A

Seminar report

On

Middleware Technologies

Submitted in partial fulfillment of the requirement for the award of degree
of Bachelor of Technology in Computer Science

SUBMITTED TO:
www.studymafia.org

SUBMITTED BY:
www.studymafia.org

Acknowledgement

I would like to thank respected Mr…….. and Mr. ……..for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a seminar report. It helped me a lot to realize of what we study for.

Secondly, I would like to thank my parents who patiently helped me as i went through my work and helped to modify and eliminate some of the irrelevant or un-necessary stuffs.

Thirdly, I would like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Next, I would thank Microsoft for developing such a wonderful tool like MS Word. It helped my work a lot to remain error-free.

Last but clearly not the least, I would thank The Almighty for giving me strength to complete my report on time.

Preface

I have made this report file on the topic **Middleware Technologies**; I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to ……..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

Middleware Technology

TABLE OF CONTENTS

TABLE OF FIGURES

Introduction

Middleware, which is quickly becoming synonymous with enterprise applications integration (EAI), is software that is invisible to the user. It takes two or more different applications and makes them work seamlessly together. This is accomplished by placing middleware between layers of software to make the layers below and on the sides work with each other (Figure 1). On that broad definition, middleware could be almost any software in a layered software stack. Further, middleware is a continually evolving term. Since much of the software business is driven through the perceptions of the "hottest" current technologies, many companies are giving their software the name "middleware" because it is popular.

Middleware, or EAI, products enable information to be shared in a seamless real-time fashion across multiple functional departments, geographies and applications. Benefits include better customer service, accurate planning and forecasting, and reduced manual re-entry and associated data inaccuracies.

Middleware is essential to migrating mainframe applications to client/server applications, or to Java or internet-protocol based applications, and to providing for communication across heterogeneous platforms. This technology began to evolve during the 1990s to provide for interoperability in support of the move to client/server architectures.[1[1]] There are two primary applications for middleware using any of the above middleware initiatives: Computer Telephony and Software Interfaces such as via Java based middleware applications. In this discussion of middleware, we will explore both uses.

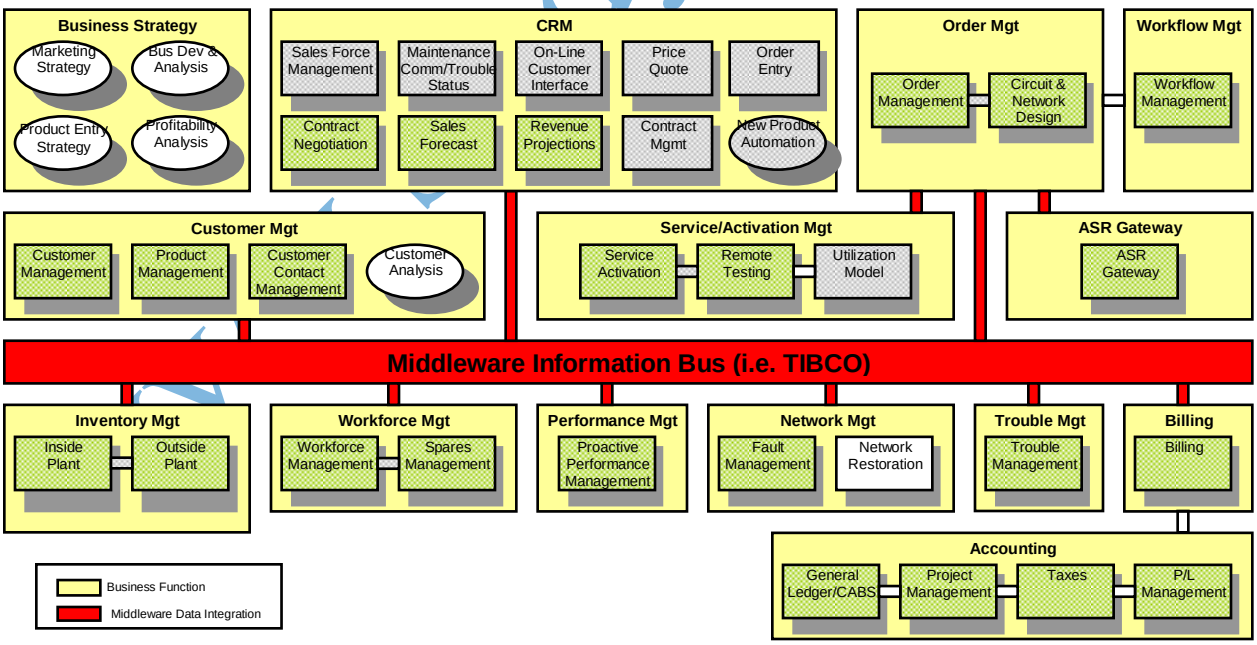

Figure 1: Middleware Software "Bus" Architecture

Brief History of EAI/Middleware

Enterprise applications, from as early as the 1960s through the late 1970s, were simple in design

and functionality, developed largely in part to end repetitive tasks. "There was no thought whatsoever given to the integration of corporate data. The entire objective was to replicate manual procedures on the computer."[2][2]

By the 1980s, several corporations were beginning to understand the value and necessity for application integration. Challenges arose, though, as many corporate IT staff members attempted to redesign already implemented applications to make them appear as if they were integrated. Examples include trying to perform operational transaction processing (associated with enterprise resource planning (ERP) system functionality) on systems designed for informational data processing (data warehousing functionality).

