and Graesser [Franklin, 1996] as goal oriented, communicative, and mobile i.e.:

- Goal oriented – they do not simply act in response to the environment
- Communicative – they are able to communicate with other agents
- Mobile – they are able to transport themselves from one host to another.

In order for these agents to exist within a system or to themselves form a system they require a framework for implementation and execution. This is known as the agent environment.

It can be argued that Java has become the *de facto* language for writing mobile agent environments. Systems using it include Aglets, Voyager, Mole, JATLite and Concordia. The use of this platform neutral language provides the portability demanded by the mobile agent paradigm. The platform neutrality forms a good basis for enterprises requiring system agility, or wishing to form virtual enterprises, although accessing legacy systems is still an issue. This problem has been recognised, and the recent availability of facilities for database access via the Java JDBC interfaces has improved conditions.

---

* Contrary to popular opinion JDBC is not an acronym for 'Java Database Connectivity', but a name trademarked by Sun Microsystems

3.2 JDBC

Java Version 1.1 introduced the JDBC Application Programming Interface (API); a set of classes and programming interfaces for executing Structured Query Language (SQL) statements which are modelled loosely on Microsoft's Open Database Connectivity† (ODBC) standard. JDBC allows Java programs to access ODBC compliant databases in pure Java, without using C++ stubs, which makes it possible for developers to write database applications using a pure Java API. JDBC is a '*low-level*' interface allowing developers to invoke SQL commands directly and providing a base upon which to build higher level interfaces and tools.

Direct access to ODBC from a Java program has previously been achieved by using the JDBC-ODBC bridge supplied free as part of the standard Java Developers Kit (JDK)‡. However, there are several key issues that JDBC addresses with respect to ODBC access with Java that provide incentives for using the new approach:

- ODBC uses a C interface, and calls from Java to native C code produce drawbacks in security and portability;
- There are language differences between C and Java e.g. Java does not use pointers, whereas ODBC uses them extensively;
- When using ODBC the driver manager and drivers must be installed manually on every client machine. If the driver is to be portable and secure on all Java platforms then the driver must be written solely in Java.

In addition, one of the major advantages of mobile agent technology is server flexibility [Clements, 1997]. With a pure Java JDBC driver it is possible to serialise and encapsulate the driver within a mobile agent and dispatch it to a remote server, where it can be automatically installed and loaded. This could have advantages for enterprise integration where changes to a remote database server can be achieved with a single agent.

Since mobile agent technology is being proposed by the authors as an integration tool, and the majority of Agent Environments are written in Java, it too must be capable of utilising these technologies. In consequence, JDBC plays a pivotal role in the proposed pattern for a generic query agent.

4 SCENARIO

The scenario upon which the applications work in this paper is based was derived from data collected during a case study of a leading vacuum component manufacturer based in the UK. A simplified model, subsequently generated, describes their Sales/Order process.

4.1 Scenario Explanation

An order is placed with the company by a customer. This generates a **Sales Order**, which has a Product ID associated with it. A query can now be made to the **Stock Database** requesting allocation of the full product, or subparts if assembly is still required. Successful completion returns the allocation to the sales order with a possible delivery date. If the order can not be supplied, it is possible to generate a **Bill of materials (BOM)** from the Product ID. Encapsulated within the BOM are details of all the subparts (if any) of the product. A subsequent query is then made to the database requesting allocation of raw materials to manufacture any or all of the subparts not fulfilled. This allocation is then associated with a newly generated **Works Order**, which is subsequently dispatched to the production scheduler.

† The ODBC API is based upon the Call Level Interface defined by the SQL Access Group and is endorsed by X/Open.

‡ For more information about different styles of drivers and their availability, see the JDBC WebSite at http://www.javasoft.com/jdbc

4.2 A Mobile Agent Model for the Sales/Order Process

In the Sales/Order model there are four key component types; stock, sales, orders, BOMs and works orders. The proposed agent architecture reflects this real world model but concentrates on the sales and ordering process, requiring three different agent types, one mobile and two static. It is understood that there will also be a need to generate new agents for Purchase Orders or internal Works Orders, when an ordered product is not currently in stock.

