A perspective on evolution of middleware technology supporting Internetware*

ZHOU Minghui†, WANG Qianxiang, JIAO Wenpin, MEI Hong

1. Software Institute, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China
2. Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China
† Corresponding author: Phn: +86-10-62757670, Fax:+86-10-62751792,E-mail: zhmh@sei.pku.edu.cn, http://www.sei.pku.edu.cn

Abstract: With the growing of network, middleware is playing an important role in the development, deployment and management of typical applications. Although researchers have different understanding with middleware, there are still many developing opportunities left for middleware, especially when grid computing, pervasive computing and service computing are driving the increase of new technologies. This paper presents a perspective of middleware, and proposes further a novel evolution model for next generation middleware.

Key words: Middleware; Evolution; Internetware

Introduction

Stimulated by the growth of network-based applications, middleware technologies are taking an increasing importance. In common sense, middleware is computer software that sits ‘in the middle’ between applications (e.g. Web-based Shopping System, Custom Relationship Management Software) and operating systems (e.g., Unix, Windows). It is widely used because it can ease the task of designing, programming, deploying and managing distributed applications by providing a simple, consistent and integrated distributed programming environment [1, 2].

But the middle layer between applications and the operating systems is a quite broad concept. Starting from interoperability abstraction in client/server architecture, middleware today is used widely to support complex and distributed applications [3]. In fact middleware represents different kinds of software, e.g., Reflective Middleware, Event-Based Middleware, Object-Oriented Middleware, Message-Oriented Middleware, and so on [4]. How to give a clear definition for middleware is still keeping an open issue.

In recent years, the emergence of new computing models, e.g., grid computing,
Pervasive computing, service computing, influences the software behavior greatly. In order to adapt to the deep appliance on the Internet, software has to be cooperative, evolutive, and context-aware for its survival, and growing-up. To meet the above challenges and take advantage of the new opportunities, a new paradigm of software is presented which proposes the characteristics of autonomous, cooperative, polymorphic, reactive, and evolutive. This paradigm of software is named "Internetware"[5]. As a result, existing methodologies are not sufficient for the development, deployment and management under the paradigm. A series of new technologies are appealed to support the paradigm “Internetware”.

Middleware is one of such related technologies. Now, we have to face with some problems: How should we recognize middleware today? How do we know the future trend of middleware, especially when the network is becoming more dynamic, open and uncontrollable? What are the current challenges for middleware, especially supporting Internetware?

To explore answers to these questions, in section 2, we investigate the existing middleware technologies, and propose two important factors that influence the trend of middleware: application abstraction and machine virtualization. In section 3, we survey the middleware challenges brought by the current grand applications, and present a novel evolutionary model of middleware to support the Internetware paradigm. Section 4 concludes this paper.

2 A perspective of middleware

When network-based applications grew along with the network, how to share the network resources and make them cooperate with each other became an important issue. The basic solution is to abstract the resources and exhibit a simple view to developers and users. The problem is that we have to cope with the existing technologies. That means, the underlying software is already there, i.e., the operating systems, the network protocol stack, and so on. The diversity is already there, all we have to do is to abstract the diversity, which may generate a high level diversity at the same time. Different kinds of abstraction generate different kinds of middleware. In this paper, we propose two aspects of abstraction. One is from the view of the machine virtualization, and the other is from the application abstraction.

2.1 An overview of middleware

One of the objectives of middleware is to hide the heterogeneity of the networking environments, support advanced coordination models among distributed entities and make the distribution of computations as transparent as possible. This covers the application infrastructure supporting the development, deployment and integration, and management of applications. Middleware provides the reusable capabilities whose qualities are critical to simplifying the interoperation and coordination among interconnected application systems. Existing middleware platforms vary in terms of programming models, networking abstractions and value-added services enforcing non-functional properties. Middleware can be categorized as below [2, 4, 6].
Transactional middleware

Transactions are contracts that guarantee a consistent system state transition and are used in various distributed application domains. A transactional middleware controls transaction applications and performs business logic/rules computations and database updates. It provides integrity by ensuring that transactions do not get lost or damaged offering an interface for running transactions among distributed components.

Examples mainly include TP Monitor like BEA Tuxedo.

Object-Oriented Middleware

Object-oriented middleware (OOM) extends the object-oriented programming paradigm to distributed systems.

Developed out of a need to extend the Object-Oriented programming paradigm to distributed systems, the middleware typically consists of a mechanism to allow methods to be invoked on remote objects, plus services to support the naming and location of objects in a system-wide manner.

Examples of OOM are Java RMI/JINI and CORBA.

Component-Based Middleware

Component-Based Middleware (CBM) extends the object-oriented middleware by integrating non-functional properties such as reliability, scalability, and security together in the component runtime environment.

CBM is mainly known as application server. Application servers can handle all of the application logic and connectivity found in client-server applications. Many application servers also offer features such as transaction management, clustering and failover, and load balancing; nearly all offer ODBC support. The main benefit of an application server is the ease of application development since applications are assembled from building blocks provided by the application server instead of being coded from scratch.

Examples of CBM are CORBA Component Model(CCM), J2EE application server and .NET Framework.