As ERP applications became much more prevalent in the 1990s, there was a need for corporations to be able to leverage already existing applications and data within the ERP system; this could only be done by introducing EAI (Figures 2 & 3). "Companies once used client/server technology to build departmental applications, but later realized the gains in linking multiple business processes."[3][3] Other issues driving the EAI market include the further proliferation of packaged applications, applications that addressed the potential problems of the Year 2000, supply chain management/business-to-business (B2B) integration, streamlined business processes, web application integration, and overall technology advances within EAI development.[4][4]

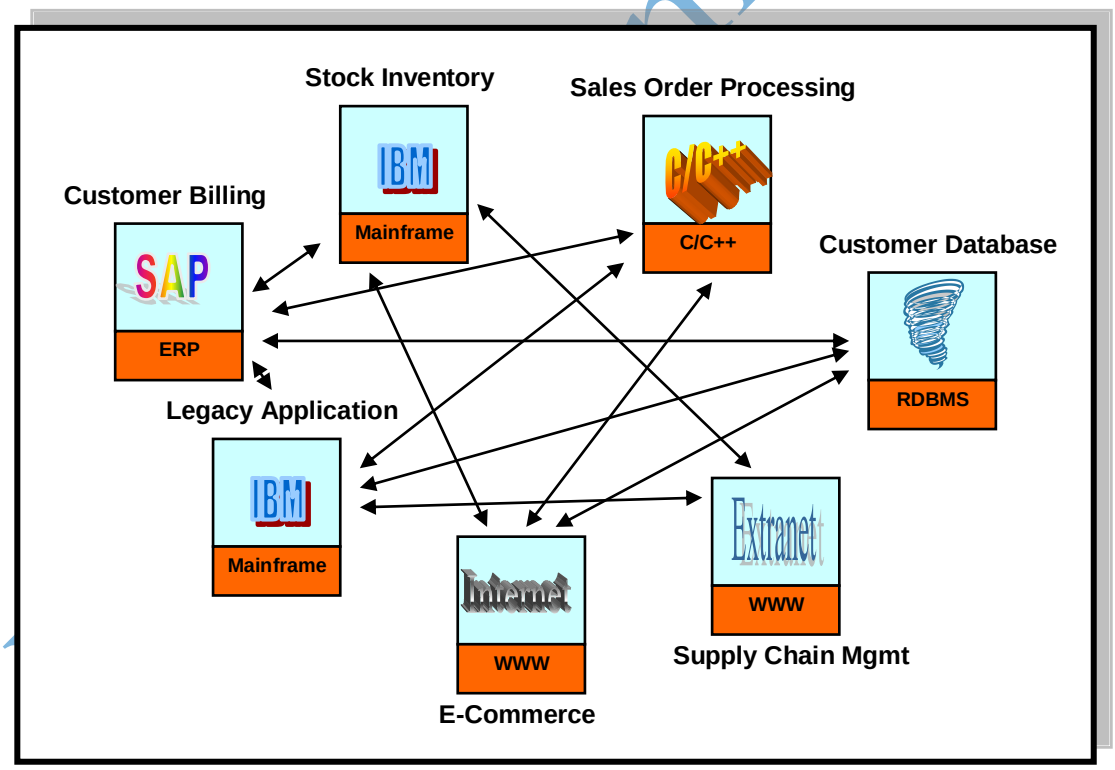

Figure 2: Legacy Enterprise Situation

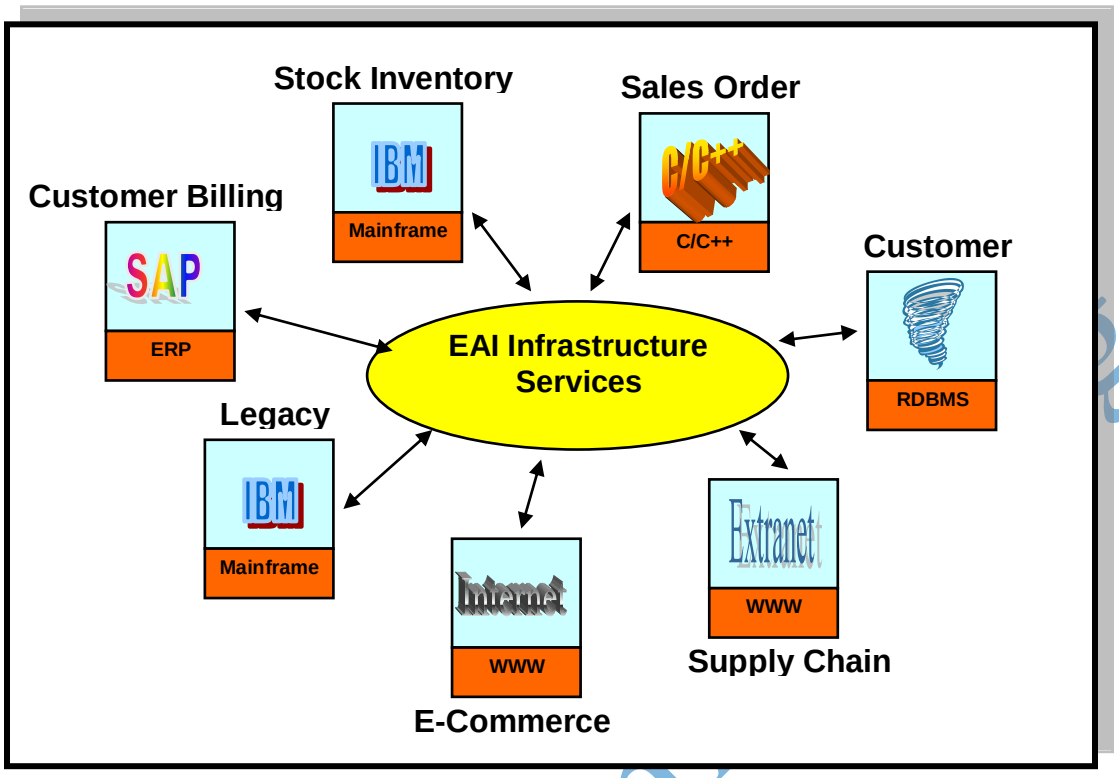

Figure 3: Middleware/EAI Enterprise Solution

What is involved in EAI/Middleware?

EAI is very involved and complex, and incorporates every level of an enterprise system – its architecture, hardware, software and processes. EAI involves integration at the following levels:

- ***Business Process Integration (BPI):*** When integrating business processes, a corporation must define, enable and manage the processes for the exchange of enterprise information among diverse business systems. This allows organizations to streamline operations, reduce costs and improve responsiveness to customer demands.[5[5]] Elements here include process management, process modeling, and workflow, which involve the combination of tasks, procedures, organizations, required input and output information, and tools needed for each step in a business process.
- ***Application Integration:*** At this level of integration, the goal is to bring data or a function from one application together with that of another application that together provide near real-time integration. Application Integration is used for, to name a few, B2B integration, implementing customer relationship management (CRM) systems that are integrated with a company's backend applications, web integration, and building Web sites that leverage multiple business systems. Custom integration development may also be necessary, particularly when integrating a legacy application with a newly implemented ERP application.
