Database Programming using Middleware Technologies

Done at

P- 923

Satyam Computer Services Pvt. Ltd.

Chennai

PROJECT REPORT

Submitted in partial fulfillment for award of Degree of

M.Sc.[Software Engineering]

Done by

Ganesh.S

9837S0046

Guided by

Miss.Rajathi

&

Mr.Krishna Kumaraswamy

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

(Affiliated to Bharathiar University)

COIMBATORE – 641006

KUMARAGURU COLLEGE OF TECHNOLOGY

(Affiliated to Bharathiar University)

COIMBATORE – 641006

BONAFIDE CERTIFICATE

This is to certify that this is the Bonafide Project Record Work done by S.Ganesh, Reg.No 9837S0046 in Partial fulfillment for the award of Degree of M.Sc. [SOFTWARE ENGINEERING], during the academic year 2002 –2003.

S. Thangaraj 29/3/03

Prof & HOD

Guide

Certified that the candidate was examined by us in the project work viva voce examination held on 4-4-2003 and the university register number was 9837S0046.

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

Any work requires the wholehearted contribution of not only the team members, but also of the people who guide and support the work. In this regard, on the successful completion of my project, I would like to express my heart-felt gratitude to a few people.

To begin with I express my deep sense of gratitude to our principal, **Dr. K.K. Padmanabhan, B.Sc (Engg), M.Tech, Ph.D**, for having provided the necessary support to carry out this project successfully.

I would like to convey my sincere thanks to our head of the Department **Computer Science and Engineering, Dr. N.Thangaswmay, Ph.D.** for his excellent motivation and guidance.

I am deeply indebted to our course co-ordinator Mrs S.Devaki , Asst Professor CSE Department for her invaluable support and guidance throughout the project.

I would like to take this opportunity to thank **Mr.Krishna Kumaraswamy Practice Head**, Satyam Computer Services and Mr.Karthikeyan Ramiah Project Leader, Satyam Computer Services and all the staff members of Satyam Computer Services for their beloved support and encouragement.

It is my privilege to thank my internal guide **Miss.Rajathi Senior Lecturer**, Kumaraguru College of Technology for her wonderful support throughout the project.

I thank all my friends and other fellow mates for their motivation and support.

Last but not the least I thank all the teaching and non-teaching Staff members of Dept of Computer Science and Engineering of our college.

PREFACE

The characteristics of a good document include clarity, brevity and simplicity. Thee document that you are currently going through is an attempt to make a document covering all aspects of good project-documentation.

The first phase of this document introduces you the synopsis of the entire project work. The reader of the document is assumed to have reasonable exposure to Client/Server technologies. However, the subsequent section which introduces thee problem and describes the tools has a detailed description of the middleware technologies adopted for this project work.

The next phase is the Software Requirement Specification document containing thee Task specification and modular level details. The expected product attributes are prescribed in this document.

The means to achieve those attributes are prescribed in the design document that follows the Software Requirement Specification Document. The database and Parameter specifications are also included as a part of a document. This is followed by the testing phase document that consists of a list of error scenarios and means to handle them.

The conclusion and references form the last phase of this document. The source code is not given as a part of the document due to internal restrictions. However interested readers can contact me to get a complete version of the source code. Suggestions and discussions on Client/Server computing are most welcome.

Ganesh.S
No 42, Kamaraj Avenue,
Adayar,
Chennai- 600 020

ganel43@yahoo.com

SYNOPSIS

The project titled "Database programming using Middleware Technologies" is a part of the complete Business Solution provided to a warehouse client in Japan. The objective of this project is to ensure the shipment of goods from one warehouse to another in a reliable and secure manner. There is centralized control and monitoring from the SAP Server.

The participating entities are the warehouses, the SAP Server, a Message Oriented Middleware, which is IBM MQ Series, a Local SQL Server, and Crossworlds. The project focuses only on the inbound data flow. The shipment operations are carried out using four fundamental messages explained using an example.

Suppose there are two warehouses DPWH1 & DPWH2.

If DPWH2 is running short of stock, DPWH1 may be requested to send its excess stock to DPWH2.This information is passed to the warehouses as

- Request to Ship for DPWH1.
- Request to Store for DPWH2.

The warehouses in turn sends back information

- Advice of Shipment from DPWH1
- Advice of Receipt from DPWH2

Request to Ship and Request to Store information comes from SAP and is outbound /with respect to SAP R/3.The data coming from SAP is passed into Crossworlds through MQ Series. The necessary transformation of data takes place in Crossworlds and it is again passed through MQ Series to the Communication Server.

The data from the MQ Series is parsed and updated into the SQL Server Database in the Communication Server. From the Communication Server the data is converted as a Text file and stored in the Data directory.

It should be noted that the SQL Server database merely acts as an intermediary storage area while the data flows from SAP R/3 to the warehouses.

A batch program, which is scheduled to run 8 times a day calls the VB exe and reads the text file from the Communication Server and transmits to the warehouses through VAN.

In the Inbound scenario data from warehouses come in the form of text file and is stored in the data drive. The scheduler calls the exe, which acts up on this data and updates it into the SQL Server database.

From the SQL Server database the data is send to the MQ Series in the format required by Crossworlds which then updates the SAP R/3 system.

MQ Series provides the role of a reliable MOM (Message Oriented Middleware) .It ensures delivery of messages across diverse mediums and geographical boundaries thereby facilitating message passing across different warehouses.

There is centralized control of stocks from the SAP Server. As mentioned above, this is a part of the complete Business solution provided to the vendor by Satyam Computer Services .

CONTENTS

INTRODUCTION

Current Status of the problem taken up

Concepts and Tools

- NT Service
- Messaging Concepts and IBM MQSeries
- Visual Basic, Visual C++ and Microsoft SQL Server.

1.Overview

An organization wishes to pass information to its various warehouses as to move stocks from one warehouse to another.

Eg:

Suppose there are two warehouses DPWH1 & DPWH2.

If DPWH2 is running short of stock, DPWH1 may be requested to send its excess stock to DPWH2.This information is passed to the warehouses as

- Request to Ship for DPWH1.
- Request to Store for DPWH2.

The warehouses in turn sends back information

- Advice of Shipment from DPWH1
- Advice of Receipt from DPWH2

Outbound Data Flow

Request to Ship and Request to Store information comes from SAP R/3 and is outbound /with respect to SAP R/3.The data coming from SAP is passed into Crossworlds through MQ Series. The necessary transformation of data takes place in Crossworlds and it is again passed through MQ Series to the Communication Server (TKRS8).

The data from the MQ Series is parsed and updated into the SQL Server Database in the Communication Server. From the Communication Server the data is converted as a Text file and stored in the Data directory.

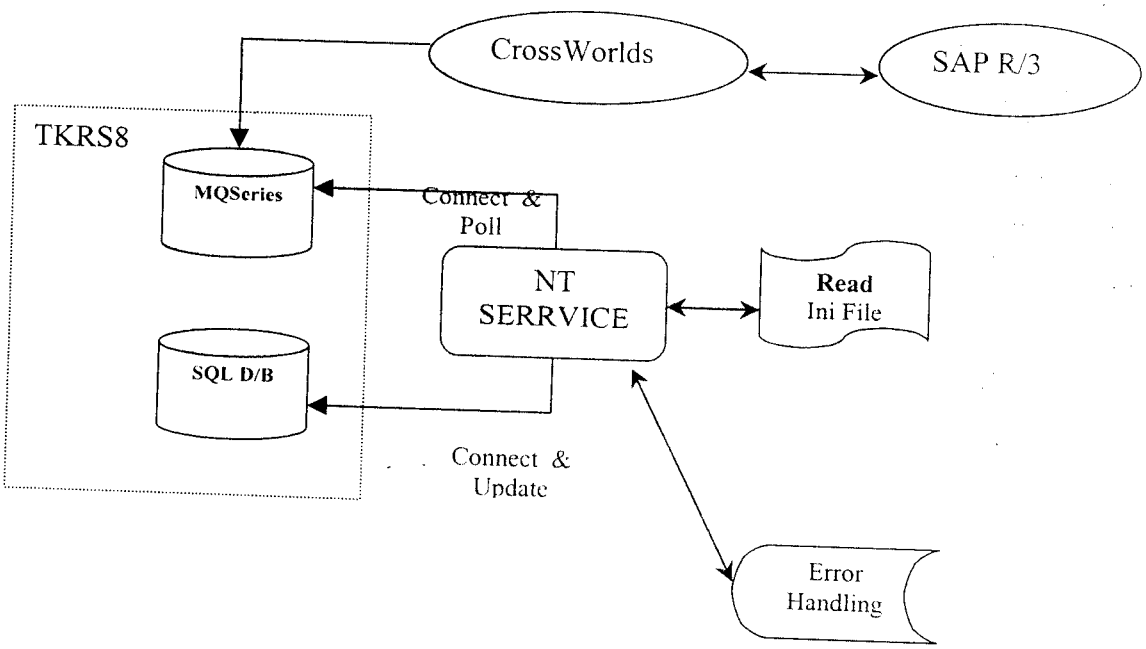
It should be noted that the SQL Server database merely acts as an intermediary storage area while the data flows from SAP R/3 to the warehouses.

A batch program, which is scheduled to run 8 times a day calls the VB exe and reads the text file from the Communication Server and transmits to the warehouses through VAN.

Inbound Data Flow

In the Inbound scenario data from warehouses come in the form of text file and is stored in the data drive. The scheduler calls the exe, which acts up on this data and updates it into the SQL Server database.

From the SQL Server database the data is send to the MQ Series in the format required by Crossworlds which then updates the SAP R/3 system.


CrossWorlds
SAP R/3
TKRS8
MQSeries
Connect &
Poll
NT
SERRVICE
Read
Ini File
SQL D/B
Connect &
Update
Error
Handling

1) Hence there is a service routine, which keeps polling the database and the queue and updates the database if there is any message in the queue and updates the queue if there is any message in the database.

2) There is a **Message oriented Middleware** that communicates from the internal world to the external world and vice versa.

3) There is a routine that maintains a local table in the warehouse and uses a text file to append it to the table.

4) There are error conditions ands other unexpected scenarios, a few of which can be handled locally.

Concepts And Tools

NT Services

The term "service" in Windows NT is used to denote both a special kind of Win32 process and Windows NT kernel-mode device drivers. In fact, a component of the operating system known as the "service controller" (or "service control manager," or SCM) is used to load and control both types of services.

Services have some special capabilities beyond those of the typical Win32 process. For one thing, you can tell NT to start your service when the system loads, before any users have logged on. That makes services a good choice for software that needs to start automatically and run constantly in the background, whenever the system is up. For example, if you want to write a program that continually monitors changes to files in a particular directory [using the `FindFirstChangeNotification()` function], you might want to implement that program as a service so that users cannot make unobserved changes to the file before your monitor program starts running.

Another important feature of services has to do with NT security. When you log onto NT, your account gives you certain privileges. Every program you execute has only those privileges, no more. Under UNIX, it's possible to create a privileged program that unprivileged users can execute, but you can't do that under NT (there's no "setuid bit" in NT).

Services, however, are executed by NT (not directly by you) and are associated at installation time with a particular account. In effect, a service logs on as a separate user, so although your account may not have the privileges needed to access some special file, you can ask NT to start up a service that does have the necessary privileges to do so (assuming a privileged user created such a service and gave you permission to use it). When you want to allow users to perform some privileged operation or access some special file in very restricted circumstances, consider writing a service. That way, the user gets to perform the privileged operation - but only via your program. [The new NT

function `LogonUser()` offers an alternative, but less secure, method of writing programs that acquire the privileges of some other account.]

