Implementation of High-volume Reliable Intelligence E-business

Zhong Zhou[1] Wen Jun[2]

[1] Computer Science & Technology Institute in UESTC, Chengdu 610054, P. R. China
[2] Computer Science & Technology Institute in UESTC, Chengdu 610054, P. R. China

Abstract

E-business intelligence and reliability are the crucial step for companies' huge volume data processing. This paper presents a transaction middleware and web server working together supported by business intelligence mechanism, which can get a reliable scaleable plug-in based system. The efficiency of data-processing will be reduced. It can facilitate both the developers and consumers.

Keywords: Transaction middleware server, Business intelligence, Resource recovery Mechanism, Logical Unit Work

1. Introduction

Recently, we have been witnessing the High-volume E-Business Processing plays a central role in the daily operations of most of the world's largest corporations for its powerful data processing capability and high security. Although there are many forms of computing solutions are used extensively in various business capacities, the Reliable Intelligence occupies a coveted place in today's e-business environment. For example finance, health care, insurance, public utilities, online-business, and a multitude of other public and private enterprises, the Reliable Intelligence continues are going to form the foundation of modern business. No doubt, the processing capability of modern computing system is powerful, but it is hard to say this has implemented a perfect solution in today's e-business environment especially when focusing on Reliable Intelligence. Some flaws exist in current solutions.

Firstly, the reliability and intelligence are two different concepts, which means they have their own requirements, how to merge them into an integrated solution is much more difficult than we had expected before. In the e-business environment, transactions become even more important since, almost by definition, e-business has to be thought of in terms of pervasive computing. That is, resources can be spread across several computers, and many different types of servers can be succeeded in a single request. It is very important to have architecture capable of "tying up" all those related changes and reverting them all if one of them did not complete successfully. Therefore we must abstract customers' requests into transactions. For example, Mr. A sent the points (100$) in his credit card to Mr. B's card, when the transaction happens, there are two "writing-operation" in the database (1. subtracting 100$ from Mr. A's personal account; 2. Adding (100 $) to Mr. B's account ;), the two writing-operation must been considered as one transaction. The BI (Business Intelligence) web/application is a process-driven architecture, process is the basic unit in BI application server. The TM server (Transaction Middleware server) is a transaction-driven architecture. Now we can use a special adapter which can transform the customers' request into transaction. Now, when a customer requests come, the BI web/application server deals the requests with its BI capability and sends these requests to TM server through the adaptor.

Secondly, it seems to be a good idea when we only focusing on the security and reliability. However, if considered from other perspectives, some potential problems may arise. For example, when a transaction accesses some resources through TM server usually costs more than accesses it directly, in addition, this may have some impact on its efficiency. As we known, the more secure and more complex, the cost for the transaction will be more expensive, which may lead to a dilemma:when we are considering both the cost and efficiency. Therefore, the high-level reliabilities is performing at costing the efficiency, how to make a balance between two of them is a important problem; However, if we make a more careful analysis on such problems, we find that not all the transaction needs most security protect mechanism. For example, Mr. Black needs a financial transaction such as online trading, now he wants to buy a notebook PC, the commercial system will do two things during the transaction that is change both the Mr. Black' credit card points and the amount of notebook. The two operations relate to database and indeed make a "write-operation", no doubt, this need the transaction to guarantee the reliability. If Mr. Black just wants to

browse his account this moment, there are only some "read-operations" on database, this won't change the values in database and result in the no-consistence phenomenon, the high-level middleware protecting mechanism is not necessary, only a simple security can satisfy the requirement, at last can lower the developing cost and enhance the system processing efficiency. What's more, if we can separate the mainframe applications logic from web server applications, which means the mainframe applications concentrate on data processing, on the other hand the web server application only focus on business logical and parse the request (Analyzer—request arbitrator) decide which transaction need high-level protecting mechanism (through transaction middleware) which transaction only need normal-level security (such as JDBC).

