Evolution of Middleware Technology and Its Widespread Applications

Shweta Roy
Department of Computer Science, Ajay Kumar Garg Engineering College, P.O. Adhyatmic Nagar, Ghaziabad
sguddr@gmail.com

***Abstract*-- Service-oriented architectures are poised to transform the industrial scene by enabling more flexible and agile IT infrastructures. The key change agent in this transformation is middleware. Middleware optimises the cost and delivery of IT services.**

This article explains how Middleware software evolved to become a technological "glue" between software components to provide support and simplify complex, distributed applications.

Keywords: Middleware, Service-Oriented Architectures, Simulation Technology, Object Request Broker

I. INTRODUCTION

Today, industries need to transform their client/server infrastructures into services-oriented setups to stay competitive. Focus of IT has shifted from a technology-centric approach to a flexibility-driven approach measured in time-to-delivery and ability to change.

Though it is universally accepted that service-oriented architectures implementations lead to quantifiable benefits, yet in practice, their adoption has been sluggish.

The strategy to remedy this situation is via middleware.

In the computer industry, middleware is a general term for any programming that serves to "glue together" or mediate between two separate and often already existing programs.

In essence, Middleware is a computer software that interconnects software components or applications. This software consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network. Middleware is especially integral to modern information technology based on XML, SOAP, Web services, and service-oriented architecture.

A common application of middleware is to allow programs written for access to a particular database to access other databases. Typically, middleware programs provide messaging services so that different applications can communicate.

The systematic tying together of disparate applications, often through the use of middleware, is known as enterprise application integration.

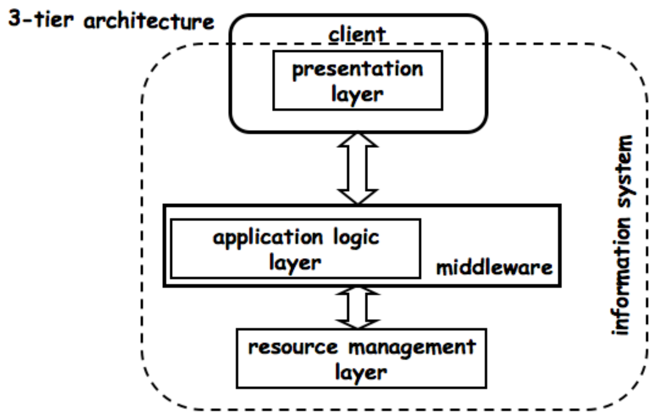

Figure 1: Typical Middleware setup

This technology evolved to provide for interoperability [1] in support of coherent distributed architectures, which are used most often to support and simplify complex, distributed applications. It includes web servers, application servers, and similar tools that support application development and delivery.

II. BACKGROUND

How did middleware evolve from necessary evil—proprietary plumbing that glued disparate systems together—to one of the most strategic areas of IT and business today?

Because businesses, institutions, and technologies change continually, the software systems that serve them must be able to accommodate such changes.

Following a merger, the addition of a service, or the expansion of available services, a business can ill afford to recreate its information systems. It is at this most critical point that it needs to integrate new components or to scale existing ones as efficiently as possible.

The easiest way to integrate heterogeneous components is not to recreate them as homogeneous elements but to provide a layer that allows them to communicate despite their differences.

This layer, called middleware, allows software components (applications, enterprise java beans, servlets, and other components) that have been developed independently and that run on different networked platforms to interact with one another. It is when this interaction is possible that the network can become the computer.

Service-oriented architectures get many of their core services directly from middleware, including critical security functionality, deployment and management capabilities. One also finds business intelligence, content and collaboration tools, as well as portal capabilities that allow connections to customers and partners enabled at the middleware level.

The convergence of these critical business services at the middleware layer is reflective of the mid-tier's strategic position within the enterprise. The broad range of capabilities offered by today's middleware products enables industry to:

- Support and accelerate business expansion
- Deliver greater insight into business issues and drivers
- Reduce exposure to risk and support governance initiatives.

Using these tools within a standards-based SOA environment, corporations can leverage data in more strategic ways to deliver accurate, actionable information to business decision-makers when and where they need it.

It's not difficult to see why selecting the right middleware solutions should be among the top priorities of CIOs and other technology decision makers.

III. BASIC CONFIGURATION

Middleware sits "in the middle" between application software working on different operating systems. It is similar to the middle layer of a three-tier single system architecture, except that it is stretched across multiple systems or applications.

Examples of Middleware include database systems, telecommunications software, transaction monitors, and messaging-and-queuing software [2].

The distinction between operating system and middleware functionality is, to some extent, arbitrary. While core kernel functionality can only be provided by the operating system itself, some functionality previously provided by separately sold middleware is now integrated in operating systems. A typical example is the TCP/IP stack for telecommunications, nowadays included in virtually every operating system.