Another handy feature of services is that they obey a common API for starting, pausing, stopping, etc. Once you write your service, any user can use the standard "Services" control panel applet to control it (assuming you have the necessary privileges). For instance, the user can pause a service and then continue it later. The configuration information (such as when and how a service starts) can be viewed and modified via the service controller. Services can even establish elaborate dependency lists on other services and control their load order among other services. What's even nicer is that much of the capability of the service controller is made available to applications in the form of a service controller API. These routines are implemented as an RPC server and are thus extensible to other machines (i.e., you can control services on other machines if you have sufficient privilege). The "Services" control panel applet uses these routines to allow users to view and configure all the services currently installed on a system. That's one reason why services are an attractive choice for the server side of client/server software; if you implement your server program as a service, all the work of letting client-side users administer your program (start it up, check if it's running, and so on) is already done for you.

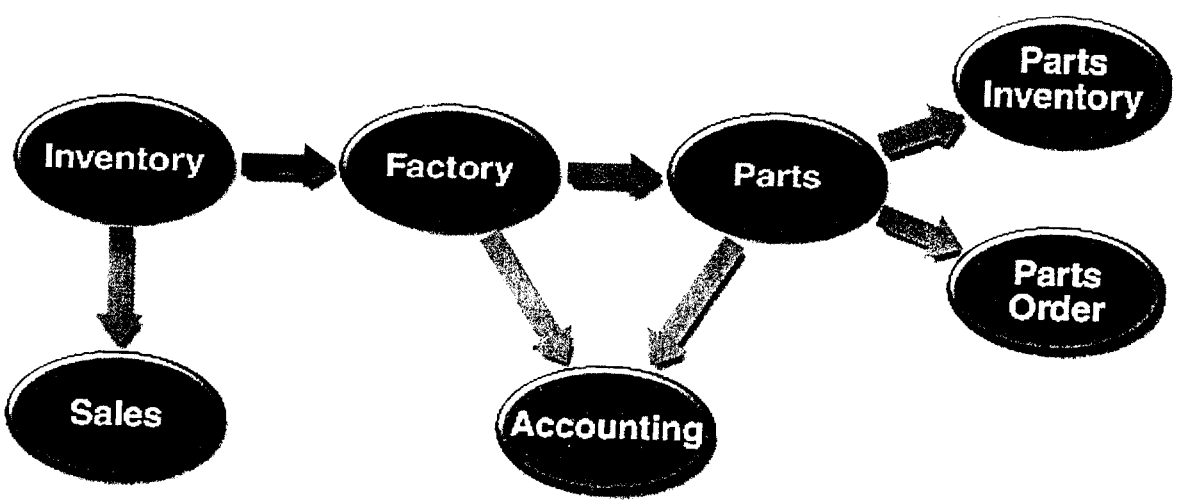
Services have many uses. Background processes that used to be implemented as DOS TSRs or perhaps UNIX daemons are good candidates for Windows NT services. Other examples of processes that are well suited as services include RPC servers, communications programs, utility programs such as virus scanners, and backup programs. The Windows NT operating system itself uses services extensively. The EventLog service is a classic example of a Win32 service. It is implemented as a service within the `services.exe` process (which contains several other services) and uses RPC, so that the event log routines it exports are available to remote machines. Of course, the EventLog service must be available before logon so that it can receive events that occur while the system is booting. The network subsystem also makes extensive use of services to manage network communication.

Messaging

Messaging is a method of communication between software components or applications. A messaging system is a peer-to-peer facility: A messaging client can send messages to, and receive messages from, any other client. Each client connects to a messaging agent that provides facilities for creating, sending, receiving, and reading messages.

Messaging enables distributed communication that is *loosely coupled*. A component sends a message to a destination, and the recipient can retrieve the message from the destination. However, the sender and the receiver do not have to be available at the same time in order to communicate. In fact, the sender does not need to know anything about the receiver; nor does the receiver need to know anything about the sender. The sender and the receiver need to know only what message format and what destination to use.

Messaging also differs from electronic mail (e-mail), which is a method of communication between people or between software applications and people. Messaging is used for communication between software applications or software components



Assuming that you are a vendor who has all these offices at different proximities extending over geographical boundaries.

To write code that communicates over all these boundaries and exchange information one would require :

1) A set of API's that work across multiple platforms.
2) A set of API"S that work on the TCP and UDP layer for reliable data transmission over networks.
3) Frequent error checking routines.
4) Ultimately a code that involves huge cost.

Message oriented middleware is a concept that helps overcome the above difficulties by playing the role of what is called a middleware.

Middleware:

Many definitions exist, many of them are more or less correct. The problem is that the word "Middleware" is somewhat overused and as a result has been applied to just about every piece of systems software ever written. Here's one definition:

"Middleware is a general term for any piece of software that serves to 'glue together', mediate between, or enhance separate existing programs."

As you can see, even that definition covers a huge range of applications. Here's another definition:

"Middleware is software that is used to move data from one program to another, shielding the developer from dependencies on communications protocols, operating systems and hardware platforms."

For a while, attempts have been made to categorise the different types of middleware. It's not always clear what category a product fits in to as vendors are always introducing new functionality, or include support for more than one category of middleware within a product. BEA's Tuxedo for example, includes transaction processing and message queuing services via the product's API (Application Programming Interface).

Having said that, middleware products tend to fall in to one of the following categories:

Middleware Category	Product Example
Message Oriented Middleware (MOM)	IBM MQSeries, Microsoft MSMQ, BEA MessageQ, Talarian SmartSockets Momentum's X-IPC, Level 8's Falcon MQ (a Unix p of MSMQ)
Data Connectivity	ODBC, Vendor specific products
Remote Procedure Calls (RPC's)	The OSF Distributed Computing Environment (DCE), Sun Microsystems NFS (Network File System)
Object Request Brokers (ORB's)	Traditionally associated with the CORBA (Common Object Reque Broker Architecture) standard. Iona's Orbix, Visigenic's Visibrol BEA Objectbroker, Java RMI (Remote Method Invocation)
Transaction Monitors	Microsoft Transaction Server (MTS), IBM CICS, IBM Encina, BEA Tuxedo.

The important thing to note is that each type of middleware has been designed to accomplish a specific task. The choice of middleware therefore depends on the business requirement that is being satisfied by deploying the technology. Many organisations employ more than one middleware technology. There is no "one size fits all" solution.

Synchronous versus Asynchronous Communication

Whilst there are different types of middleware as detailed above, they can all support one, or sometimes two, modes of operation. The modes are: synchronous or time dependent; and asynchronous, or time independent. Put simply, in a synchronous environment, at least two parties have to be present at the same time in order for communication to take place, rather like a telephone call. In an asynchronous environment, only one party has to be present, rather like an email. Synchronous and asynchronous modes are sometimes referred to as "connection oriented" and "connectionless" modes respectively.

Just to put this into context, it's necessary to jump ahead a little. MOM works primarily in an asynchronous (connectionless) mode, which means that for an application to send a message, the consumer of that message does not *have* to be available. Similarly when the consumer eventually reads the message (at an indeterminate point in the future) the producer of the message does not have to be available. Of course the designer of an application can "force" a pseudo-synchronous mode by insisting that the consumer sends an acknowledgement message back to the producer. The producer will then wait for the acknowledgement message before continuing processing. There's a fuller description of MOM in the next section.

The following table shows the various combinations of modes of operation. Transaction Monitors are almost always synchronous, although as mentioned earlier, products like Tuxedo bend the rules a bit! Also note again that MOM products, whilst asynchronous in nature, can be *made* to work in a pseudo-synchronous environment.

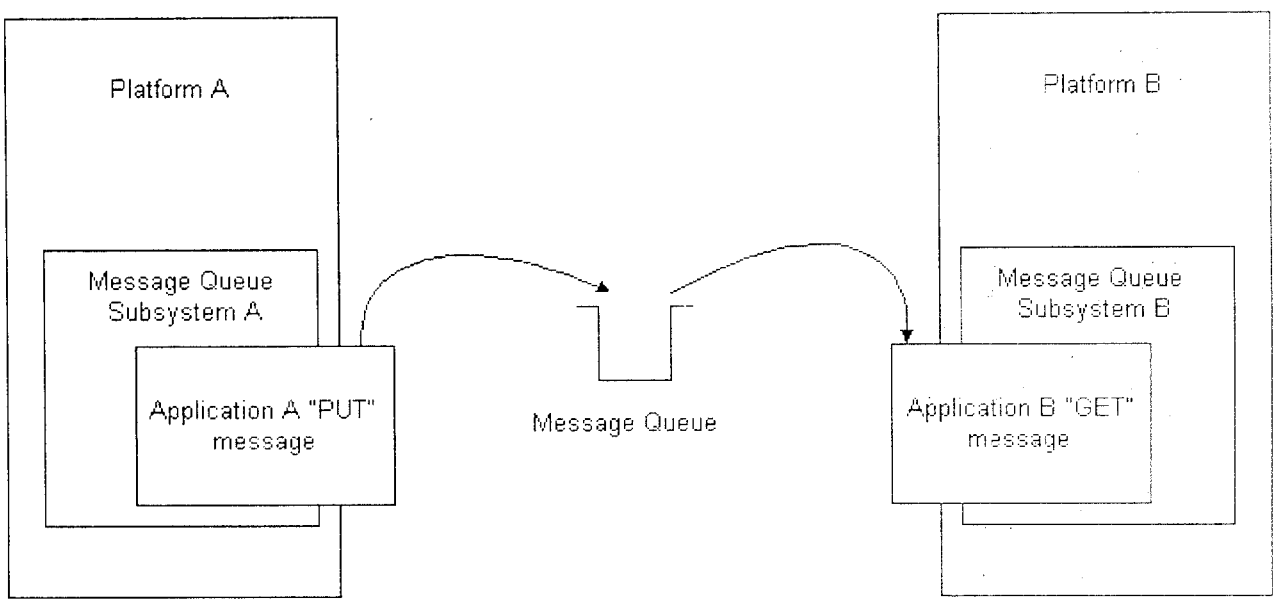
Middleware Category	Synchronous	Asynchronous
MOM		X
Data Connectivity	X	
RPC	X	

ORB	X	
Transaction Monitors	X	

As the main aim of this document is to describe the features and benefits of MQSeries, there will be no further discussion of other types of middleware.

What is Message Oriented Middleware?

It's already been suggested that MOM is middleware that allows an application to send a message to another application without the other application necessarily being available. That's it in a nutshell, however a picture showing the relevant components may give a better perspective.



This picture shows the logical world from the application's point of view. It's just one example of a common messaging scenario. Messages are placed on a queue by an application and retrieved by another application. The implication in this diagram is that the physical location of the queue is not known to either application. Similarly, the physical details of the host platform are not known. All that is required is that an application is in some way registered or connected to the message queue subsystem. The "cleverness" is contained in

that subsystem and therefore shields the applications from the physical world. (This matches closely with one of the definitions of middleware suggested at the start of this document) It also provides a useful abstraction that enables physical implementations to be changed on either platform, without affecting the rest of the implementation.

Why Do I Need Middleware?

There are benefits to utilizing middleware technology that are realized on many different levels. It depends who you are within an organization. Bearing in mind what we know so far about middleware in general, and MOM in particular, lets try to imagine how different people in an organization would see those benefits.

The second and third parts of the document will then concentrate on how MQSeries and JMS work in particular.

7

Board Level Director

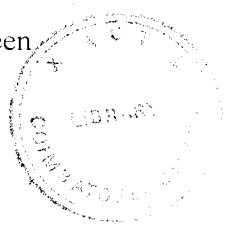
Your primary interests are business-related. You probably have significant investment in existing systems. Those systems provide business-critical functions, and will probably continue to do so for the foreseeable future. Unfortunately your competitors are all embracing the Internet, and have managed to "expose" their systems to the Internet and are now transacting business over it. Understandably, you want to do the same. Your IT manager informs you that this is possible, using the right technology. However, you are more interested in what is possible (and at what cost) rather than how it is achieved.