The last problem considered is how to make the system with the capabilities of extendable and updatable intelligence. Because BI web/application server and TM server are designed in components-based, plug-in supported deployment style. Some important components such as decision-maker or transaction Task schemer must be changeable and extendable. Therefore, the popularization of Reliable Intelligence in today's e-business environment needs a new solution.

This paper presents a detailed solution of a new Reliable Intelligence Architecture and can solve or lower the negative impacts which are mentioned above in this section. This will facilitate both the developers and consumers. In addition, we also can change the configuration (transaction scheme or the protected level) of the transaction middleware so that it can satisfy more changeable requirements easily. On the other hand, the Business intelligence is merged into a scalable, components-based application server, with the MVC (Module View Controller) designed pattern, we can effectively separate the business-logic from the presentation-logic, which guarantee better components organizing and code reusing. At last gives a demo experiment and makes a conclusion. The rest of this paper is organized as follows. Section 2 to Section 4 describes the most important module in our system. Section 5 presents an experiment to demonstrate our design. The related works and the paper conclusion are discussed in Section 6.

2. Intelligence implement: business intelligence (BI)

In order to implement a high-reliable and intelligence, we use a "*process-driven"* application environment. In this environment, all the business applications execute transactions supporting business processes, performing activities such as receiving customer orders, managing inventory, shipping products, and billing customers. Transaction data and/or events are captured and integrated in a data warehouse environment for reporting and analysis by "*actionable BI applications"* in Fig 1; How to deal with data warehouse data into useful and actionable business information is depicted in Fig 2.

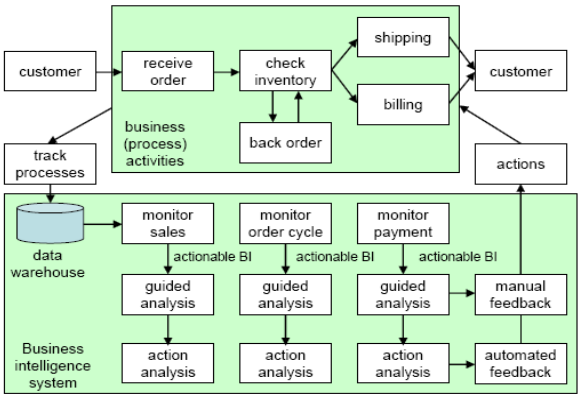

Fig 1: BI merged into web servers.

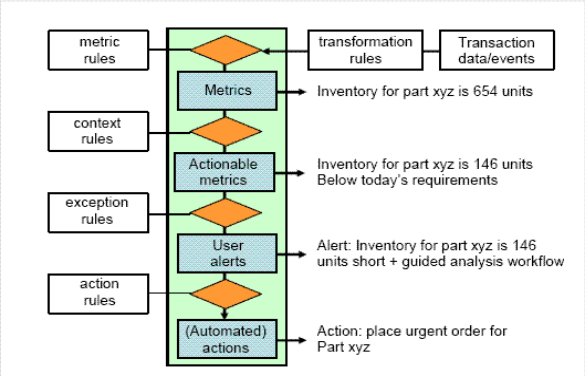

Fig 2: Tracing mechanism for useful and actionable business information.

NOTE: Here is just the simple inventory analysis flow. In fact, the intelligence analyzing algorithm for inventory or payment needs further discussion, now is just considered as a special component in our system.

The BI application interface in our intelligence system supports the "Plug-in mechanism". In this environment, the actionable BI applications can be seen as changeable and renewable components in our system. The interfaces for the three monitors dealing with sales, order cycle and payment are designed in a

standard regulation, they can be updated or modified as we want. The actionable BI decision-making logic can be generated in many ways, for example we can use business expertise and guided analysis to evaluate the actionable business information produced by BI systems to determine what decisions, if any, need to be made to optimize business operations and performance. Applying business expertise to business information creates business knowledge. This knowledge can then be fed back to the business processes that created the transaction data and business information being analyzed, and the business processes enhanced as appropriate. In some situations this feedback loop can be automated.