- ***Data Integration:*** In order for both Application Integration and Business Process Integration to succeed, the integration of data and database systems must be tackled. Prior to integration, data must be identified (where it is located), cataloged, and a metadata model must be built (a master guide for various data stores). Once these three steps are finished, data can then be shared/distributed across database systems.

- **Standards of Integration:** In order to achieve full Data Integration, standard formats for the data must be selected. Standards of Integration are those that promote the sharing and distribution of information and business data – standards that are at the core of Enterprise Application Integration/Middleware. These include COM+/DCOM, CORBA, EDI, JavaRMI, and XML.
- ***Platform Integration:*** To complete the system integration, the underlying architecture, software and hardware, and the separate needs of the heterogeneous network must be integrated. Platform Integration deals with the processes and tools that are required to allow these systems to communicate, both optimally and securely, so data can be passed through different applications without difficulty. For example, figuring out a way for an NT machine to pass information reliably to a UNIX machine is a large task for integrating an entire corporate system.[6][6]

Application Programming Interface (API)

In order to fully understand middleware, one must first understand the concepts surrounding Application Programming Interfaces (APIs). The API, by definition, is a software program that is used to request and carry out lower-level services performed by the computer's operation system or by a telephone system's operating system (Figure 4). In a Windows environment, APIs also assist applications in managing windows, menus, icons, and other GUI elements. In short, an API is a "hook" into software. An API is a set of standard software interrupts, calls, and data formats that application programs use to initiate contact with network services, mainframe communications programs, telephone equipment or program-to-program communications. For example, applications use APIs to call services that transport data across a network. Standardization of APIs at various layers of a communications protocol stack provides a uniform way to write applications. This technology is a way to achieve the total cross-platform consistency that is a goal of open systems.