IT Manager

The IT Manager has a difficult job these days. Never before have organisations had so many different pieces of hardware and software that were never designed to work together. The vision of a single, unified set of software and hardware looks further away from reality than ever before. Now the demand to produce working systems from within a heterogeneous environment places the emphasis on re-using what you've got, rather than rewriting from scratch each time a new system is designed. The Board are increasingly nervous about

spending money on new systems when the business requirements are already being serviced through existing applications that have taken many years to evolve.

There is hope though. Your team of Architects tell you that middleware technology can be used to promote re-use by enabling the definition of interfaces between systems, or between individual business processes across systems. And you don't have to throw things away.



Technical Architect

The message from senior management these days is "re-use". This makes good sense to you. There are sound architectural reasons for pursuing software re-use. The main reasons are that re-using bits of systems reduces overall complexity, delivery times and project costs.

To achieve a decent level of re-use, you should pursue component based development (CBD). CDB relies *heavily* on middleware technology. How do you do CBD? Well, CBD is a subject unto itself, but here's a fly-by.

- Identify components. In order for applications to be re-used, the useful bits of them have to be identified and turned into components. A component is something that services a business request, like "Get me this customers account history", or "send this customer a reminder letter", and so on. This component identification, which will be influenced by the level of re-use you want to achieve, is the main challenge for the Architect.

- Define and publish the component interfaces. A component-based architecture relies upon each component having a known interface. This interface is then useable by any other "consumer" (which may in turn be another component). In this way systems can be assembled from components rather than being duplicated.

- Use the interface, the whole interface, and nothing but the interface. That is, a user (consumer) of a component service only has to know how to invoke a component through it's interface. The implementation details of that component are never known to the consumer. The corollary of this is that you can change a component's

implementation *without* having to change any of the consumer applications. That saves time and money.

The whole concept of CDB is very sound, but it hinges on the fact that components need interface to each other in a platform-independent way. That where the choice of middleware(s) come in, a represents a very useful tool in the architectural armoury. A component can be now be used by any other system in the enterprise.

Applications Developer

Architects have big ideas, but they seldom have to turn them into reality. That's the job of the Developer. Now to build your system, your Systems Analyst tells you that you need data from a legacy database that lives on a platform you know nothing about, on a network running a protocol you've never heard of. Normally, this would be a big problem. However, a member of the legacy system team in question has already "middleware enabled" the parts of that application that provide that vital data. You can thank the Architect for spotting that, and for the choice of (in this case) message oriented middleware. All you have to do is ask for the data. To do this, you write some code that puts a message on a queue. Then you wait for a reply (or have some other process that handles replies). That's it. You don't care how it works, it just does. And you can carry on building your system without duplicating the effort that went in to producing the legacy system in the first place. In short, you really are re-using software components. If you are curious and you want to see a map of all the components you can use for other things, speak to your Architect.

MQSeries Message Oriented Middleware

Background

Now that it is clearer where MOM fits in to the overall middleware picture, the remainder of this document will concentrate on IBM's MQSeries (MQ) specifically. MQ Provides a multi-platform, robust, assured delivery application to application messaging infrastructure. It comes with a vast array of functionality but can be extended further by the addition of user-written "exits". An exit is a piece of code that is executed by the queue manager upon certain events. Those events include; the placing of a message on a queue, the starting of a

communications channel, the exchange of security information with a remote queue manager. Many exits are provided as add-on "SupportPacs". IBM's MQSeries SupportPac web site URL is given in the "links" table at the end of this document.

MQ forms the basis of a family of products that include:

- MQSeries Integrator – a message broker with data transformation and mapping functions
- MQSeries Workflow - Model-driven e-business process automation and tracking
- MQSeries Everyplace – Allow mobile workers with laptops, phones and PDA's to participate in MQ networks.

Platform Support

MQ runs on every hardware platform and O/S that IBM produces, as well as NT, HP/UX, Solaris, Linux, Tandem NSK and more. IBM maintain a full list on their web site.

Network protocol support includes TCP/IP, IPX and SNA (LU6.2). This means that MQ applications can exchange data over disparate networks. Building on the idea of abstraction again, it's worth restating that the configuration of network details is all managed by the administrator of the MQ network and is contained within MQ, *not* within the applications that want to participate in the MQ network.

MQSeries Objects

In order to understand how MQ works, it is necessary to appreciate some of the objects that go to make up a working MQ installation. The word "object" in this context is not used in an Object Oriented sense. It merely refers to the "things" that have to be set up after the product is installed. After a brief description of each object, a picture will bring together all of the objects based upon the earlier conceptual diagram.

Queue Manager

The queue manager represents a physical instance of the messaging subsystem on a server. For an application to use MQ, it must first connect to a queue manager. The queue manager is created on the command line using the "crtmqm" command and is given a name by it's creator. During the creation process, a number of default objects are created. These default objects are used by MQ for it's own purposes during the life of the queue manager instance, as well as "model" objects than can be used as templates for creating further objects later on.

Once a queue manager is created, it can be started via the "strmqm" command, and stopped using the "endmqm" command. It is possible to run more than one queue manager on the same machine, although this often causes confusion when testing applications. Once it is started, administrative commands can be issued using the "runmqsc" command. MQ employs a scripting language to describe other objects that can be manipulated. The following objects are all created using runmqsc script.

Queue

The most frequently used object type. A queue can be defined using a rich set of attributes, far too many to describe within the scope of this document. Suffice to say queue attributes include message length, maximum queue depth, and default message persistence (decides if a message on a queue can survive a queue manager restarting)

Queues can be defined as *local* in which case they reside on the local machine; or as *remote*, in which case they exist on a remote machine with the definition of the remote queue "pointing" to the actual location on another queue manager. If an application puts a message on a remote queue, it is immediately placed on a "transmit queue" ready for transmission over a channel to a remote queue manager.

Finally, an application may create a queue dynamically. Typically this is used to accommodate replies to request messages, and once the reply message is read, the queue is closed and destroyed.

Channel

In order for messages to be transmitted to remote queues managed by remote queue managers, a channel between two queue managers must be defined. This usually involves some scripting on both queue managers although it is also possible to dynamically create a sender channel on request.

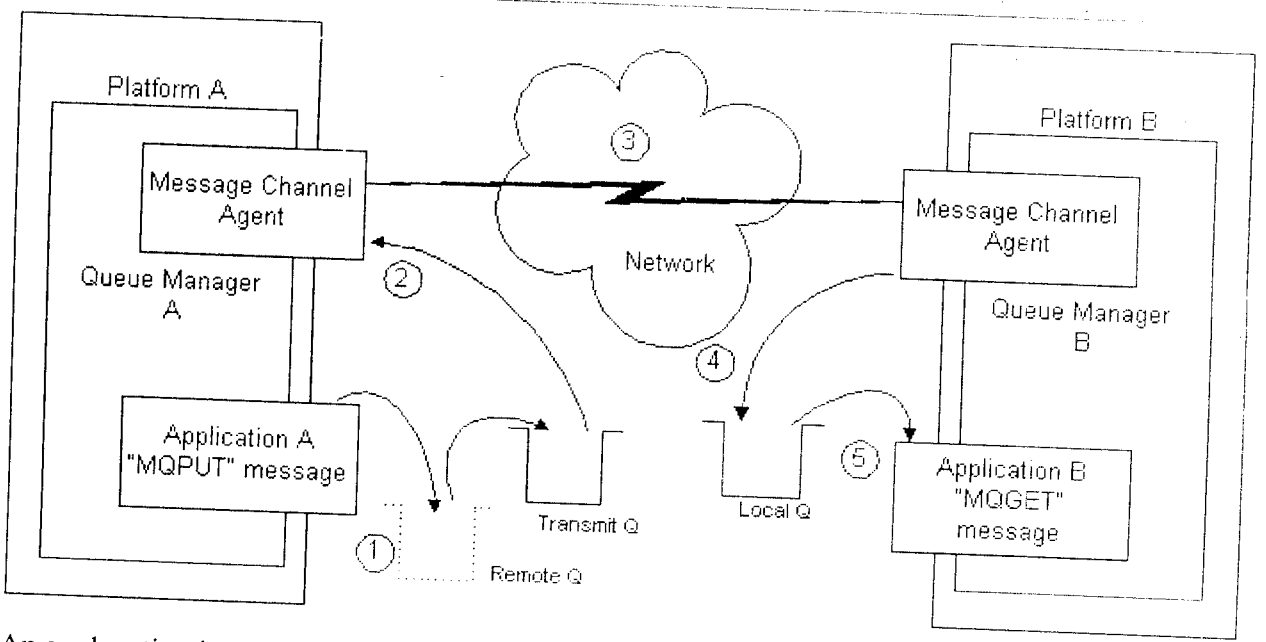
Channels are unidirectional. In other words a channel can be used for sending or receiving, but not both. It's therefore common to have a pair of channels defined between two queue managers – one for sending and one for receiving.

It may be the case that a remote queue exists on a queue manager that is more than one "hop" away. It is possible to configure remote queues so that they are routed through any number of intermediate queue managers.

Here's an example of how this would be useful. A queue manager on an SNA network manages a particular queue. Another queue manager on a TCP/IP network has a remote queue definition that points to the SNA queue manager, however because of the different network protocols in place, there is no direct route between the two networks. Enter a third machine, running SNA *and* TCP/IP, and is therefore visible to both networks. This third queue manager has two pairs of channel definitions, one pair to the TCP/IP queue manager, and the other pair to the SNA queue manager. The third queue manager can now act as a gateway for messages between the SNA and TCP/IP queue managers.

Again, all of this is in the configuration of each queue manager, not within any application that uses MQ.

Here's a picture based upon the earlier conceptual view, fleshed out with some MQ physical detail.



An explanation is appropriate. As you can see each step in the process is numbered.

(1) An MQ application, connected to queue manager A, puts a message on a queue. It so happens that this queue is not actually held locally, but does another queue manager, queue manager B, on a different box, manage a definition of a remote queue. The message is therefore immediately placed on a transmit queue.

(2) A component of the queue manager known as the Message Channel Agent (MCA) gets the message from the transmit queue. Note that the MCA is actually an MQ application, with built-in communications functionality. The MCA is controlled by the channel definitions given by the MQ administrator and represents the physical instance of a started channel.

(3) The two MCA's negotiate with each other, and the MQ "channel" is started. Channels can be configured to be open all the time, or to start when a message arrives on a transmit queue (as in this example).

(4) The receiving MCA inspects the message and determines where to place it. At this point, if the message details indicate that the message is destined for a different queue manager, it would be placed on another transmit queue, and another channel to that queue manager would be negotiated.

(5) The message is now available on the local queue, ready for application B to process.

Client Connections

In the example given above, it is assumed that the application physically resides on the same machine as the queue manager. This does not have to be the case, as applications can be configured to run as MQ clients instead. In this case the application is not coded any differently than before. The queue manager and client do not have to be the same platform type either. For example it is common to see a queue manager running on a Unix server, with client applications running on Windows 98 PC's.

When the client application makes a connection to the queue manager, a special kind of channel, a client channel, is negotiated. This channel stays active for as long as the client application remains connected to the queue manager.

Support for Different Messaging Models

Datagram (fire & forget)

A datagram message is a message placed on a queue, for which no reply is expected.

Request/Reply

An application places a message on a queue and expects a reply in return. This does not necessarily imply synchronous operation, as the process receiving the reply may be separate from the requester.