3. Resource recovery mechanism (RRM)

RRM is the security mechanism in our transaction middleware, which works as a coordinator to help transaction to achieve the commit or rollback of all the components, to guarantee the ACID (Atomicity, Consistency, Isolation, and Durability) properties, it is based on the two-phase commit protocol:

(1). When a transaction controller arrives at the end, it uses RRM to issue "*prepare" to commit* statement to the other transaction managers or data managers. They then do whatever is necessary to ensure that they can either make the changes permanent or roll them back. When this is done, they acknowledge the "*prepare*" to commit statement either positively or negatively. RRM collects all the "votes".

(2). If all the votes are positive, RRM asks all the resource managers to commit their changes. If any of the votes are negative, RRM asks all resource managers to roll back. At this point, RRM forgets about that "*prepare*" state and it is up to each resource manager to complete the commit or roll-back for their part.

4. Reliable implement: transaction middleware (TM)

First, we should realize that each user's interaction with a business information system involves one or more operations dealing with the data residing in database, here we must abstract all operations into transactions, For example, a customer sends a set of updating requests to TM, the updating operations will be seen as a atomic operations and encapsulated in a uniform mechanism shown in Fig 3. This means each transaction is a set of sub operations that must be executed as a unit (although each operation can run in a different process respectively). In TM, a group of related operations is called LUW (a logical unit of work). All the LUWs processed in TM ensure the integrity and consistency of business information systems by providing the ACID properties.

A transaction can comprise one or more LUWs, as shown in Fig 4. When an LUW completes successfully, it issues a "*synchronization point"* (also called a "*syncpoint"*), which marks the end of the LUW. This commits any data changes made in the LUW and releases the data for use by other transactions. If a task fails, any uncommitted changes are backed out automatically. This restores recoverable resources to the consistent state they were in at the beginning of the interrupted LUW (that is, at the most recent syncpoint or the start of the task). This reversal process, called "*dynamic transaction backout"*, occurs within the same task to safeguard other tasks from any chance of using corrupted data.

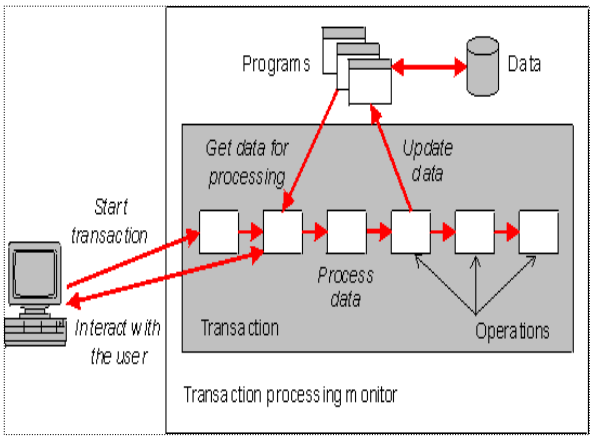

Fig 3: A transaction.

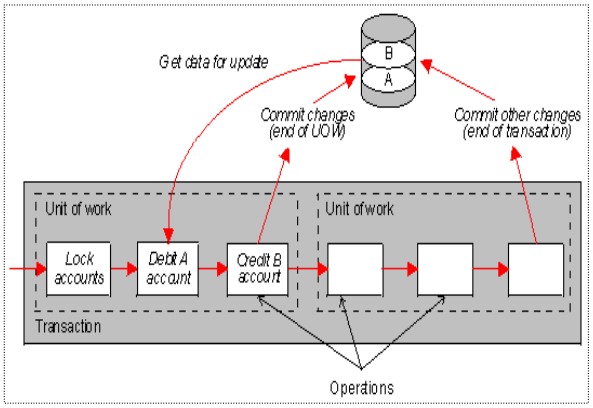

Fig 4: A transaction comprises LUWs.

When customers' requests are generated from applications, they will be considered as transactions and sent to TM server, then these transactions will be processed in a special order, the following numbered steps correspond to the numbers in Fig 5.