The model is based on the scenario faced by the vacuum component manufacturer. Sales outlets are distributed around the world, whilst the central stock warehouse and manufacturing plants are based in the UK. The company has a requirement to add new sales agents to their existing network and possibly new storehouses. For this to be achieved there is a need for an agile and flexible IT system. This model could equally be applied to the formation of a virtual enterprise.

An order is placed by a customer; the Sales agent then generates an Order agent that is dispatched to the StockControl agent and requests the fulfilment of its order by passing over an `Order` object. The StockControl agent queries the stock database to see if enough products are in stock. If so, it then returns a `DeliveryDate` object to the Order agent which itself returns to its parent Sales agent. If there are not enough products in stock to satisfy the order, the StockControl agent uses the Product ID encapsulated in the `order` object and queries the BOM database for a list of subparts or raw materials required. This is then encapsulated within a WorksOrder agent and dispatched whilst the Order agent returns with a `DeliveryDate` object containing a standard delivery date[§]. If there are not enough raw materials in stock, the StockControl agent generates a PurchaseOrder agent that encapsulates details of all the required materials.

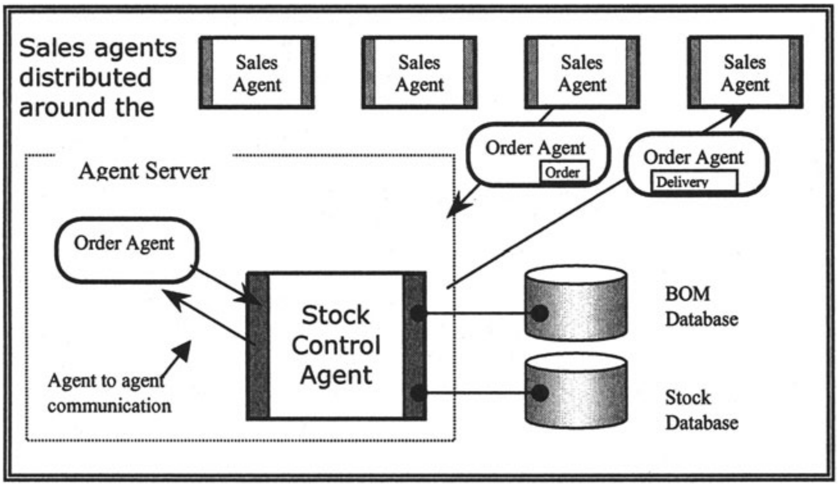

Figure 1 Proposed Agent Model for Sales/Order Process

The three key agent types identified are Sales Agents, Order Agents and StockControl Agents. These are discussed in the following section.

§ In the case of the manufacturer in question, a standard delivery date is three weeks from the receipt of the order.

5 AGENT TYPES

5.1 Sales Agents

Sales Agents are static, GUI based agents that are responsible for generating Order Agents and dispatching them to the StockControl Agent. These could be resident in a very slim client for a bank of sales persons working on terminals or NetPCs, or even hosted in a laptop for a travelling sales person. Must be capable of keep track of all current orders that have been placed.

5.2 Order Agents

Order Agents are mobile, encapsulating a single order per agent; they are responsible for completion of the Sales/Order process for that order. Valid outcomes could be reporting to the Sales Agent a delivery date for the product or an allocation for materials and an internal works order number. Order Agents have no GUI.

5.3 StockControl Agent

The StockControl Agent is static, with no GUI**. It is responsible for handling all requests for products, parts, or materials, and thus must interface to the stock control database. If the current stock levels are unable to satisfy an order, it must be able to use the encapsulated Product ID to derive the Bill of Materials. How this is achieved would depend on the existing IT system. The BOM's may be kept in a separate database, or be part of the same legacy system. There will also be a need to generate new agents for Purchase Orders or internal Works Orders, when the ordered product is not currently in stock.