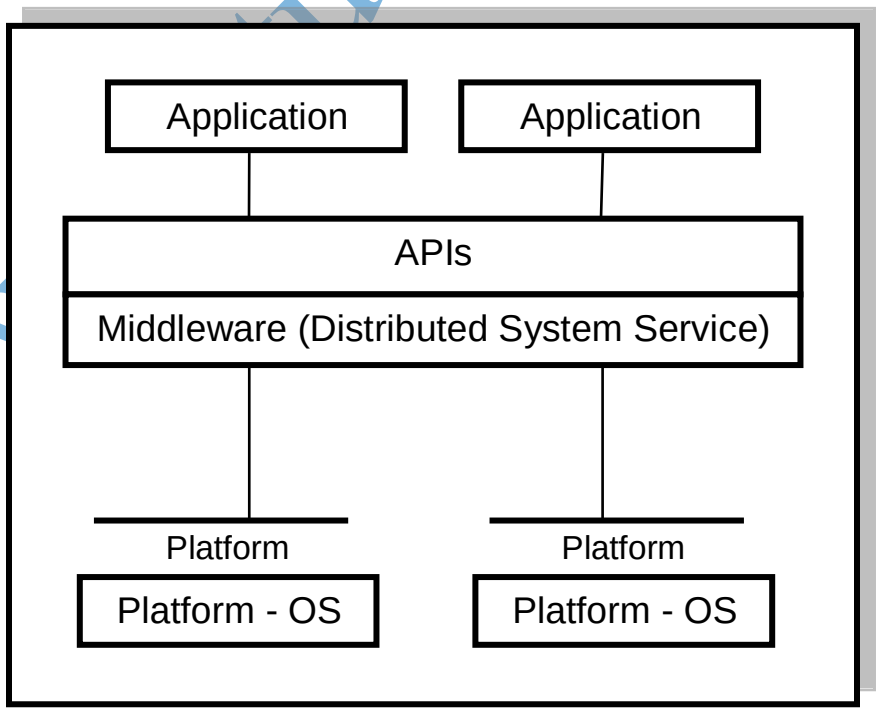

Figure 4: Basic API Architecture

Middleware/EAI Basics

As seen above in Figure 4, middleware works in concert with APIs. Further, it exists between the application and the operating system and network services on a system node in the network. Middleware services are sets of distributed software that provide a more functional set of APIs than does the operating system and network services. This increased functionality allows an application to:

- Locate transparently across the network, providing interaction with another application or service.
- Be independent from network services.
- Be reliable and available.
- Scale up in capacity without losing functionality.

Middleware accomplishes the above tasks via one of the following forms:

1. Transaction processing (TP) monitors, which provide tools and an environment for developing and deploying distributed applications.
2. Remote Procedure Call (RPCs), which enable the logic of an application to be distributed across the network. Program logic on remote systems can be executed as simply as calling a local routine.
3. Message-Oriented Middleware (MOM), which provides program-to-program data exchange, enabling the creation of distributed applications. MOM is analogous to email in the sense it is asynchronous and requires the recipients of messages to interpret their meaning and to take appropriate action.
4. Object Request Brokers (ORBs), which enable the objects that comprise an application to be distributed and shared across heterogeneous networks.
5. Transaction Flow Manager (TFM) or Intelligent Trade Management (ITM) – emerging technologies - which will act as a radar screen that tracks transactions from launch to landing.

Middleware and Computer Telephony

Middleware in computer telephony tends to be software that sits right above that part of the operating system that deals with telephony. This is the Telephone Server Application Programming Interface (TSAPI) in NetWare and the Telephone Application Programming Interface (TAPI) in Windows. Further, the middleware sits below the user interface and is, thus, invisible to the user.

TSAPI was described by AT&T, its' inventor, as "standards-based API for call control, call/device monitoring and query, call routing, device/system maintenance capabilities, and basic directory services."[7][7] TAPI is also called the Microsoft/Intel Telephony API. As stated above, the API is a software program that is used to request and carry out lower-level services performed by the computer's operation system or by a telephone system's operating system. In

the case of the TAPI, it is the telephone system's operating system. The TAPI set of functions allows windows applications (i.e. Windows 2000, NT) to program telephone-line-based devices such as single and multi-line phones (both digital and analog) and modems and fax machines in a device-independent manner. TAPI essentially does for telephony devices what the Windows printer system did to printers – makes them easy to install and allows many application programs to work with many telephony devices, irrespective of the device manufacturer.

TAPI is an evolutionary API providing convergence of both traditional PSTN telephony and IP Telephony. IP Telephony is an emerging set of technologies which enables voice, data, and video collaboration over existing LANs, WANs, and the Internet. TAPI enables IP Telephony on the Microsoft Windows operating system platform by providing simple and generic methods for making connections between two or more machines, and accessing any media streams involved in the connection (Figure 5).

In addition, TAPI also supports standards based H.323 conferencing (these standards define real-time multimedia communications for packet-based networks – now called IP Telephony) and IP multicast conferencing. Further, TAPI utilizes the Windows operating system's Active Directory service to simplify deployment within an organization, and includes quality of service (QoS) support to improve conference quality and network manageability.