Publish & Subscribe

In this model messages are "broadcast" to interested participating applications in the MQ network. There are a number of elements within a publish & subscribe model.

1) The **publisher**. Publishers supply information about a number of topics. A topic can be anything the designer of the system wants, say stock market prices or seating arrangements for a theatre.

2) The **broker**. The publisher sends MQSeries messages containing topical data to a Publish & Subscribe broker (installed on a queue manager), which then forwards it to other brokers in the network.

3) The **subscriber** registers an interest in a number of topics. Brokers then send MQSeries messages to subscribers containing data on the subscribers registered topics. It is possible for many publishers to publish the same topic (for example, publishers for both NASDAQ and London Stock Exchange prices) and subscribers can register interest in any number of topics.

The publish and subscribe model can be quite sophisticated. Writing pub/sub applications can therefore be complicated, so IBM have provided another API, the AMI (Application Messaging Interface) for C, C++ and Java. This greatly simplifies the creation of publishing and subscribing applications. Much of the configuration of the pub/sub environment is held in an external repository which is referenced by the participating applications.

Queue Manager Clusters

Queue managers can be logically grouped together to form clusters. Each queue manager may be on a different platform type (NT, AIX, Solaris) and may be physically remote from each other. Cluster queue managers host cluster queues, which are advertised to other queue managers in the cluster. It is possible to have the same queue logically shared by more than one queue manager, thus providing a level of high availability. There is also a benefit in terms of reduced systems administration, as clustered queues do not have to have a definition within each queue manager in the cluster.

Queue manager clusters can be quite sophisticated, and individual clusters may logically "overlap" one another. A repository of cluster information is held on at least two queue managers within the cluster.

For a detailed description, have a look at the IBM MQSeries clustering documentation.

XA Compliance

The Open Group's XA interface standardises the relationship between a *transaction manager* and a *resource manager*. This allows applications to group operations into a single unit of

work that will be committed only when all operations have completed successfully. Any failure will result in *all* of the operations in the unit of work being backed-out. This enables applications to maintain a guaranteed level of integrity in the event of an application failure.

An MQSeries queue manager is XA compliant. This means it can act as a *resource manager* in a distributed unit of work managed by an external *transaction manager* such as Encina or Tuxedo. This gives the application designer the freedom to use whatever tools are appropriate for the task. Other resource managers include Oracle, Informix, Sybase, DB/2 and SQL Server.

For example, it is possible for an application to have a unit of work that does an SQL INSERT, then put a message on a queue. If any of the statements fail, the messages will be rolled back off the queue, along with the database change, thus ensuring the "all or nothing" integrity of the unit of work. The changes are only committed when all operations have been successful. This kind of distributed transaction processing is often vital in enterprise level systems.

Web Development

MQ is becoming increasingly popular as a web-enabling technology. Perhaps the simplest technique is to use the MQSeries Internet Gateway. This is essentially an MQ enabled CGI program. It can handle HTML "POST" requests, forwarding form data to a queue. A reply can be processed, with the appropriate HTML response being sent back to the client browser. Although simple, the Internet Gateway product provides a straightforward way of accessing existing applications via the web – a hot topic for many organisations at the moment.

More sophisticated applications can be developed with Java. Java is supported in a number of different ways. IBM describe this support as follows:

- MQSeries classes for Java and MQSeries classes for Java Message Service - The MQSeries classes for Java allow a program written in the Java programming language to connect to MQSeries as an MQSeries client using TCP/IP, or directly to an MQSeries server using the Java Native Interface (JNI). They allow Java applets,

applications, and servlets access to the messaging and queuing services of MQSeries. If the client-style connection is used, no additional MQSeries code is required on the client machine. The MQSeries classes for Java enable a message-based approach to application integration using Java. MQSeries classes for Java Message Service is a set of Java classes that implement Sun Microsystems Java Message Service specification. A JMS application can use the classes to send MQSeries messages to either existing MQSeries or new JMS applications.

- The MQSeries Client for Java - an MQSeries client written in the Java programming language for communicating via TCP/IP. It enables Web browsers and Java applets to issue calls and queries to MQSeries giving access to legacy applications over the internet without the need for any other MQSeries code on the client machine.

Security

The "built in" security system within MQ is the Object Authority Manager (OAM). This system provides basic access control security to MQ objects, rather like Unix file and group permissions. With the OAM, it is possible to control which users or groups can connect to a queue manager, put or get messages to queues, or create objects dynamically. This system works well as far as access control is concerned, but cannot deliver the sophisticated message-level security that organisations now require.

Many organisations are looking towards PKI (Public Key Infrastructure) technologies to provide security services for their applications.

IBM have produced an MQ SupportPac that allows MQ to participate specifically within Entrust's PKI infrastructure. The SupportPac is C source code for a channel message exit that enables the use of the PKI API (via GSS-API function calls), therefore enabling security features such as queue manager authentication, digital signing of messages, and message encryption.

Since the source code uses the GSS-API, the source code should be portable to environments that provide security services through other systems such as the Massachusetts Institute of Technology's Kerberos system. A version of Kerberos is now being used (somewhat controversially) in Microsoft's Windows 2000 operating system.

As the web page for SupportPac MS0C says:

"The mechanism used by MQSeries to transfer messages between two adjacent queue managers, or between a client and a queue manager is called a channel. This SupportPac provides security services at the middleware level; more specifically, it integrates with the MQSeries channels as channel exit programs. These channel exit programs are based on the well-defined open standard security application interface (GSS-API). The GSS-API services are provided by the EntrustSession toolkit, which runs in the Entrust/PKI environment. Entrust is a company that provides Certificates and Key management services. More information about the company and its products can be found at Entrust Technologies."

This approach offers a convenient way to integrate MQ into a PKI. The MQ applications do not have to have any knowledge of the PKI environment, as the channel exit will be executed by the queue manager, outside of the control of the application.

The MQI (Message Queue Interface)

Background

The MQSeries MQI benefits from standardisation across all platforms. Although there are semantic differences between programming languages, the basic verbs and data structures are the same if you are programming in C on Solaris, or COBOL on OS/390. In addition, an OO interface is available for C++ and Java programmers.

The API is physically implemented as a set of run-time libraries linked in to the MQ application. The application then invokes each verb in a fashion appropriate to the programming language environment, for example a C function all, a class instance, or subroutine call.

A brief overview of each verb is given below. Verbs are grouped together to illustrate complementary functionality. The overview does not give the full prototype for each verb, just a description of the function performed by that verb.

Language Support

There is direct support for C, C++, Java, COBOL, Perl (via a SupportPac), and PL/I.

Verbs

MQCONN / MQCONNX / MQDISC

MQCONN connects the application to a named queue manager. If successful, a handle is returned that is passed to all subsequent MQ calls. MQCONNX is a variation of MQCONN, but it establishes a "trusted" connection to the queue manager. This means that the application bypasses an IPC layer that all MQI calls normally go through, and operations are processed more rapidly. The disadvantage of MQCONNX is that an errant MQ application can adversely affect the queue manager.

MQDISC disconnects an application from a queue manager.

MQOPEN / MQCLOSE

Open or close an MQSeries object. Usually associated with opening and closing queues.

MQPUT / MQGET

Get or put messages to a previously opened queue.

MQBEGIN/ MQCMIT / MQBACK

These are the MQSeries unit-of-work (syncpoint) co-ordination verbs. They can be used if the application requires MQSeries to control units of work. If an external transaction manager is being used then the application would use the calls appropriate for that environment. See the section on XA Compliance for more information.

MQBEGIN marks the start of a unit of work. This is a useful shorthand that means that every subsequent MQ operation will be processed under the same unit of work. The alternative to this is to include an option within each MQ operation that explicitly states that the operation is to be included under syncpoint.

MQCMIT commits a unit of work, so all MQPUT and MQGET operations are physically, rather than logically executed. MQBACK "rolls out" a unit of work, effectively undoing the last group of operations under syncpoint.

MQINQ / MQSET

MQINQ provides the application with a way to retrieve the attributes of an MQ object. They may be inspected, altered, and then applied to the object using the MQSET verb.

SQL Database Servers

In database-centric client/server architecture, a client application usually requests data and data-related services (such a sorting and filtering) from a database server. The database server, also known as the SQL engine, responds to the client's requests and

provides a secured access to shared data. The client application can, with a single SQL statement, retrieve and modify a set of server database records. The SQL database engine can filter the query result sets, resulting in considerable data communication savings.

An SQL server manages the control and execution of SQL commands. It provides the logical and physical views of the data and generated optimized access plans for executing the SQL commands. In addition, most SQL provides server administration features and utilities that help manage the data. A database server also maintains dynamic catalog tables that contain information about the SQL objects housed with it.

Visual C++

Visual C++ is the product that should be used if we want to produce programs for Windows.

One of the best reasons to use C++ at all is its flexibility. We have total control over out programming environment. Other languages tend to protect the programmer, which can be a very good thing when time is of the essence. Unfortunately, that protection can get in the way, and that's when you need something like C++ to cut through the red tape to get the job done.

VC++ is also the ideal programming environment for writing ActiveX controls, along with IIS-specific code like ISAPI (Internet Server Application Programming Interface) extensions and ISAPI features. Even if other application programming belongs to the RAD programming environments, no one wants to take the time to download a bloated control from the Internet or saddle their Web server with a slow filter. VC++ provides the small executable that we really want. In addition, the added flexibility that VC++ provides actually makes writing these kinds of application easier.

Longevity is another reason to use Visual C++. A developer needs great support today and tomorrow. Since every implementation of C++ has small differences, especially if we use vendor-specific bells and whistles, it really pays to go with a product we can depend on.

Another area where VC++ excels is database programming. It can provide the speed needed to get the information quickly without too much more programming than a RAD language would require.

Software Requirement Specification

Purpose

General Description

Product Functions

Specific Requirements

Design Constraints

Purpose

This document is directed towards the staff members of Kumaraguru College of Technology, Coimbatore for their reference. The Software Requirement Specification document is for the "News Interface " project in Satyam Computer Services.

Scope

a) The product is aimed at providing business solution for a warehouse company with regard to-
   1) Transactions
   2) Shipment of goods
   3) Storage of goods
   4) Advice of Receipt

b) The solution extends across Wide Area Network and provides platform independent communication service.

Overview of the Document

The Software Requirement Specification document addresses the requirements of the News Interface Project for company x. The document begins with an overview of the problem in hand. The various tasks to be accomplished are explained with a broad perspective.

The specificity of each module to be developed is elucidated in the functional requirements and constraints. Then follows the software requirements, Hardware requirements, user requirements and performance requirements.

The conclusion to this document brings a perspective of the final system in hand i.e. the business solution in stock exchange in Warehouses.

2.General Description

Overview:

A company X wishes to pass information to its various warehouses as to move stocks from one warehouse to another.

Suppose there are two warehouses DPWH1 & DPWH2.

If DPWH2 is running short of stock, DPWH1 may be requested to send its excess stock

to DPWH2.This information is passed to the warehouses as

- **Request to Ship for DPWH1.**
- **Request to Store for DPWH2.**

The warehouses in turn send back information

- **Advice of Shipment from DPWH1**
- **Advice of Receipt from DPWH2**

Product Functions

1.Outbound Data Flow

Request to Ship and Request to Store information comes from SAP R/3 and is outbound with respect to SAP R/3.The data coming from SAP is passed into Crossworlds through MQ Series. The necessary transformation of data takes place in Crossworlds and it is again passed through MQ Series to the Communication Server (TKRS8).

The data from the MQ Series is parsed and updated into the SQL Server Database in the Communication Server. From the Communication Server the data is converted as a Text file and stored in the Data directory.

It should be noted that the SQL Server database merely acts as an intermediary storage area while the data flows from SAP R/3 to the warehouses.