When generating StockControl agents to unify the variety of database systems typically seen within a virtual enterprise it became apparent that some required aspects were particular to each database, whilst others were generic to all StockControl agents. In considering this problem the use of a generic 'Database Query Agent' was conceived which could be used as a base pattern for all StockControl agents in the system. The advantage of such a technique is the consistency and reusability inherent in using a pattern from which to build. The generic Database Query Agent discussed in the next section.

6 QUERY AGENT

Whilst investigating the requirements for the StockControl agent, it became clear that there was always a need for the Agent to be capable of querying an enterprise's different databases. Further, if the proposed system was to truly enable system agility, then it must be capable of querying a variety of new or legacy databases, not just ODBC compliant ones, since in the main, legacy systems are not ODBC compliant.

By taking a modular approach to the design of the Query Agent and using established OO principles, the authors have derived an effective and reusable agent pattern that affords a flexible and extensible infrastructure to the system designer. The Query Agent can be decomposed into several key components, as shown below.

6.1.1 The Infrastructure

This is dictated by the agent environment for which the Query Agent is being developed. It must include mechanisms for dispatching, retrieving, shutting down and restarting agents in a suitable host. Most of this functionality is generally to be available through the host environment or by subclassing from abstract classes provided by the environment. For example, in the authors' implementation using the Aglets Workbench it is usual to extend the abstract Aglet class, and then provide implementation specific details if required.

** Note: No GUI refers to the unattended running mode of the Agent. A hidden GUI containing a configuration screen could be very useful for a system administrator when installing the agent.

6.1.2 The Identifier

The Identifier plays an essential role in system security. Whilst it is more usual for mobile agents to have the need to carry an Identifier, static agents must also be able to prove their credentials. A key role of the StockControl Agent is to generate PurchaseOrder Agents and WorksOrder Agents in order to fulfil unsatisfied orders. Part of the Identifier is handed to these child agents, as proof of is origin on dispatch to another host.

6.1.3 The Comms Package

In the ever-increasing array of agent environments available, communication methods vary widely, as do the communication requirements of differing implementations. In some examples, simple `String` matching is sufficient for communication. This can be seen in basic Aglet examples. The problem of achieving semantic level communication between agents is the subject of much research. The use of KIF and KQML[Finin, 1997] is typical of the more advanced approaches that are being proposed. To handle this variety of communication methods, the proposed pattern includes a 'plug and play' Comms Package that can be interchanged by the designer at will with respect to the system requirements. Its role is to receive the incoming communication from arriving or querying agents and translate this into a format the Business Logic Unit or Database Handler can understand.

6.1.4 Business Logic Unit

In the Sales/Order scenario when an Order Agent is dispatched by the Sales Agent, it encapsulates an Order object. On arrival at the StockControl agent, it is tasked with attempting to fulfil the encapsulated order. This in itself requires some simple rule based logic. Since all the Order agents will require this same logic it is clear that it is better suited to being part of the StockControl agent. By simply handing over the order object and awaiting an answer, the size of the Order agent is kept small, reducing network traffic.

Depending on the requirements and implementation of the Business Logic Unit the Query Agent can in fact be viewed as autonomous or even 'intelligent'; if the rule based logic was replaced with a neural net for example.

6.1.5 The Database Handler

The Database Handler deals with all the details of connecting to a database, retrieving information, updating the database, or even switching databases transparently to the requesting agent. It works in tandem with the business logic unit to fulfil the request of the Order agent.

Figure 2 clearly shows the benefits of adopting a modular approach to the design of the Query Agent. The examples shown address a large percentage (but by no means all) of the real world situations and the methods currently being employed in the use of databases within an enterprise. The modular Database Handler can easily be interchanged for whatever the situation requires.

In the case that a virtual enterprise already running a mobile agent system requires a new collaborator, accessing the new information contained within their databases requires solely the production of a new Database Handler. If their IT system is sufficiently advanced this could be as easy as linking a JDBC-ODBC bridge to their middle tier, and providing a host for the agent. Very different from the bespoke solutions prevalent until very recently.