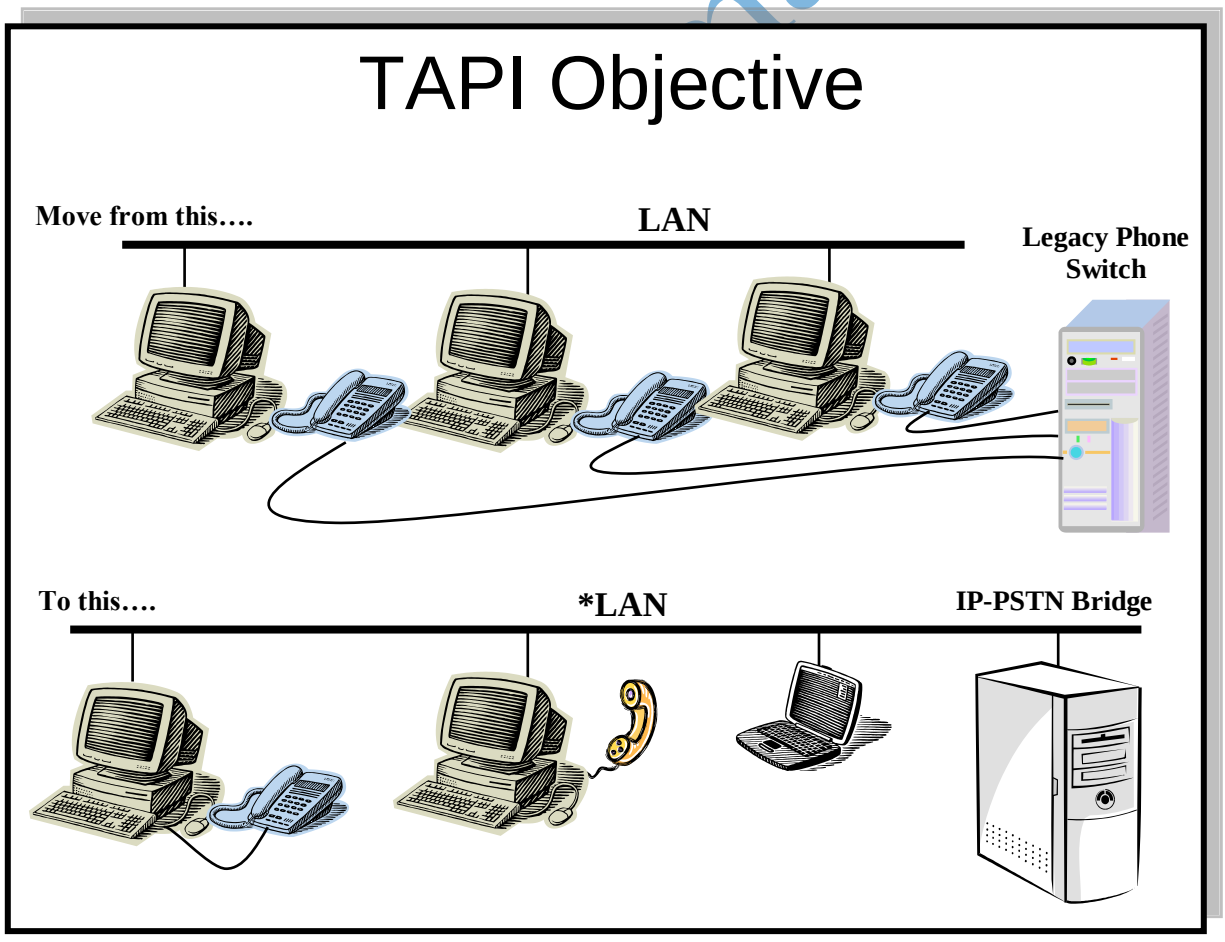

Figure 5: TAPI facilitates IP Telephony, which enables voice, data, and video over existing LANs, WANs, and the Internet.

Java Middleware – Evolving Use of EAI Technology

Java middleware encompasses application servers like BEA WebLogic, messaging products like Active Software's ActiveWorks and Push Technologies' SpiritWAVE, and hybrid products that build on a DBMS legacy and add server-based Java object execution features. Further, even among application servers there is quite a spectrum, including those that are primarily servlet servers as well as those that are ORB-based or OODB-based. Drawing a line between all these products proves increasingly difficult. The unifying feature, however, is that they all attempt to solve the multitier application deployment problem by using Java and Internet technologies. The business case to use Java in middleware is compelling. Among the advantages offered by Java middleware are the following:

- The ability of the internet to economically interconnect offices and organizations.
- The need for organizations to cooperate by sharing data and business processes.
- The desire to consolidate generic services and the management of these services.
- The desire to provide centralized application management, including startup, shutdown, maintenance, recovery, load balancing, and monitoring.
- The desire to use open services and protocols.
- The desire to redeploy business logic at will and unconstrained by infrastructure; this necessitates using open APIs and protocols, which are widely supported across most infrastructure products.
- The need to support cooperating mixed-architecture applications.
- The desire to move network and service infrastructure decisions out of the application space, so that system managers can make infrastructure decisions without being hampered by applications that depend on proprietary protocols or features.
- The desire to reduce the diversity and level of programmer staff skills needed and minimize the need for advanced tool-building expertise within projects.
- The desire to leverage object-oriented expertise by extending it into the server realm. Hence, newer object-oriented server products and object-to-relational bridges.