A batch program, which is scheduled to run 8 times a day calls the VB exe and reads the text file from the Communication Server and transmits to the warehouses through VAN.

2.Inbound Data Flow

In the Inbound scenario data from warehouses come in the form of text file and is stored in the data drive. The scheduler calls the exe, which acts up on this data and updates it into the SQL Server database.

From the SQL Server database the data is sent to the MQ Series in the format required by Crossworlds, which then updates the SAP R/3 system.

General Constraints

Database Connectivity

MQ Series blocked

SQL database blocked

Reliability of physical connections in the network

Specific Requirements

1.Modules to be developed

The modules to be developed are:

4.1.Outbound

1. Two NT Service developed in VC++ that polls for any incoming data in the MQ Series and updates the SQL Server database.

- One for Request to Ship.
- One for Request to Store.

2. Two executable files developed in VB, which reads any new data that is available in the SQL Server database and converts it into a text file.
   - One for converting Request to Ship into text file.
   - One for converting Request to Store into text file.

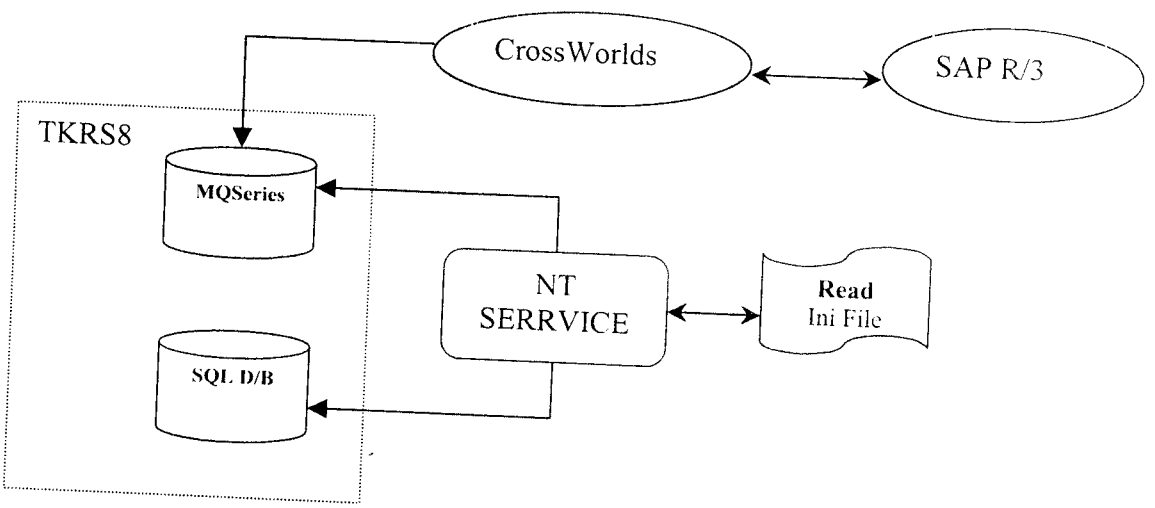
4.2.Inbound

1. Two executable files developed in VB, which reads any new data that is available in the data drive in the text file and updates into the SQL SERVER database.
   - One for converting Advice of Shipment into SQL Server Database
   - One for converting Advice of Receipt into SQL Server Database.

2. Two NT Service developed in VC++ that poll for any new data in the SQL server database and updates the MQ Series.
   - One for Advice of Shipment.
   - One for Advice of Receipt.

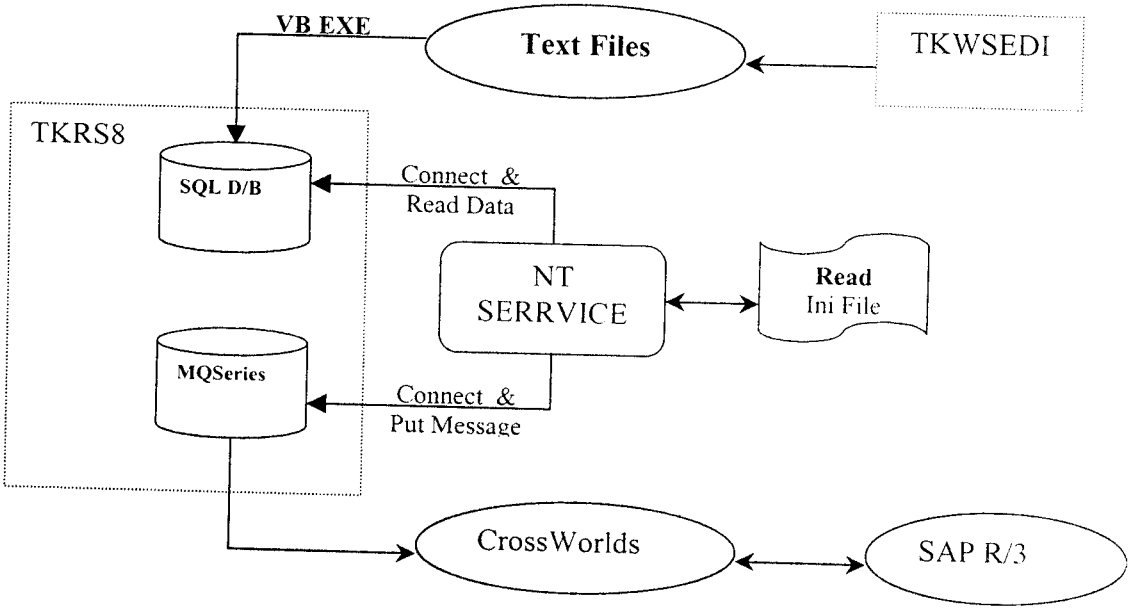
Task Specification Diagrams

8.1.NTService: Request to ship & Request to Store



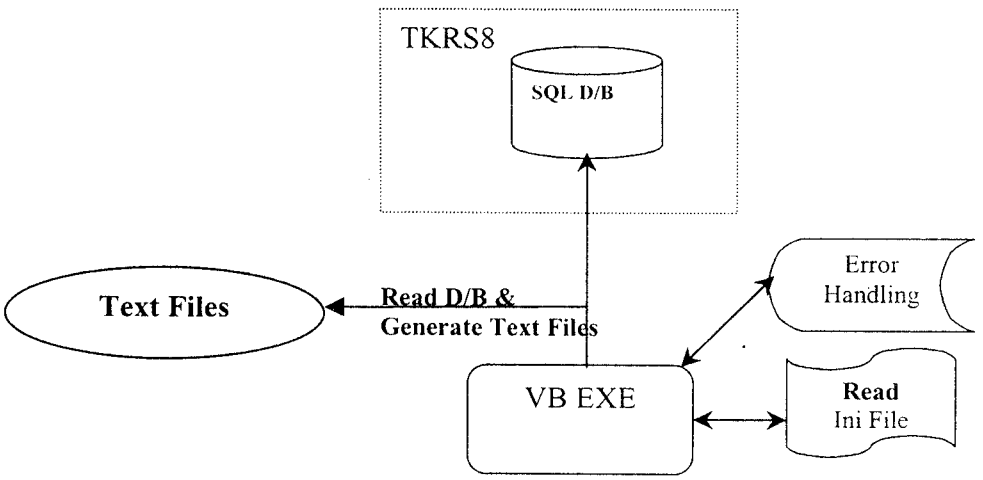
1. Once the REQSHIP & REQSTOR Service is started it reads the INI file.
2. Stores the INI values in temporary variables.
3. Connect with the Queue Manager.
4. Connect to the SQL Server Database.
5. The number of attempts to retry connection with the MQ or the SQL Database will be as specified in the INI file.
6. If after those number of retries either MQ or SQL is down the Service will stop and has to be restarted.
7. Keep polling the Queue and whenever a new message arrives, it parses and updates the respective tables in the database as specified in the Database section.
8. The errors encountered if any are written into the Error table.

NTService: Advice of Shipment & Advice Of Receipt functionality:



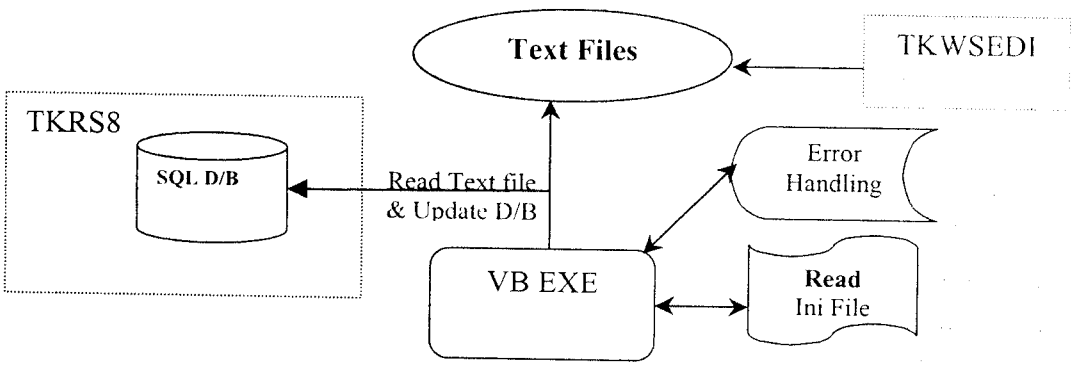
- Once the ADVSHIP & ADVRECT Service is started it reads the INI file.
- Stores the INI values in temporary variables.
- Connects with the Database
- Connect to the MQ Series
- The number of attempts to retry connection with the SQL and MQ will be as specified in the INI file.
- If after those number of retries either the MQ or SQL is down the service will stop and will have to be restarted.
- Keep polling the respective tables in the Database for any new records and whenever a new record arrives, it retrieves it, forms an MQ message and sends it across to the Crossworlds.
- The errors encountered if any are written into the Error table.

VB Exe: Request to Ship & Request to Store functionality



1. The VB Program will read the INI file.
2. Store the values into temporary values.
3. Connect to the SQL Server Database.
4. Read the necessary table, check for unread records and convert the read data into a Text file and stores the Text file in the specified directory and with the name specified in the INI file.

VB Exe: Advice of Receipt & Advice of Shipment functionality



1. The VB program will read the INI file.
2. Store the values into temporary values.
3. Connect to the SQL Server Database.
4. Reads the Text file and updates the necessary tables in the SQL Server database as a new transaction.

Design Constraints:

Hardware specification:

Processor: 1

Make: Intel

Configuration:

Frequency: 833 MHZ

Memory: 64 MB RAM

Hard disk: 10 GB each

Cache: 512 KB

Video memory: 32 MB Ram

Floppy: 3 1 /2 inch floppy drive

Human-ware specifications:

The project team consists of four members:

1) Mr. Krishna Kumaraswamy
2) Mr. Karthik.R
3) Mr. Senthil.B
4) Mr. Ganesh.S

The human-ware is trained in VC++, IBM MQ series and SQL database.

Software Specification

SQL Server

IBM MQ Series

Visual C++ and NT service utility

Visual Basic

Design

Design Overview

Structured Chart

Structured English

Data Flow Diagram

Detailed Design

- Database and Function parameters specification

Glossary of Terms

Design Overview

As mentioned in the Software Requirement Specification ,the modules to be developed are

Outbound

Two NT Service developed in VC++ that polls for any incoming data in the MQ Series and updates the SQL Server database.

- One for Request to Ship.
- One for Request to Store.

Two executable files developed in VB, which reads any new data that is available in the SQL Server database and converts it into a text file.

- One for converting Request to Ship into text file.
- One for converting Request to Store into text file.