In the authors' experience, it is the connection of the Data Handler to a new database, and the required configuration that poses the main problems to a system designer.

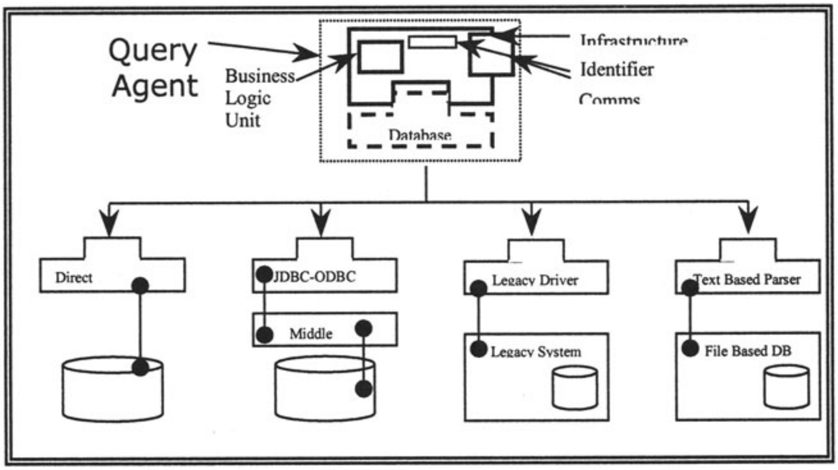

Figure 2 The Query Agent pattern and examples of DataHandler modules

6.2 DataHandler Implementation

For the implementation of the StockControl agent and its resident DataHandler a mock database was created using Microsoft Access. Its contents were based on a products and materials list obtained during the case study. This database was then made globally accessible using the dbAnyWhere server available from Symantec. The mobile agent environment used to develop and test the implementation was the Aglets Workbench, and the StockControl agent was hosted within the Tahitti server supplied with the Aglet package. It was decided that the best way to access the database was to use the driver supplied by Symantec and the JDBC-ODBC Bridge. The implementation was also tested using the native ODBC support present in the Win32 platforms.

In the construction of the two DataHandlers a common set of functionality emerged which evolved into two distinct Java classes that were very light weight, but invaluable in connecting and testing database connections. They are `ConnectionInfo` and `DBConnector`.

6.2.1 DBConnector

This class is used to perform all operations in connecting to a JDBC compliant data source. For systems that allow it, the driver name can be put into `System.properties`. If not, the driver is just loaded using `Class.forName`. Connection details are encapsulated within an instance of `ConnectionInfo`.

6.2.2 ConnectionInfo

This class is used to encapsulate all Connection details required to connect to a JDBC complaint database. Normal use is with the `DBConnector` class. It supports serialisation in order to allowing saving the details to file.

In the construction of the Sales/Order agent architecture, it became apparent that to the untrained the hardest part of setting up a StockControl agent was in making the connection, via the DataHandler, to the database. Whilst on the surface a relatively simple task, there are several variables that must be set correctly, and a number of JDBC interfaces that must be used accurately. Continual tweaking of the parameters in an ASCII file soon became extremely tedious.

To alleviate the problems this caused the DataConnector tool was produced to automate some of these tasks.

7 DATACONNECTOR TOOL

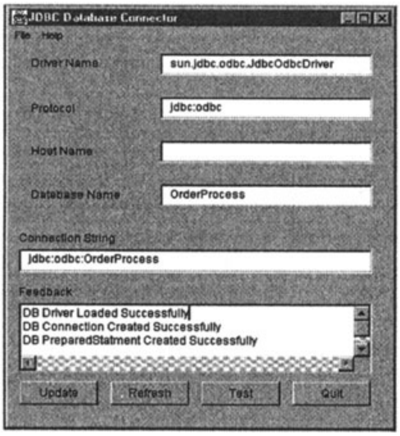

Figure 3 Screenshot of DataConnector

The DataConnector Tool is a simple Java program, with a basic UI that allows the user to insert the relevant parameters for connection to a data source. The validity of these parameters can then be tested, and if need be, tweaked and tested again using the refresh, update and test facilities. Once a satisfactory connection has been made, the data can be saved as a serialised java class file.