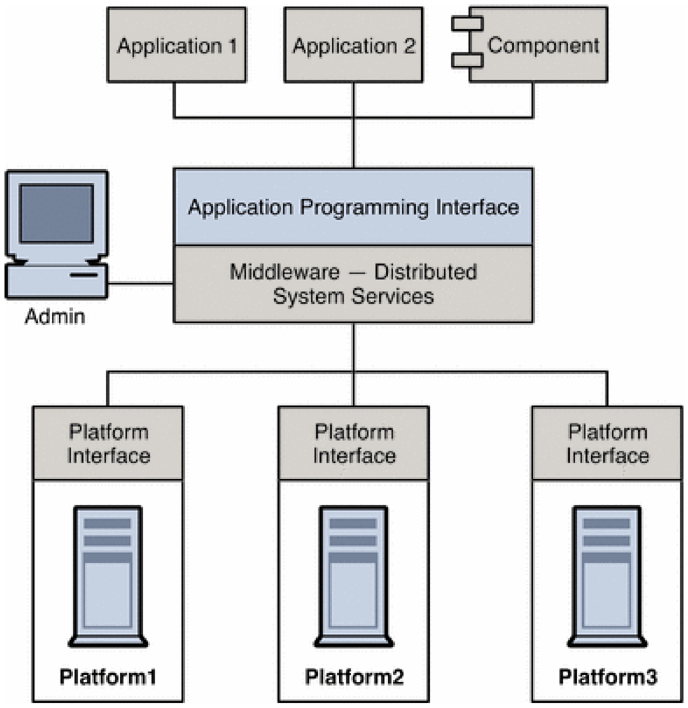

Figure 2 Middleware resides between the application layer and the platform layer (the operating system and underlying network services).

In simulation technology, *middleware* is generally used in the context of the high level architecture (HLA) that applies to many distributed simulations. It is a layer of software that lies between the application code and the run-time infrastructure.

Middleware generally consists of a library of functions, and enables a number of applications – simulations or federates in HLA terminology – to page these functions from the common library rather than re-create them for each application.

IBM, Red Hat, and Oracle Corporation are major vendors providing middleware software. Vendors such as SAP, TIBCO, Mercator Software, Crossflo, Vitria and webMethods were specifically founded to provide Web-oriented middleware tools.

Groups such as the Apache Software Foundation and the ObjectWeb Consortium encourage the development of open source middleware.

IV. MIDDLEWARE APPLICATIONS

Middleware services provide a more functional set of application programming interfaces to allow an application to:

- Locate transparently across the network, thus providing interaction with another service or application.

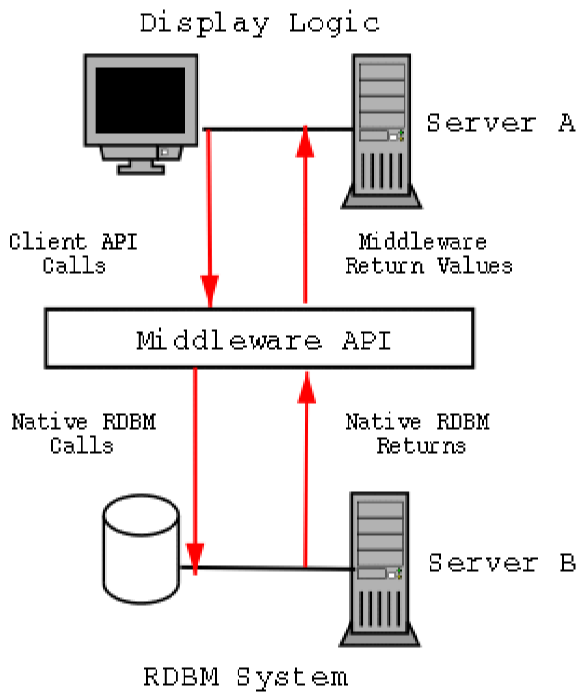

Figure 3 Middleware in Client-server architecture

- Be independent from network services
- Be reliable and available always when compared to the operating system and network services.

V. TYPES OF MIDDLEWARE

Hurwitz's classification system organizes the many types of middleware that are currently available.. These classifications based on scalability and recoverability are mentioned below.

Remote procedure call (RPC) is an Inter-process communication technology that allows a computer program to cause a subroutine or procedure to execute in another address space without the programmer explicitly coding the details for this remote interaction.

Thus, the programmer would write essentially the same code whether the subroutine is local to the executing program, or remote. When the software in question is written using object-oriented principles, RPC may be referred to as remote invocation or remote method invocation. Client makes calls to procedures running on remote systems, which can be asynchronous or synchronous.