Since the goal of middleware is to centralize software infrastructure and its deployment, Java middleware is the next logical step in the evolution of middleware building upon the client/server roots. Organizations are now commonly attempting integration across departments, between organizations, and literally across the world. The key to building such integration is to leverage the existing technology of the internet. The internet has enticed businesses with its ability to serve as a global network that lets departments and partners interconnect efficiently and quickly.

Java provides a lingo that allows for easy interconnection of data and applications across organizational boundaries. In a distributed global environment that allows an organization no control over what technology choices partners make, smart companies choose open and platform-neutral standards. Companies cannot anticipate who will become their customers, partners, or subsidiaries in the future, so it is not always possible to plan for a common

infrastructure with partners. In this uncertain situation, the best decision is increasingly thought to be the use of the most universal and adaptable technologies possible.

Java allows for the reduction of the number of programming languages and platforms that a staff must understand. This is because Java is now deployed in contexts as diverse as internet browsers, stored procedures within databases, business objects within middleware products, and client-side applications.

Middleware Usage Considerations

Although middleware has numerous obvious benefits in solving application connectivity and interoperability problems, middleware services are not without tribulations. The main issues are outlined below:

- There is a gap between principles and practice. Many popular middleware services use proprietary implementations (making applications dependent on a single vendor's product). The commercial off-the-shelf (COTS) software vendors (i.e. TIBCO and Vitria) are addressing this problem through the more stringent use of standardization. In spite of this progress, the issue remains a substantial problem.
- The sheer number of middleware services is a barrier to using them. To keep their computing environment manageably simple, developers have to select a small number of services that meet their needs for functionality and platform coverage. Since the goal of middleware is to provide for maximum interoperability, this barrier is particularly frustrating.
- While middleware services raise the level of abstraction of programming distributed applications, they still leave the application developer with hard design choices. For example, the developer must still decide what functionality to put on the client and server sides of a distributed application.

The key to overcoming the above listed problems is to completely understand both the application problem and the value of middleware services that can enable the distributed application. To determine the types of middleware services required, the developer must identify the functions required. These functions all fall into one of three categories:

1. Distributed system services – These include critical communications, program-to-program, and data management services. RPCs, MOMs and ORBs are included in this type of service.
2. Application enabling services – These middleware services give applications access to distributed services and the underlying network. This type of service includes transaction monitors and database services such as Structured Query Language (SQL).
3. Middleware management services - These enable applications and system functions to be continuously monitored to ensure optimum performance of the distributed environment. Middleware Developmental Stage.

Middleware Developmental Stage

A significant number of middleware services and vendors exist. Middleware applications are continuing to grow with the installation of more heterogeneous networks and as the client/server architecture continues to entrench and evolve into Java-centric and Internet-protocol centric architectures, more middleware products are emerging. For example, at least 46 Java middleware products were in existence in 2000 and that number has nearly doubled to date. Such products will continue to grow in demand as examples such as the Delta Airlines Cargo

Handling System emerge. This system uses middleware technology to link over 40,000 terminals in 32 countries with UNIX services and IBM mainframes.

Middleware Costs, Limitations, and Economic Outlook

No technology is right for every situation and each technology has associated costs (monetary and otherwise). Some examples of the kinds of costs and limitations that a technology may possess are the following: A technology may impose an otherwise unnecessary interface standard, it might require investment in other technologies (see bullets below), it might require investment of time or money, or it may directly conflict with security or real-time requirements. Specific dependencies include:

- What is needed to adopt this technology (this could mean training requirements, skill levels needed, programming languages, or specific architectures).
- How long it takes to incorporate or implement this technology.
- Barriers to the use of this technology.
- Reasons why this technology would not be used.

After taking all of the above into consideration, the costs of using middleware technology (i.e. license fees) in system development are entirely dependent on the applications, required operating systems, and the types of platforms. As an example, at a start-up telecommunications carrier-of-carriers service provider, AFN Communications, there were approximately $20 million dollars expended just to implement operational and business support systems that were fully integrated across a client/server middleware information “bus”, which indicates that all applications are integrated via a common path. TIBCO software was utilized for this implementation and the use of this software required staff training, it took approximately one year to complete the project, a multi-platform and multi-OS situation presented continual barriers to using the information bus technology, and concerns over continued maintenance of the complicated integration were constantly discussed. In spite of all of the challenges, the heavily used middleware system is in place and has allowed their organization to reduce their necessary headcount by about 35%.