In fact, the functionality of the DataConnector was achieved very simply by linking a UI to the `DBConnector` class. The use of this tool decreased the time taken to set up a new DataHandler, and allowed a set of `ConnectionInfo` files to be built up.

7.1 Benefits of DataConnector

The biggest advantage in using this tool is the ability to test connections to a database and server across the network, or even the Internet. If a virtual enterprise were to decide to use mobile agent technology as a tool for rapid integration, it is likely that one of the collaborators (or their systems administrator) will have some prior experience in using the technology. The DataConnector tool allows a single administrator to test all the required database connections between the relevant systems, and produce a set of `ConnectionInfo` files that can be forwarded to the respective sites. Moreover, if the agent environments and servers have already been set up, a Messenger agent could deliver the files, and the DataHandlers completed and initialised automatically. The light weight nature of a `ConnectionInfo` file means that continued use of the agent system would allow an administrator to build up a set of predefined files for various configurations which would accelerate the speed with which new collaborators or data sources could be added in the future, increasing the system agility of the enterprise.

8 CONCLUSIONS

This paper proposes the notion that mobile agent technology can be a significant aid in the rapid formation of the information systems that support virtual enterprises. The adoption of Java as the *de facto* language for mobile agent environments, and its database connection capabilities provided by the JDBC interfaces, ensures mobile agents are fully capable of integrating with legacy and ODBC complaint data sources.

A model generated from a case study identifies three key types of agents required in the Sales/Order process. The authors propose a pattern for database query using agent technology, and a tool to aid rapid integration of existing data sources.

Implementation based upon this pattern has provided flexibility additional to that inherent to the agent paradigm.

ACKNOWLEDGEMENTS

The authors thank the EPSRC for their continued support of the work.

REFERENCES

Ball et al, 1997, "*Enterprise Enablement for Java Applications*", XDB Systems.

P.E.Clements, T.Papaioannou, J.Edwards, 1997, "*Aglets: Enabling the Virtual Enterprise*", Published and Presented at ME-SELA '97. p425, ISBN 1 86058 066 1

Hamilton et al, 1997, "*JDBC Database Access with Java*", Addison Wesley.

System Software AssociatesInc., "*BPCS Client/Server Distributed Object Computing Architecture*", 1995

D.T.Wright, N.D.Burns, "*Impact of Globalisation on Organisational Structure and Performance*", Proc. of the Organisational Management Division, International Association of Management 14th Annual Conference. Toronto, Canada, August 2-6, 1996, pp. 58-63

Franklin, S and Graesser, A., 1996, "*Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*", Proceedings of the 3rd Int. Workshop on Agent Theories, Architectures, and Languages, Berlin, Springer-Verlag.

Finin, T., Labrou, Y., Mayfield, J., 1997, "*KQML as an agent communication language*", MIT Press, Cambridge, to appear. Available at http://www.csee.umbc.edu/kqml/papers/

BIOGRAPHY

Todd Papaioannou received a BEng (Hons) in Product Design with Manufacture from Loughborough University, before becoming a member of the MSI Research Institute. He is currently studying for his PhD. His main research interests lie in mobile agent technology, distributed components and enterprise integration. A list of his papers can be found at http://luckyspc.lboro.ac.uk/Docs/Papers/

John Edwards gained his PhD from Loughborough University in 1994. Having spent 13 years in UK process and manufacturing industry, being involved in the creation of computer control and information systems, he joined Loughborough University in 1987. During his 11 years at Loughborough he has been involved with the Systems Integration Group and is now a member of the MSI Research Institute where his role is as principal investigator on a number of UK government funded research Initiatives.