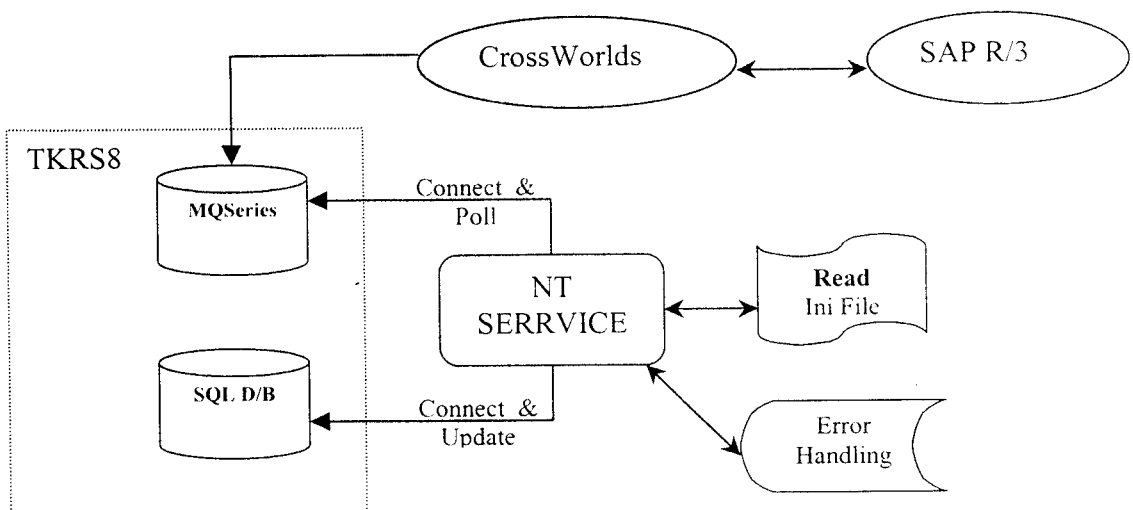
Inbound

1. Two executable files developed in VB, which reads any new data that is available in the data drive in the text file and updates into the SQL SERVER database.

- One for converting Advice of Shipment into SQL Server Database
- One for converting Advice of Receipt into SQL Server Database.

2. Two NT Service developed in VC++ that polls for any new data in the SQL server database and updates the MQ Series.
   - One for Advice of Shipment.
   - One for Advice of Receipt.

8.1.NTService: Request to ship & Request to Store



Structured English for Request to ship and Request To Store- NT Service

Task A:

Once the REQSHIP & REQSTOR Service is started it reads the INI file.

Task B:

Stores the INI values in temporary variables.

Task C

Connect with the Queue Manager.

Task D

Connect to the SQL Server Database.

Task E:

The number of attempts to retry connection with the MQ or the SQL Database will be as specified in the INI file.

Task F:

If after those number of retries either MQ or SQL is down the Service will stop and has to be restarted.

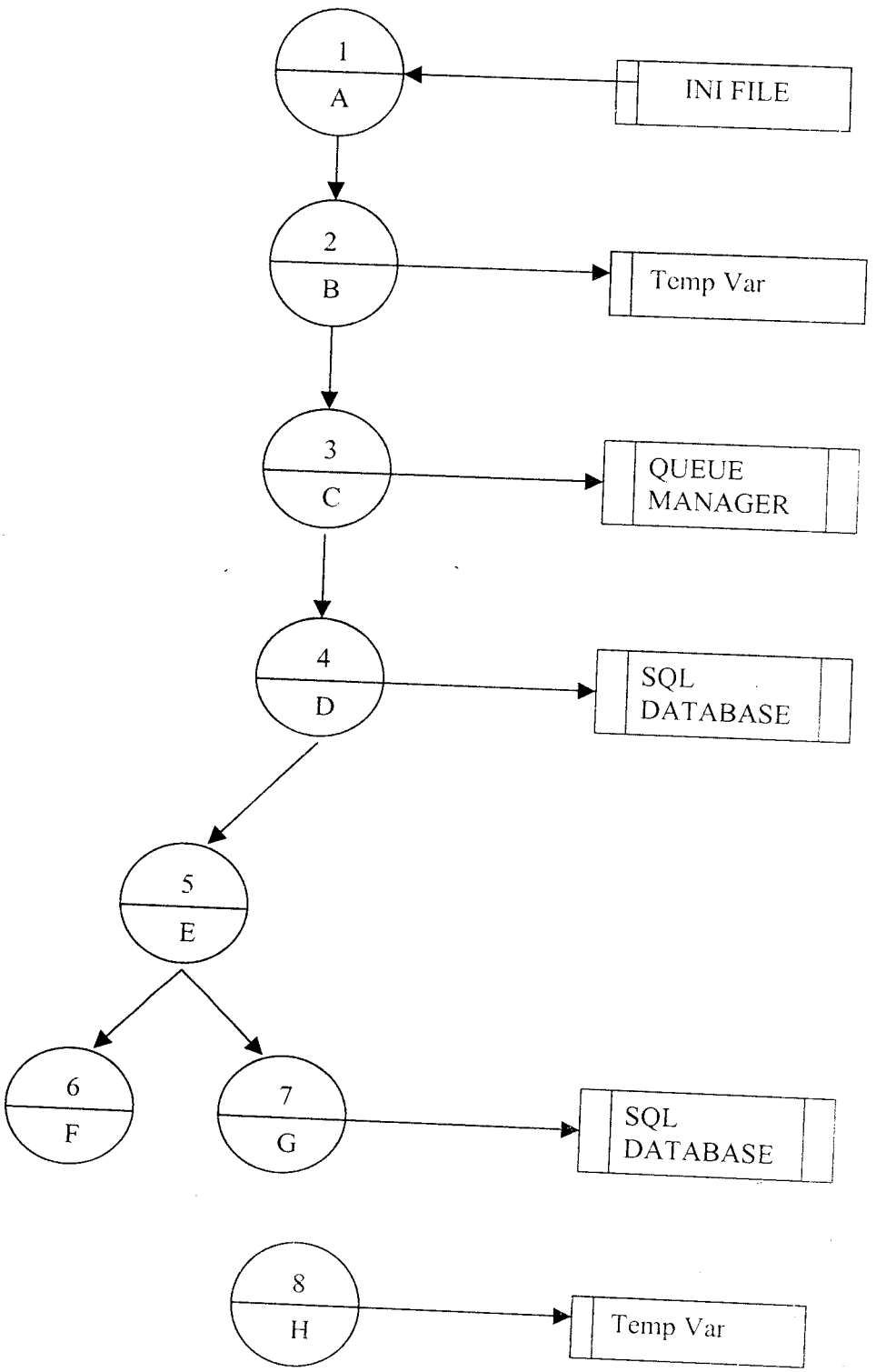
Task G:

Keep polling the Queue and whenever a new message arrives, it parses and updates the respective tables in the database as specified in the Database section.

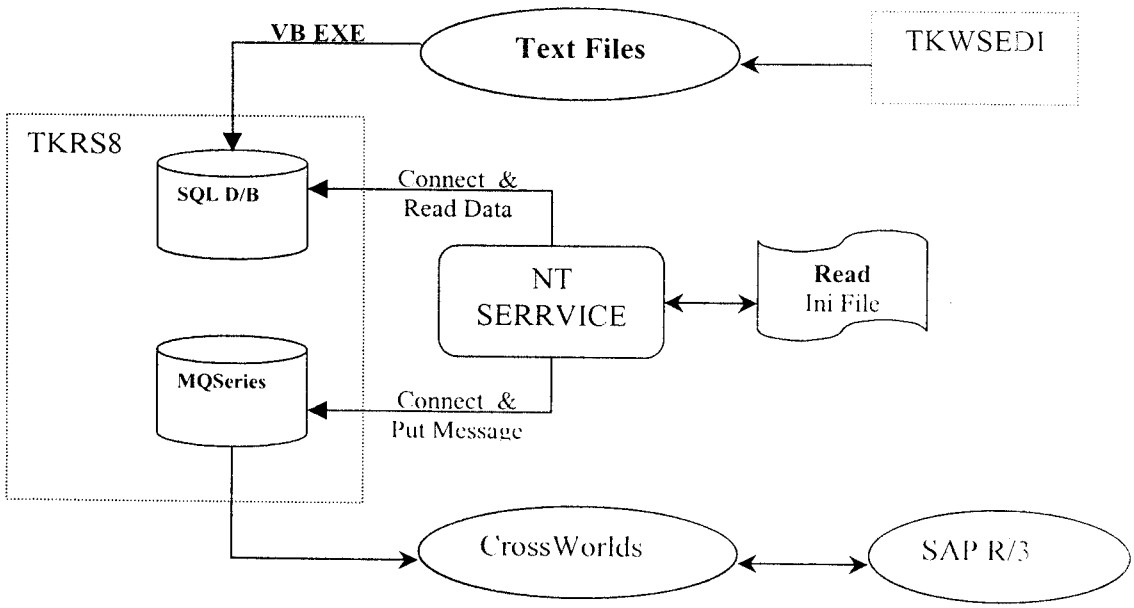
Task F :

The errors encountered if any are written into the Error table.

Data Flow Diagram



NTService: Advice of Shipment & Advice Of Receipt functionality:



Structured English for Advice of Shipment and Advice of Receipt- NT Service

Task A

Once the ADVSHP & ADVRECT Service is started it reads the INI file.

Task B

Stores the INI values in temporary variables.

Task C

Connect with the Database.

Task D

Connect to the Queue Manager.

Task E

The number of attempts to retry connection with the MQ or the SQL Database will be as specified in the INI file.

Task F

If after those number of retries either MQ or SQL is down the Service will stop and has to be restarted.