Another current example of the use of middleware technology is in the financial business sector. Financial institutions are preparing for T+1, the holy grail of the world's stock market settlement system. Under this exacting standard, trades would settle within one day instead of the two, three or even more days that it takes in some markets.[8][8] These financial institutions are preparing to spend a lot of time with their middleware vendors as a 2004 deadline for full implementation of T+1 looms. Middleware, or EAI, is an "absolutely fundamental requirement" for straight-through processing (STP). "Participating in end-to-end transactions without the middleware layer that provides messaging and securities and data exchange between applications running the business is not possible."[9][9] Further, without the advent and extension of the TFM or ITM transaction tracking capabilities from launch to landing within the middleware technology, it will be impossible to meet the T+1 goal.

Concentrating only on the aforementioned T+1 market need, the financial industry is expected to spend $8 billion. Although hesitant to venture a guess regarding the dollars that will be dedicated to EAI, estimates of spending by financial institutions alone, which includes

middleware, will hit $733 million this year and rise to $1.19 billion by 2003, a compounded annual growth rate of almost 25 percent.

EAI/Middleware Market Leaders and Sample Middleware Vendors and Earnings

Market Leaders

Because of the breadth and diversity of the EAI market, several software companies offer products that are integral to some aspect of application integration, but may not offer a comprehensive solution. EAI market leaders include:

1. **BEA Systems** (worldwide locations).
2. **CrossWorlds Software** (Australia).
3. **IONA Technologies** (Ireland, Scotland, California).
4. **Level 8 Systems** (North Carolina, California, New Jersey, Virgina, London, Milan, and Paris).
5. **Mercator Software** (England, Massachusetts).
6. **NEON** (purchased in 2001 by Sybase) (California).
7. **SeeBeyond** (California, multiple U.S. States, Australia, Korea, Japan, Singapore, Cyprus, multiple European locations).
8. **Software AG** (Germany).
9. **TIBCO** (California headquarters and worldwide offices).
10. **Vitria Technology** (California headquarters and offices in: Australia, Brazil, France, Germany, Italy, Japan, Korea, Singapore, Spain, Sweden, Switzerland, Taiwan, United Kingdom).
11. **webMethods** (Virginia headquarters with offices throughout the U.S., Europe and Asia Pacific.

Sample EAI/Middleware Vendors and Earnings
(Per WallStreet & Technology Online – July 3rd, 2002)

In spite of the extensive demand for middleware services both domestically and worldwide, individual companies specializing in these arenas are experiencing varied financial results:

“Messaging and identity management vendor Critical Path Inc. said it will meet Wall Street's expectations with revenues between $22 million and $24 million. The company did not specify its profit outlook, but said operating expenses excluding certain items will be $27 million to $29 million. It said those numbers are better than expected. In early afternoon trading, the company's shares were 11 percent higher, at 90 cents.

Integration software vendor SeeBeyond Technologies Corp. said it will report revenues between $32 and $34 million, with a loss of 8 cents to 10 cents per share. Analysts had expected a profit, as well as revenue of $38 million. Its shares dropped 17 percent to $2.10 in early afternoon trading.

Legacy extension/integration vendor Jacada Ltd. forecast revenues of $5.2 million to $5.5 million, and a net loss of 6 cents to 8 cents per share. Analysts were looking for a loss of 3 cents

per share. The company's shares were trading down 17 percent at $1.90 Wednesday afternoon." [10]

Conclusion

Potential Challenges of Middleware/EAI

Middleware product implementations are unique to the vendor. This results in a dependence on the vendor for maintenance support and future enhancements. Reliance on vendors, in this manner, could have a negative effect on a system's flexibility and maintainability. However, when evaluated against the cost of developing a unique middleware solution, the system developer and maintainer may view the potential negative effect as acceptable. Also, as Java and internet protocol middleware technologies evolve, many of these potentially detrimental issues will dissolve.

The Future of Java Middleware

Java technology is still somewhat immature. On the other hand, we may now be in an era when products never truly reach maturity because the underlying technologies on which they're based change so rapidly. In fact, there are significant documented problems with virtually every middleware product, including those supposedly mature products that have been on the market for years. The point is, by the time a vendor manages to fix problems, new features have been added. The cycle for adding new features is now much shorter than it has ever been. The result is that products do not have enough time to become stable before they include the next major feature set. This problem may be something that continues into the future. Further, the burden for determining the strengths and weaknesses of all chosen products will likely be a vital component of any application design and prototype cycle.

The Future of Middleware/EAI

IDC Research expects the EAI services market to become the most important and fastest-growing IT sector in the next three to five years. According to IDC research, "worldwide revenues in this market will jump from $5 billion in 2000 to nearly $21 billion in 2005. This increase represents a strong compound annual growth rate (CAGR) of over 30%. By comparison, the corresponding opportunity of the overall IT services industry will increase at a CAGR of 11% during the same period." IDC also reports that North America and Western Europe will generate more than 90% of the demand for global EAI services through 2005, with Japan and Latin America driving the remainder of this service demand. Issues that may inhibit the growth of EAI include, "cost of services, human issues regarding EAI engagements, and business-to-business integration challenges."[11]

APPENDIX A

Middleware Glossary[xii][12]

Activation
Bringing an executable component into a live state, after which it can respond to invocations

Application server
A server program that allows the installation of application specific software components, in a manner so that they can be remotely invoked, usually by some form of remote object method call.