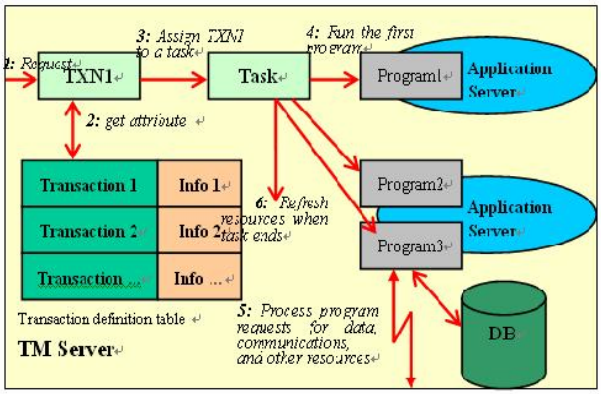

Fig 5: Transaction processing in TM server.

(1). The TM server receives a request from a user application. The TM server must first verify that it can communicate with the user's device and that the user is authorized to use the system. If the user has been verified, his request will be encapsulated into a transaction, the attributes (such as Task ID, privilege, resource require and which atomic operation is needed on database) for this transaction will be save in to the "Transaction definition table".

(2). The TM server scans the table of transaction definitions for information about the transaction. (Before a transaction can be used, it must be defined with attributes such as the name of the first program to be run when the transaction is requested.)

(3). If a transaction has been selected, the TM server assigns the request to a "*task*" that it uses to control the processing of the transaction's programs. The region schedules the task to be processed with other tasks and allocates processing time and access to the required data.

(4). The task runs the transaction's first program on a process called an "*application server*". If the transaction is implemented by several programs, those programs can run on the same or separate processes, depending on how the programs are invoked.

(5). The TM server monitors the progress of the task, serving its requests for data communications and other resources. It also performs background operations needed to ensure that the task continues to run optimally without conflicting with other tasks and with the data integrity required.

(6). When the task completes, the TM server commits any data changes, terminates the task, and frees resources for use by other transactions. If the task fails, the attributes and information for this task will not be deleted from the "Transaction definition table", and the "*retry*" will be processed when the required resources are available.

5. Experiment

This section will give a demo experiment based on the Implementation mentioned above.

5.1. Demo experiment instruction

Suppose there is an online-bank-service System. Now, Mr. Black wants to use his credit card to order many Notebook PCs and some other fittings (like mouse, modem) from X market. A few months earlier, Mr. Black has done the similar purchases 3 times, and then the history transaction information has been stored and processed in data warehouse by the BI components. When he login his account, the online BI components will realize the current customer is Mr. Black, and supply some services which is related to Notebook PCs ordering view because often ordering Notebook PC before. When the transaction between Mr. Black and the X market has finished, the points in Mr. Black's credit card will be move to the X market's financial account, the cash in both Mr. Black and X market's account will be changed at the same moment, this will be represented in the database of Bank.

Note: We suppose that both the personal account and X market account are served by same web server and TM server, the service of which also includes X market's financial and storage information;

5.2. Technical analysis

The Experiment contains both the "*read-operation*" and "*write-operation*" to the database of the Bank, now we make a further analysis about this demo experiment project.

Note: the analysis is based on the balance between the requirements and cost.

Transaction: Mr. Black orders some notebook PCs and other fittings (like mouse, modem) from X market.

The transaction is a typical online-trade business, customers can use online-listings to buy merchandise from X market. Now customers face a commerce net site, for this situation the dynamic web page is

necessary, because the dynamic web server page can contain some graphics and multimedia information to convenient the customers. With the help of the web page composed in and BI components, the advisable advertisements and utilities suiting to customer' requirements can be easily implemented. No doubt this can better attract customers for online-trading. Now, suppose the customer is Mr. Black, he wants to order many notebook PCs, besides the BI module processing on data warehouse, the commercial system will do two things during the online purchase that is change both the Mr. Black' credit card points and the amount of notebook. The two operations relate to database and indeed make a "*write-operation*", no doubt, this need to be combined into a transaction and use the TM server to guarantee the reliability, when he browses sale information and advertisements on the other hand, only "read-operation" on database happens, this won't result in inconsistent information happening.