Message Oriented Middleware—Message-oriented middleware (MOM) is a client/server infrastructure that increases the interoperability, portability, and flexibility of an application by allowing the application to be distributed over multiple heterogeneous platforms. It reduces complexity of developing applications that span multiple operating systems and network protocols by insulating the application developer from details of various operating system and network interfaces.

APIs that extend across diverse platforms and networks are typically provided by the MOM. MOM is software that resides in both portions of client/server architecture and typically supports asynchronous calls between the client and server applications. Message queues provide temporary storage when the destination program is busy or not connected.

MOM reduces the involvement of application developers with the complexity of the master-slave nature of the client/server mechanism.

MOM comprises a category of inter-application communication software that generally relies on asynchronous message-passing, as opposed to a request-response metaphor. Most message-oriented middleware depend on a message queue system, but there are some implementations that rely on broadcast or multicast messaging systems.

Messages sent to the client are collected and stored until they are acted upon, while the client continues with other processing.

Object Request Broker — In distributed computing, an object request broker (ORB) is a piece of middleware software that allows programmers to make program calls from one computer to another via a network. ORBs promote interoperability of distributed object systems because they enable users to build systems by piecing together objects from different vendors that communicate with each other via the ORB.

ORB's handle the transformation of in-process data structures to and from the byte sequence, which is transmitted over the network. This is called marshalling or serialization.

Some ORB's, such as CORBA-compliant systems, use an Interface Description Language (IDL) to describe the data which is to be transmitted on remote calls.

In addition to marshalling data, ORB's often expose many more features, such as distributed transactions, directory services or real-time scheduling.

In object-oriented languages, the ORB takes the form of an object with methods enabling connection to the objects being served. After an object connects to the ORB, the methods of that object become accessible for remote invocations.

The ORB requires some means of obtaining the network address of the object that has now become remote. The typical ORB also has many other methods.

VI. IMPLEMENTATION OF ORBS

Implementation of ORBs is possible via different ways as under.

- CORBA - the Common Object Request Broker Architecture.
- Ice - the Internet Communications Engine
- .NET Remoting - object remoting library within Microsoft's .NET Framework
- Windows Communication Foundation
- ORBexpress - real-time ORBs by Objective Interface Systems
- Orbix - An Enterprise-level CORBA ORB from IONA Technologies
- DCOM - the Distributed Component Object Model from Microsoft
- RMI - the Remote Method Invocation Protocol from Sun Microsystems
- RPC - Remote Procedure Call
- SimpleORB - a small, non-CORBA ORB
- ORBit - an open-source CORBA ORB used as middleware for GNOME
- OmniORB - a CORBA-compliant ORB released under the GPL
- opalORB - a CORBA implementation completely written in Perl.

This type of middleware makes it possible for applications to send objects and request services in an object-oriented system.

SQL-oriented Data Access: This is a middleware between applications and database servers.

Embedded Middleware denotes communication services and integration interface software/firmware that operate between embedded applications and the system software. In literature additional classifications of Middleware are found. These include:

Transaction processing monitors —These provide tools and an environment to develop A Transaction Processing System or Transaction Processing Monitor that monitors *transaction programs* (a special kind of program).

The essence of a transaction program is that it manages data that must be left in a consistent state *e.g.* if an electronic payment is made, the amount must be either both withdrawn from one account and added to the other, or none at all. In case of a failure preventing transaction completion, the partially executed transaction must be 'rolled back' by the TPS.

While this type of integrity must be provided also for batch transaction processing, it is particularly important for online processing: if for example, an airline seat reservation system is accessed by multiple operators, after an empty seat inquiry, the seat reservation data must be locked until the reservation is made, otherwise another user may get the impression a seat is still free while it is actually being booked at the time. Without proper transaction monitoring, double bookings may occur.

Other transaction monitor functions include deadlock detection and resolution (deadlocks may be inevitable in certain cases of cross-dependence on data), and transaction logging (in 'journals') for 'forward recovery' in case of massive failures.

Transaction Processing is not limited to application programs. For example, the 'journaled file system' provided with IBM's AIX Unix operating system employs similar techniques to maintain file system integrity, including a journal.

Application servers — This Middleware software is installed on a computer to facilitate the serving function. An application server, in an n-tier architecture, is a server that hosts an API to expose business logic and business processes for use by third-party applications. The term refers to:

1. The services that are made available by the server
2. The computer hardware on which the services are deployed
3. The software framework used to host the services such as JBoss application server or Oracle Application Server
4.

Enterprise Service Bus — This is an abstraction layer on top of an Enterprise Messaging System. In computing, an enterprise service bus (ESB) refers to a software architecture construct. This construct is typically implemented by technologies found in a category of middleware infrastructure products, usually based on recognized standards, which provide fundamental services for complex architectures via an event-driven and standards-based messaging engine (the bus). An ESB generally provides an abstraction layer on top of an implementation of an enterprise messaging system, which allows integration architects to exploit the value of messaging without writing code

Contrary to the more classical enterprise application integration (EAI) approach of a monolithic stack in a hub and spoke architecture, the foundation of an enterprise service bus is built of base functions broken up into their constituent parts, with distributed deployment where needed, working in harmony as necessary.