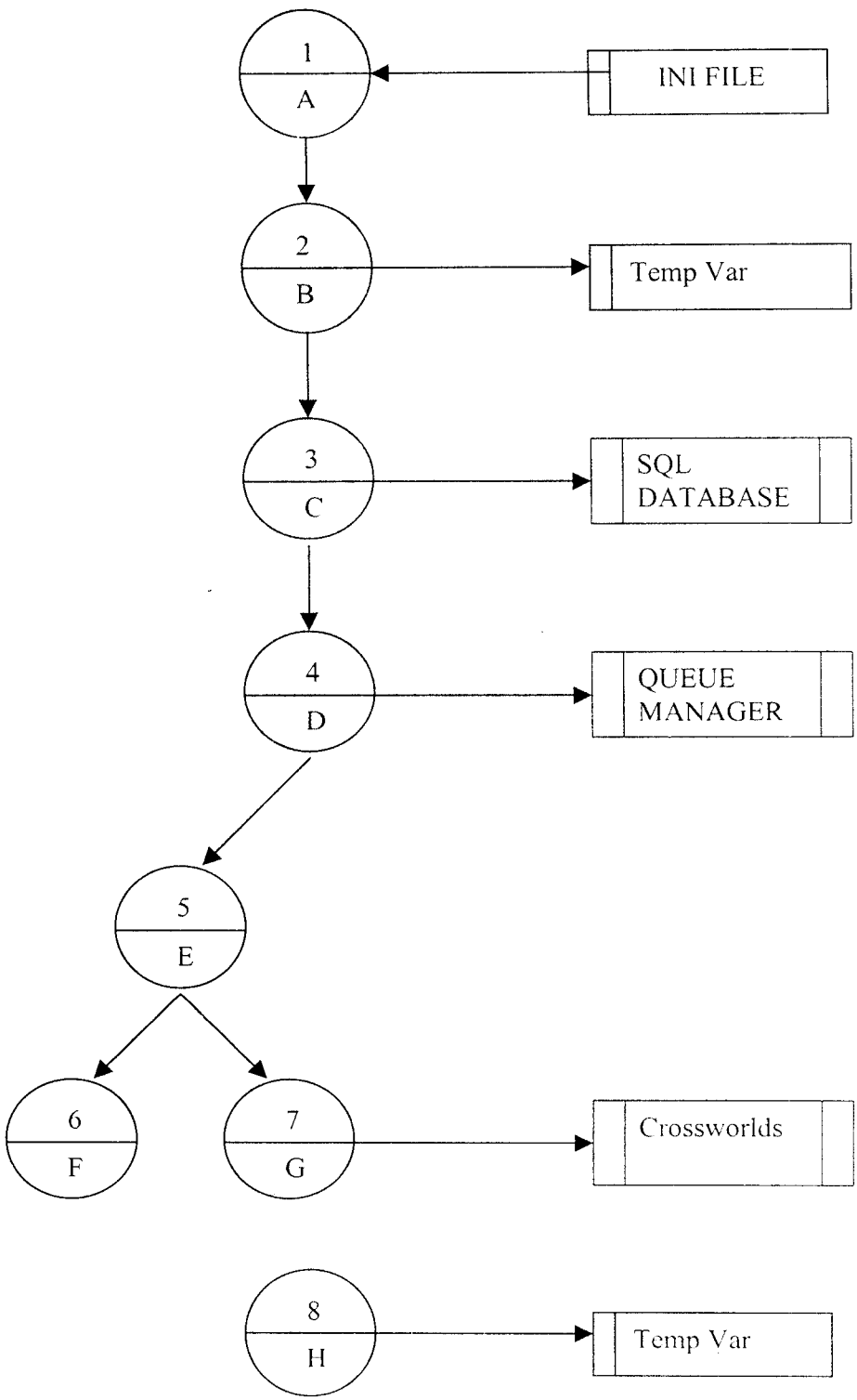
Task G

Keep polling the Database and whenever a new message arrives, it parses and sends it across to Crossworlds .

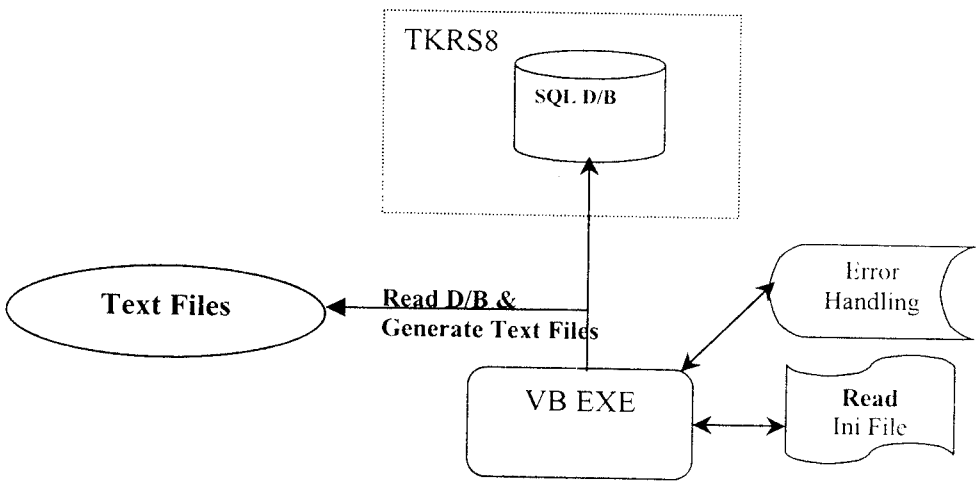
Task F

The errors encountered if any are written into the Error table.

Data Flow Diagram



VB Exe: Request to Ship & Request to Store functionality



Structured English for Request to ship and Request To Store- NT Service

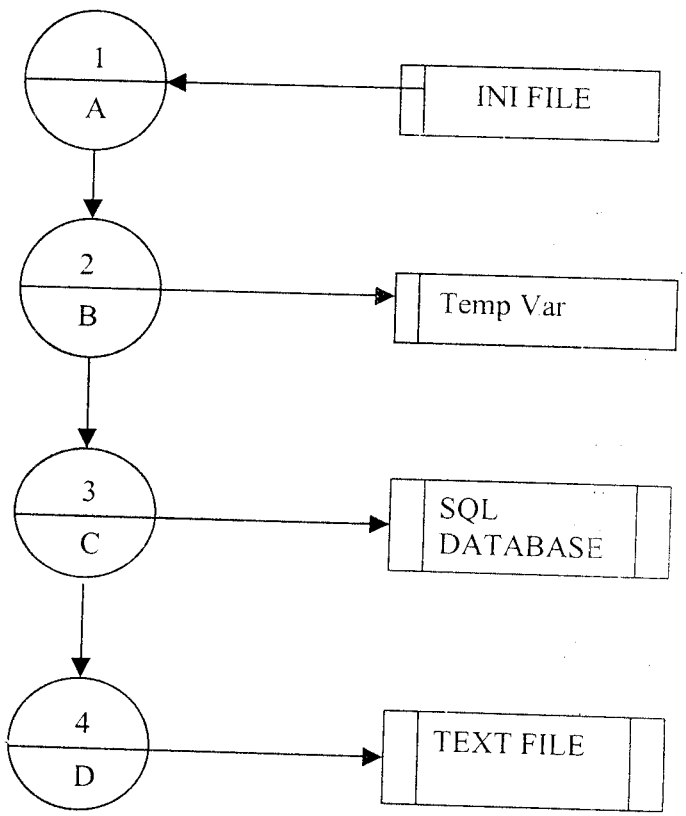
Task A
The VB Program will read the INI file.

Task B
Store the values into temporary values.

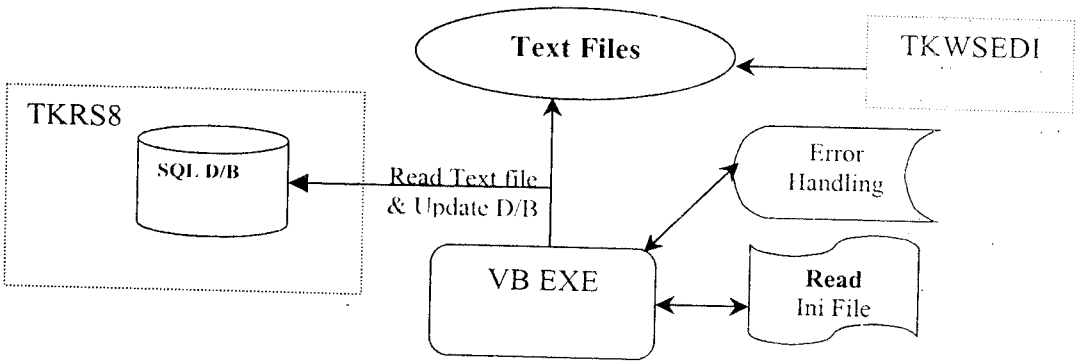
Task C
Connect to the SQL Server Database.

Task D
Read the necessary table, check for unread records and convert the read data into a Text file and stores the Text file in the specified directory and with the name specified in the INI file.

Data Flow Diagram



VB Exe: Advice of Receipt & Advice of Shipment functionality



Structured English for Advice of Shipment and Advice of Receipt- NT Service

Task A

The VB Program will read the INI file.

Task B

Store the values into temporary values.

Task C

Connect to the SQL Server Database.

Task D

Reads the Text file and updates the necessary tables in the SQL Server database as a new transaction.

Data Flow Diagram

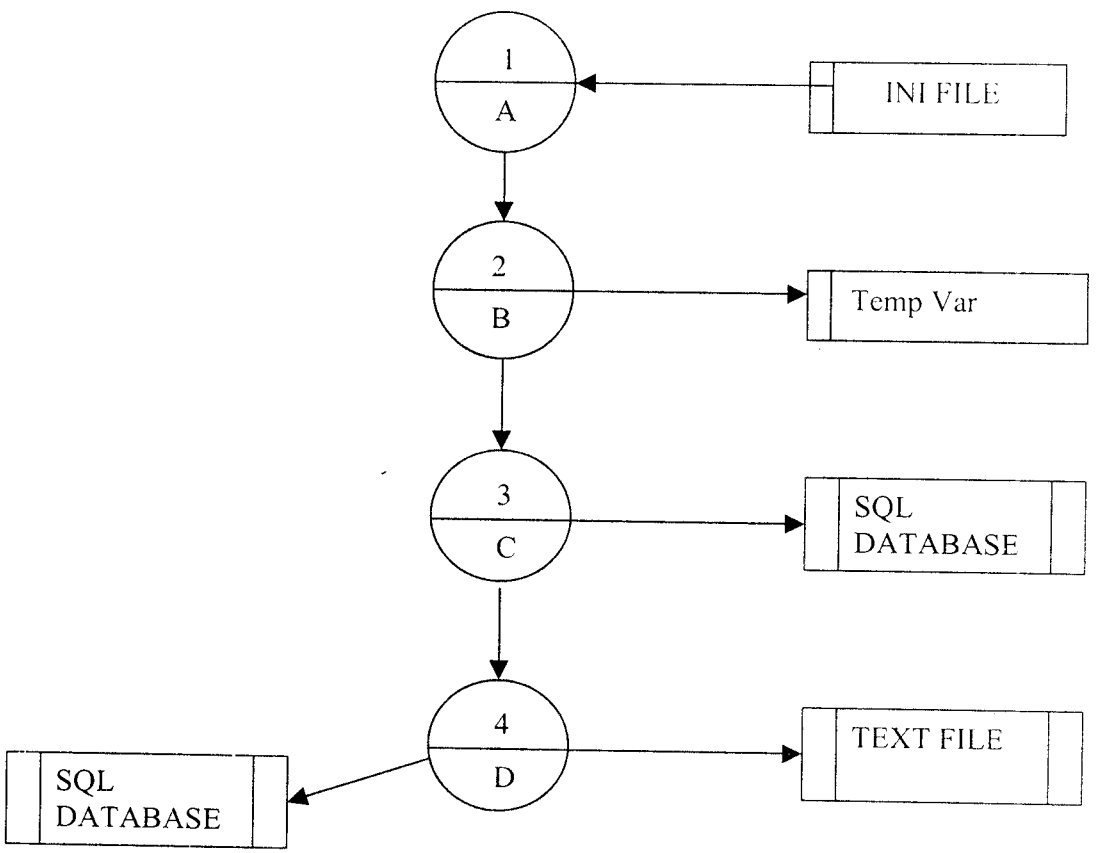


TABLE STRUCTURES

Tables involved in Request To Ship:

1. Table Name: RSP_H [Message Header]

S.No	Field Name	Name in Table	Field Type	Field Length	Null
1.	Created User Id	CRET_USER_ID	Varchar	8	N
2.	Created Time Stamp	CRET_DT_TM	SmallDateTime	10	N
3.	Altered User Id	ALT_USER_ID	Varchar	8	N
4.	Altered Time Stamp	ALT_DT_TM	SmallDateTime	10	N
5.	Sequence No	SEQ_NUM	Integer	10	N
6.	Record Type	RCD_TYP	Varchar	1	N
7.	Message ID Code	MSG_ID_CD	Varchar	8	N
8.	Filler-1	FILL_1	Varchar	1	Y
9.	Sender Company Code	SEND_CO_CD	Varchar	8	N
10.	Sender User-ID	SEND_USER_ID	Varchar	8	N
11.	Receiver Company Code	RCV_CO_CD	Varchar	8	N
12.	Receiver User-ID	RCV_USER_ID	Varchar	8	N
13.	Filler-2	FILL_2	Varchar	18	Y
14.	Creation Date	CRET_DT	Varchar	8(From MQ)	N
15.	Creation Time	CRET_TM	Varchar	4(From MQ)	N
16.	Filler-3	FILL_3	Varchar	1	Y
17.	Filler-4	FILL_4	Varchar	128	Y
18.	Updated Flag	UPD_FLAG	Char	1	N