Therefore, we classify two different way to access to the transaction service here:

(1). "*write-operation***":** During the transaction, Mr. Black account, X market's account and X market's storage information is served;

(2). "*read-operation***":** During the transaction, Mr. Black uses web browser to seek information from the commerce net site offered by X market;

The graphics and multimedia information such as advertisement will be stored in our BI application/web server with no relations to TM server. The more important, an "Analyzer (request arbitrator)" exists in the server, which can recognize which request is "write-operation" and which one is "read-operation". The "Analyzer" can also be considered as a BI application component with a standard interface, support for plug-in mechanism.

5.3. Implement the experiment

In order to obtain the high-volume processing capability, customers' transactions are processed in IBM mainframe, firstly there are some related introductions. The integrated system architecture is depicted in Fig 6.

(1).TM server: "*IBM CICSs (Customer Information Control System) server V3.1"*, which can run on IBM Z-series mainframe and offer some system API. In CICS, we can define transaction type, and coding the business processing logic.

(2).Web server: "*Websphere Application server V5.1***, which supports plug-in"**, CICS ECI adapter.

(3).Adapter: "*ECI resource adapter"***,** which is the connector, can change the process to transaction.

Deployment as below:

- **Importing the resource adapter and setting up the server**

To use the generated stateless session bean (deploy code) or the generated deploy code with a SOAP proxy, we must create and setup the server configuration so that we can connect to the CICS ECI resource adapter.

- **Creating a CICS ECI service**

When Application server has been configured, we need to create a CICS ECI service project on the server, such as Generate the enterprise service for the CICS program, import message definitions from the source program, generate deploy code for the enterprise service, bind the connection factory to the session bean, and finally generate the Server proxy and calling interface for the Servlet.

- **Coding the Servlet and JSP and Application**

In the Application-Server, containing most of the business logic module, this is Controller in MVC design pattern. The "*BI*", "*Analyzer*" detailed above is also included.

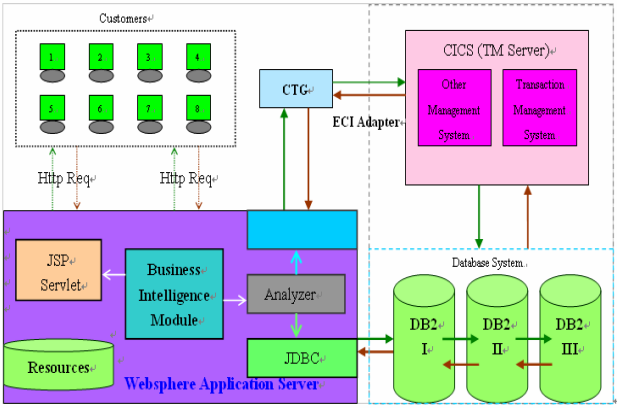

Fig 6: Integrated system architecture.

CICS Application's interface	CICS Proxy calling interface

LINKAGE SECTION.
01 DFHCOMMAREA.
10 GR-GN PIC X (10).
10 GR-GMID PIC X (10).
10 GR-GMNAME PIC X (10).
10 GR-GMPRICE PIC S9 (10).
10 GR-GMREMAIN PIC S9 (10).
10 GR-ERR-MSG PIC X (64).
	public Purchaseinfo()
	{
	addElement("gr__gn", java.lang.String.class);
	addElement("gr__gmid", java.lang.String.class);
	addElement ("gr__gmname", java.lang.String.class);
	addElement("gr__gmprice", java.lang.Math.class);
	addElement("gr__gmremain", java.lang.Math.class);
	addElement("gr__err__msg", java.lang.String.class); }

Table1:The ECI Interface between CICSProxy and JavaBean.