An ESB does not implement a service-oriented architecture (SOA) but provides the features with which one may be implemented. Though it is a common belief, an ESB is not necessarily web-services based. An ESB should be standards-based and flexible, supporting many transport mediums. Based on EAI rather than SOA patterns, it tries to remove the coupling between the service called and the transport medium.

VI. WEB SERVICE

A Web Service is defined as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-process-able format (specifically WSDL, Web Service Description Language). Other systems interact with the Web service in a manner prescribed by its description using SOAP-(Simple Object Access Protocol) [3] messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

Web services are frequently just Internet Application Programming Interfaces (API) that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. Web Services are registered at UDDI (Universal Description Discovery & Integration).Other approaches with nearly the same functionality as web services are Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA), Microsoft's Distributed Component Object Model (DCOM) or Sun Microsystems's Java/Remote Method Invocation (RMI).

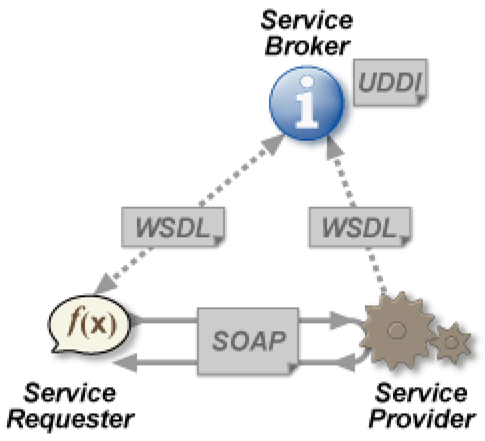

Figure 4. A typical Web-service implementation.

VII. DISTRIBUTED COMPUTING

Middleware is implemented in Distributed computing which deals with hardware and software systems containing more than one processing element or storage element, concurrent processes, or multiple programs, running under a loosely or tightly controlled regime.

In distributed computing, a program is split into parts that run simultaneously on multiple computers communicating over a network. Distributed computing is a form of parallel computing, but parallel computing is most commonly used to describe program parts running simultaneously on multiple processors in the same computer.

Both types of processing require dividing a program into parts that can run simultaneously, but distributed programs often deal with heterogeneous environments, network links of varying latencies, and unpredictable failures in the network or the computers.

If not planned properly, a distributed system can decrease the overall reliability of computations if the unavailability of a node can cause disruption of the other nodes. Leslie Lamport famously quipped that: "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

Troubleshooting and diagnosing problems in a distributed system can also become more difficult, because the analysis may require connecting to remote nodes or inspecting communication between nodes.

VIII. EXAMPLES

Berkeley Open Infrastructure for Network Computing (BOINC), became useful as a platform for several distributed applications in areas as diverse as mathematics, medicine, molecular biology, climatology, and astrophysics.

A variety of distributed computing projects have grown up in recent years. Many are run on a volunteer basis, and involve users donating their unused computational power to work on interesting computational problems.

Examples of such projects include the Stanford University Chemistry Department Folding@home project, which is focused on simulations of protein folding to find disease cures and to understand biophysical systems; World Community Grid, an effort to create the world's largest public computing grid to tackle scientific research projects that benefit humanity, run and funded by IBM; SETI@home, which is focused on analyzing radio-telescope data to find evidence of intelligent signals from space, hosted by the Space Sciences Laboratory at the University of California, Berkeley;

LHC@home, which is used to help design and tune the Large Hadron Collider, hosted by CERN in Geneva.

VIII. REFERENCES

[1]. U. S. Corporation. White Paper: SIP and SOAP. http://www.sipforum.org/whitepapers/ USC-SIPSOAP-WP2.pdf.

[2]. R.E. Schantz and D.C. Schmidt, "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications," *Encyclopedia of Software Eng.*, Wiley & Sons, New York, 2001; also available at http://www.cs.wustl.edu/ ~schmidt/PDF/middleware-chapter.pdf.

[3]. F. Curbera *et. al*., "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, vol. 6, no. 2, March/April 2002, pp. 86-93.

Shweta Roy obtained B.Sc Engg in Computer Science and Engineering from Magadh University, Gaya in 2001.

Since last four years, she is teaching at Ajay Kumar Garg Engineering College where, she is Assistant Professor in the Department of Computer Sciences. Concurrently, Ms Roy is doing MTech from the UP Technical University in the area of Middleware Web Services.

She has abiding passion for teaching and has taught a number of courses namely Computer Networks, Compiler Design, Software Engineering, Automata Theory, Java Programming and C Programming Concepts.