Tables Involved In Request To Store

Table Name: RST_H [Message Header]

S.No	Field Name	Name in Table	Field Type	Field Length	Null
1.	Created User Id	CRET_USER_ID	Varchar	8	N
2.	Created Time Stamp	CRET_DT_TM	SmallDateTime	10	N
3.	Altered User Id	ALT_USER_ID	Varchar	8	N
4.	Altered Time Stamp	ALT_DT_TM	SmallDateTime	10	N
5.	Sequence No	SEQ_NUM	Integer	10	N
6.	Record Type	RCD_TYP	Varchar	1	N
7.	Message ID Code	MSG_ID_CD	Varchar	8	N
8.	Filler-1	FILL_1	Varchar	1	Y
9.	Sender Company Code	SEND_CO_CD	Varchar	8	N
10.	Sender User-ID	SEND_USER_ID	Varchar	8	N
11.	Receiver Company Code	RCVR_CO_CD	Varchar	8	N
12.	Receiver User-ID	RCVR_USER_ID	Varchar	8	N
13.	Filler-2	FILL_2	Varchar	18	Y
14.	Creation Date	CRET_DT	Varchar	8	N
15.	Creation Time	CRET_TM	Varchar	4	N
16.	Filler-3	FILL_3	Varchar	1	Y
17.	Filler-4	FILL_4	Varchar	161	Y
18.	Updated Flag	UPD_FLAG	Char	1	N

Tables Involved In Advice Of Shipment:

3.Table Name: AOS_H [Message Header]

S.No	Field Name	Name In Table	Field Type	Field Length	Null
1.	Created User Id	CRET_USER_ID	Varchar	8	Y
2.	Created Time Stamp	CRET_DT_TM	SmallDateTime	10	Y
3.	Altered User Id	ALT_USER_ID	Varchar	8	Y
4.	Altered Time Stamp	ALT_DT_TM	SmallDateTime	10	Y
5.	Sequence No	SEQ_NUM	Integer	10	Y
6.	Record Type	RCD_TYP	Varchar	1	Y
7.	Message ID Code	MSG_ID_CD	Varchar	8	Y
8.	Filler-1	FILL_1	Varchar	1	Y
9.	Sender Company Code	SEND_CO_CD	Varchar	8	Y
10.	Sender User-ID	SEND_USER_ID	Varchar	8	Y
11.	Receiver Company Code	RCVR_CO_CD	Varchar	8	Y
12.	Receiver User-ID	RCVR_USER_ID	Varchar	8	Y
13.	Filler-2	FILL_2	Varchar	18	Y
14.	Creation Date	CRET_DT	Varchar	8	Y
15.	Creation Time	CRET_TM	Varchar	4	Y
16.	Filler-3	FILL_3	Varchar	1	Y
17.	Filler-4	FILL_4	Varchar	14	Y
18.	Updated Flag	UPD_FLAG	Char	1	Y

4.Tables Involved In Advice Of Receipt

.Table Name: AOR_H [Message Header]

S.No	Field Name	Name In Table	Field Type	Field Length	Null
1.	Created User Id	CRET_USER_ID	Varchar	8	Y
2.	Created Time Stamp	CRET_DT_TM	SmallDateTime	10	Y
3.	Altered User Id	ALT_USER_ID	Varchar	8	Y
4.	Altered Time Stamp	ALT_DT_TM	SmallDateTime	10	Y
5.	Sequence No	SEQ_NUM	Integer	10	Y
6.	Record Type	RCD_TYP	Varchar	1	Y
7.	Message ID Code	MSG_ID_CD	Varchar	8	Y
8.	Filler-1	FILL_1	Varchar	1	Y
9.	Sender Company Code	SEND_CO_CD	Varchar	8	Y
10.	Sender User-ID	SEND_USER_ID	Varchar	8	Y
11.	Receiver Company Code	RCVR_CO_CD	Varchar	8	Y
12.	Receiver User-ID	RCVR_USER_ID	Varchar	8	Y
13.	Filler-2	FILL_2	Varchar	18	Y
14.	Creation Date	CRET_DT	Varchar	8	Y
15.	Creation Time	CRET_TM	Varchar	4	Y
16.	Filler-3	FILL_3	Varchar	5	Y
17.	Updated Flag	UPD_FLAG	Char	1	Y

Input and Output Parameters

1.Outbound:

Service - Request To Ship & Request To Store:

Input Parameters:

- **Queue Manager Name**
- **Queue Name**
- **Delimiter(No delimiters)**
- **Database Name**
- **Computer Name**
- **Printer Name**
- **DSN Name User Id & password**
- **Table Names**
- **Log File Name**

Possible Error Scenarios

1. **Unable to Connect with the Queue Manager.**
2. **Unable to get message from the Queue.**
3. **Unrecognizable format in the Queue.**
4. **Sequence in the Queue is not correct.**
5. **First Field should always be the Record type**

VBExe –Request to Ship & Request To Store

Input Parameters:

- **Database Name**
- **DSN Name,User Id & Password**
- **Table Names**
- **Text File Name**
- **Delimiter-Not required**
- **Data Directory**
- **Number of time it should retry to connect with the SQL Server before coming out – 5.**

Possible Error Scenarios

1. **Unable to Open Text File**
2. **Unable to Find Text File**
3. **Data Folder not found**

Inbound:

VBExe: Advice of Shipment & Advice of Receipt

Input Parameters:

- **Database Name**
- **Table Names**
- **DSN Name,User Id and Password**
- **Destination of Text File**

Possible Error Scenarios

1. **Unable to Open Text File**
2. **Unable to Find Text File**
3. **Data Folder not found**
4. **Unable to connect with the Database(5 retries before coming out)**

7.2.2.Service: Advice of Shipment & Advice Of Receipt

Input Parameters:

- **Queue Manager Name**
- **Queue Name**
- **Dead Letter Queue Name**
- **Database Name**
- **DSN Name User Id and password.**
- **Table Names**

Possible Error Scenarios

1. **Unable to Connect with the Queue Manager.**
2. **Unable to put message into the Queue.**
3. **Unable to connect with the Database (5 retries)**
4. **Check the sequence of the data.**
5. **Check the first field of each record.**

INI FILE

Name : News.INI

```
[COMMON]
DATABASE_NAME=NEWSDB
DATA_SOURCE_NAME=DSN=NEWSDB
UID=sa
PWD=
ERROR_TABLE=ERR_RPT
SEQUENCE_TABLE=SEQ_NUM
COMPUTERNAME=
PRINTERNAME=

;;REQUEST TO SHIP-NTSERVICE
[REQUESTTOSHIP]
QUEUE_MANAGER_NAME=
QUEUE_LOCAL_NAME=
LOGFILE_NAME=

;;REQUEST TO STORE-NTSERVICE
[REQUESTTOSTORE]
QUEUE_MANAGER_NAME=
QUEUE_LOCAL_NAME=
LOGFILE_NAME=

;;ADVICE OF SHIPMENT-NTSERVICE
[ADVICEOFSHIPMENT]
QUEUE_MANAGER_NAME=
QUEUE_LOCAL_NAME=
LOGFILE_NAME=

;;ADVICE OF RECEIPT-NTSERVICE
[ADVICEOFRECEIPT]
QUEUE_MANAGER_NAME=
QUEUE_LOCAL_NAME=
LOGFILE_NAME=

;;REQUEST TO SHIP –VREQSHIP EXE
[VREQSHIP]
FOLDER_NAME=
TEXT_FILE_NAME=
LOGFILE_NAME=
```

```
;;REQUEST TO STORE –VREQSTOR EXE
[VREQSTOR]
FOLDER_NAME=
TEXT_FILE_NAME=
LOGFILE_NAME=

;; ADVICE OF SHIPMENT –VADVSHIP EXE
[VADVSHIP]
FOLDER_NAME=
TEXT_FILE_NAME=
LOGFILE_NAME=

;; ADVICE OF RECEIPT –VADVREC EXE
[VADVREC]
FOLDER_NAME=
TEXT_FILE_NAME=
LOGFILE_NAME=
```

Testing

Overview

Error Scenarios and how to handle them

Glossary

TESTING

Software Testing is the process of testing software to ensure it behaves in a controlled manner to ensure it behaves the way it is expected to behave. Software testing is, thus, a critical element of software quality assurance. Testing require3s that the developer discard preconceived notions of the correctness of the software just developed and overcome the conflict of the interest that occurs when errors are developed.

The following testing objectives are to be remembered.

- Testing is a process of executing a program with the indent of identifying the error.
- A good test case is one that has high probability of finding an undiscovered error.
- A successful test is one that uncovers an error that is yet to be discovered.
- The level of testing that a software goes through is given below.

    - Unit testing
    - System testing
    - Integration testing
    - Acceptance testing.

Although testing is an integral part of Software Development, it is a complex process by its own. Every project has a separate testing team that works in conjunction with the development team. However, the testing team requires key inputs from the development team to carry out its activities. The scope of testing for them development team is thereby reduced to a great extent.

Given are the various error conditions that might occur and how the development team handles them.

Error Handling

Services:

Outbound:

Error: Record Type is missing.

How to Handle:

Send it to Printer.

Store it in a text file.

Log into the error table stating the path of the text file.

Error: Mandatory Field is missing

How to Handle:

Send it to Printer.

Store it in a text file.

Log into the error table stating the path of the text file.

Error: MQ Series down

How to Handle:

Retry Specified number of times.

If it is still down log into error table and shut down the service.

Error: SQL Server down

How to Handle:

Retry Specified number of times.

If it is still down log into the log file and shut down the service.

Inbound:

Error: MQ Series down

How to Handle:

Retry Specified number of times.

If it is still down log into error table and shut down the service.

Error: SQL Server down

How to Handle:

Retry Specified number of times.

If it is still down log into the log file and shut down the service.

Exe:

Outbound

Error: Text File or folder not found

How to Handle:

Log into the error table as well as the Log File.

Error: SQL Server down

How to Handle:

Retry Specified number of times.

If it is still down log into the log file and shut down the service.

Inbound:

Error: Record Type is missing or the sequence is wrong.

How to Handle:

Log into the error table stating the Error Record Number.

Log the Information into Log File.

Skip the incorrect data and start scanning the next record.

Error: SQL Server down

How to Handle:

Retry Specified number of times.

If it is still down log into the log file and shut down the service.

Conclusion and Future References

This is a client project catering to the needs of a particular organization. However there is a lot of reusable components in terms of framework, code and design that can be used for subsequent Business Solutions. The objective if this business solution is explained in the synopsis of this document.
The implementation of this project will take time, as there is lot more to be done in the external world.

The fundamental components of this project are:

1) There is a service routine, which keeps polling the database and the queue and updates the database if there is any message in the queue and updates the queue if there is any message in the database.

2) There is a Message oriented Middleware that communicates from the internal world to the external world and vice versa.

3) There is a routine that maintains a local table in the warehouse and uses a text file to append it to the table.

4) There are error conditions ands other unexpected scenarios, a few of which can be handled locally.

Technical Lessons Learnt:

Database Programming using VC++

Database Programming Using Visual Basic

Writing Windows NT Services

Writing MQ Interfaces – MQ Series

Messaging and Message Oriented Middleware – Concepts

References:

Standards for programming- Satyam Lib 23451prg

John Paul Mueller "Visual C++ 6.0" 1996,PMH publications.

Visual Basic 5.0 – Course material by NIIT

Microsoft SQL Server - Course material by NIIT

IBM MQSeries Reference – IBM Books.

www.codeguru.com

www.codeproject.com

www.ibm.com/books

Search Engine Used:

www.google.com

www.ask.com