In Class Puichaseinfo, besides the attributes of the object, there are setter() and getter() methods are available, when a customer request has been sent to our system and been detect, we can use setter method to specify all the members' value in the class or use getter method to get information. Finally, we call the special method which is encapsulated in CICS Proxy to invoke the application in CICS server.

```
Purchaseinfo handler = new Purchaseinfo ();
handler.setGr__gmid(xxxx); ……..
handler.setGr__gmname(xxxx);
handler.setGr__gmprice(xxxx);
handler.setGr__gmremain(xxxx);
handler.setGr__gmtransid(xxxx);
……//set the value
……//
StationInfoCICSProxy
proxy = new StationInfoCICSProxy ();
handler_result = proxy.doPurchase(handler);
//Through CICS ECI to invoke CICS transaction.
```

6. Conclusions and future work

In this paper, we have discussed a new solution for High-volume Reliable Intelligence in modern E-Business, and introduced the security protecting mechanism design and implement. Then make a further analysis which is based on the business intelligence and balance between the requirements and cost, finally, give a demo experiment. The demo experiment can work regularly. However, there is a lot need to be improved, such as the requirement analyzer and BI components, this may related to Artificial Intelligence. In fact, the modem e-business is far more complex than our experiment. The optimization about the balance between the requirements and cost, the determination about which level of protecting mechanism should been used, and how to parse the request data package is more effective should be make a further discussion.

Acknowledgement

The project is supported by the IBM Technology Center of UESRC. We would like to thank Mr. Renuka Chekkala and Josef Klitsch for helping me in CICS debugging server and using the Websphere Application Server instead of other more complex and difficult method. The IBM Technology Center of UESTC for provides us with experiment environment, Mr. Liudi and Xuyi for help configuring and tuning DB2 CICS in mainframe.

References

[1] H.E. Huet, F. Vrije, Caromel and D. Bal, A High Performance Java Middleware with a Real Application. *Supercomputing Proceedings of the ACM/IEEE SC2004 Conference,* 2004.

[2] L. Bettini, R.D. Nicola, A Java Middleware for Guaranteeing Privacy of Distributed Tuple Spaces. *Lecture Notes In Computer Science,* 2604, 2003.

[3] M. Karlsson, K.E. Moore, E. Hagersten and David A. Wood, Memory System Behavior of Java-Based Middleware. *Proceedings of the 9th International Symposium on High-Performance Computer Architecture,* pp. 217, Washington, DC, USA , 2003.

[4] N. Williams, R. Herman, L.A. Lopez. M. Ebbers, Connecting the CICS to the Services Bus. *Implementing CICS Web Services IBM RedBooks*, 12: 150 - 170, 2006.

[5] N. Williams, R. Herman, Creating an enterprise service for a CICS with CICS ECI Adapter. *IBM Websphere manual*, 2005.

[6] C. Rayns, T. Delmenico Design the secure CICS SOA solution. *Security and CICS SOA access*, 12: 139-174, 2006.

[7] P. Havercan, T. Delmenico, Design the web support in CICS. *Security and CICS SOA access* 12: 175-255, 2006.

[8] C. Rayns, A. Bertoletti, L.Boyle, Distribute CICS Solutions. *The Next Generation of Distributed CICS* , 9: 41-171, 2006.

[9] M. Keen, C. Backhouse, J. Hollingsworth and S. Hurst. M. Pocock, CICS Web Decision making technologies. *Architecting Access to CICS within an SOA*, 10: 59-97, 2006.

[10] M. Keen, C. Backhouse, J. Hollingsworth and S. Hurst. M. Pocock. CICS access technologies. *Architecting Access to CICS within an SOA page* 10: 97-125, 2006.

[11] R. Credle, J. Adams, K. Clark and H. Jeter, et.al., Websphere Enterprise Service Bus. *SOA Design using WebSphere Message Broker and WebSphere ESB*,1: 97-120, 2007.

[12] F. Kon, F. J. Ballesteros, M. D. Mickunas and K. Nahrstedt, 2K: A Distributed Operating System for Dynamic Heterogeneous Environments. *In 9th IEEE Int. Symposium on High Performance Distributed Computing.* Pittsburgh USA, 2000.