Bean-managed persistence
When an Enterprise JavaBean performs its own long-term state management

Bytecode
In the context of Java, bytecode is the platform-independent executable program code

Class loader
A Java class that serves the function of retrieving other Java classes and loading them into memory

Clustering
Aggregating multiple servers together to form a service pool of some kind, usually for achieving redundancy or improving performance

Component standard
A definition of how software components cooperate, and in particular the roles and interfaces of each. In the context of Java middleware, component standards usually include specifications of the middleware interfaces exposed to the components, and the component interfaces required by the middleware

Concentrator
A facility for aggregating requests onto a single or small number of channels, for efficiency.

Container managed persistence
When an Enterprise JavaBean server manages a bean's long-term state.

CORBA
Standard maintained by the Object Management Group (OMG), called the Common Object Request Broker Architecture.

COS Naming
CORBA standard for object directories.

Data source
This is the term used by the JTA and JDBC specifications to refer to persistent repository of data. It usually represents a database. It also may refer to an object that makes database connections available (i.e. a driver).

DCOM
Microsoft's Distributed Component Object Model.

DNS
Domain Name Service, the Internet standard for looking up machine IP addresses by logical name.

EJB
Enterprise JavaBeans.

Enterprise JavaBeans
A server component standard developed by Sun Microsystems

Entity bean
An Enterprise JavaBean that maintains state across sessions, and may be looked up in an object directory by its key value

Failover
The ability to respond resiliently to a component failure by switching to another component

IDL
interface description language, CORBA's syntax for defining object remote interfaces

IIOP
Internet Inter-ORB Protocol, CORBA's wire protocol for transmitting remote object method invocations

ISAPI

Microsoft's C++ API for coding application extensions for its Internet Information Server

Java Naming and Directory Interface
The Java standard API for accessing directory services, such as LDAP, COS Naming, and others

Java Transaction API
Java API for coding client demarcated transactions, and for building transactional data source drivers

JDBC2
Newly released extensions to the JDBC API

JNDI
Java Naming and Directory Interface

JTA
Java Transaction API

JTS
The Java Transaction Service, which in the Java binding for the CORBA Transaction Service. Provides a way for middleware vendors to build interoperable transactional middleware

JVM
Java virtual machine

Keystore
A repository for private keys and certificates

LDAP
Lightweight Directory Access Protocol, a protocol for directory services, derived from X.500

Messaging middleware
Middleware that supports a publish-and-subscribe or broadcast metaphor

Middleware
Software that runs on a server, and acts as either an application processing gateway or a routing bridge between remote clients and data sources or other servers, or any combination of these

MOM
message-oriented middleware

NSAPI
Netscape's C language API for adding application extensions to their Web servers

OMG
Object Management Group, an organization that defines and promotes object oriented programming standards

OODB
object-oriented database

OODBMS
object-oriented database management system

ORB
object request broker, the primary message routing component in a CORBA product

Passivate
To place an object in a dormant state when it is not being accessed, such that it can later be returned to an active and usable state

Persistence
Maintaining state over a long time, especially across sessions

POA
portable object adapter

Pooling
Maintaining a collection of objects, servers, connections, or other resources for ready access, so that one does not need to be created anew each time one is needed

Portable Object Adapter
A new CORBA standard for defining object lifecycle and activation

Quality of service
The set of characteristics of a connection that include response time, reliability and error rate, throughput, and other measures that impact usability

RMI

Remote Method Invocation, the Java standard technology for building distributed objects whose methods can be invoked remotely across a network

RMI over IIOP

Using the CORBA IIOP wire protocol from an RMI API

Servant

Loosely, an object that exposes a remote interface so that it can be called from a remote client. Has a very specific meaning in the context of the POA standard

Servlet

An application extension to a Java Web server

Session bean

An Enterprise JavaBean that does not maintain its state from one session to the next. Appears to the client as if the bean was created just for that client

Skeleton

A server-side software component that serves to relay remote calls from a client to the methods of a servant running in a server. Usually a skeleton is automatically generated by a special compiler

SQLJ

An extended Java syntax for embedding SQL-like commands in a Java program

Stub

A client-side software component that serves to forward remote calls to a remote server, and receive the subsequent responses. Usually automatically generated by a special compiler

Three-tier

An architecture in which a remote client access remote data sources via an intervening server

Transaction manager

A software component that coordinates the separate transactions of multiple data sources, so that they behave as a single unified transaction. Requires data source drivers that can participate in this kind of coordination. Also usually provides the ability to monitor transactions and provide statistics

Transactional

When an operation has the property that it either completes, or if it does not complete due to a failure, it either undoes its own effects or has the ability to complete at a later time when the failure is repaired

Reference

- www.google.com
- www.wikipedia.org
- www.studymafia.org
- www.projectsreports.org