Message-Oriented Middleware

Message-oriented middleware (MOM) is the natural extension of the packet paradigm of communications prevalent in the lower layers of the OSI network model.

MOM supports the exchange of general-purpose messages in a distributed application environment. There is a wide spectrum of MOM encompassing data exchange and request/reply style interaction by publishing messages and/or message queuing in a synchronous and asynchronous (connectionless) manner. The MOM system ensures message delivery by using reliable (and persistent) queues or reliable multicast and provides directory, security, and administrative services.

MOM has a larger share of the market than OOM and has being used for database access in large business applications. Examples of MOM are IBM's MQseries (reliable, MOM service) and TIB/Rendezvous (topic-based, publish-subscribe).
**Service-oriented middleware**

Service-oriented middleware (SOM) enables services to communicate with each other by passing data from one to another, or by coordinating activities among them without knowing the implementation of the underlying supports, including software, programming, or technology.

SOM is built on Service Oriented Architecture (SOA) which is an architectural design pattern that concerns itself with defining loosely-coupled relationships between producers and consumers. In SOA-based environments, SOM provides runtime support for service providers to deploy services on service host and further advertise their presence to the registry, and for service consumers to discover and use services. SOA has been becoming a very active research area over the last few years and actually taking over most middleware technologies.

Examples of SOM include Web service, ESB (Enterprise Service Bus), and so on.

Middleware can be recognized from three vertexes which constructs a triangle: 1) the problems it faces, 2) the effects it achieves, and 3) the route/approach it adopts. For example, the problem CORBA resolved is the communication between remote objects; the effect is that CORBA enables the communication between remote objects without concerns of the programming language, OS platform, communication protocols and interconnects; CORBA offers an interface definition language (IDL) which is used because of the fact that objects may be implemented in any suitable programming language, and an object request broker which is responsible for transparently directing method invocations to the appropriate target object.

**2.2 The evolution model of middleware**

In this paper, we think that there are two kinds of abstraction which control the evolution of middleware: the virtualization of machines and the abstraction of applications. Different abstraction generate different kinds of middleware, but the common goal is to provide layers in the middle to ease the tasks of developers.

Concerning the virtualization of machines, we can think how to abstract the computing resources over the Internet for the higher layer. The resources it considers mainly include computing resources, such as servers, networks and storage. The point is how to conquer the complexity of communication among the resources on the Internet and make them available on demand in a simple view or like a local machine. The complexity comes from the technology species diversity on Internet. We never hope we can build a pantisocracy. In spite of its monotone and dull, and everything in existence is reasonable, in fact people cannot afford to get rid of the existing systems. So whenever a new technology emerges, the communication becomes more complex.

For example, CORBA is mainly to deal with the communication in the distributed environment in which applications are structured into distributed objects that interact via location transparent method invocation. Web services are used to communicate between services which have no concern about object implementation like CORBA. Further, ESB supports message transport among technologies including CORBA and Web services.
On this side we need to consider another point which Douglas C. Schmidt regarded as host infrastructure middleware. It encapsulates and enhances native OS mechanisms to create reusable event de-multiplexing, inter-process communication, concurrency, and synchronization objects [7]. For example Java Virtual machine (JVM) helps programmers eliminate many tedious, error-prone, and non-portable aspects of developing and maintaining application software by encapsulating the peculiarities of particular operating systems.

On the other hand, people are trying to refine the commonalities from applications. It is a natural trend when people are engaging in the development of applications that techniques and tools are devised to enhance the reuse of software systems to help people manage the complexity and heterogeneity of applications. For example, aimed at the transaction processing, Transaction Processing (TP) monitors emerged to monitor transaction programs and manage data that must be left in a consistent state. Like transaction, security service is the common service provided for applications which provides data access for remote users. Application servers are the same which facilitate the serving (running) of other applications. In essence it helps programmers isolate the business logic in their programs from the platform-related code.

Figure 1 gives the evolution model of middleware by enhancing the middleware layers described in [7]. The X-axis represents the trend from the angel of machine virtualization, from host virtual machine like JVM to distributed communication like CORBA, and to distributed virtual machine like distributed operating system (maybe); and the Y-axis represents the trend from the view of application abstraction, from common services like security to domain-specific services like some telecom service.

3 Next generation middleware

As described above, different middleware represents a different programming model related with some specific paradigm. A paradigm is referred to a thought pattern
in any scientific discipline or other epistemological context according to wiki [8]. A software paradigm should be a pattern which maps the real world into the machine world, i.e., giving a solution to describe and compute the problem. Description means that how developers develop the program, which involves the development environment. Computing means that how the machine executes the program, which involves runtime support and management mainly. For middleware, it is involved in the two points. For example, Object-oriented paradigm is a software paradigm which provides OO language, OO development method and OOM to address the mapping.

3.1 Challenges from Internetware

When network applications increase, the Internet is becoming more and more dynamic, open and intractable which are embodied in detail by the following items: true distribution without central control, highly autonomic and indeterminate nodes, open and dynamic connection between nodes, heterogeneity of people, devices and software, individual and flexible usage model, and diversity of networks.

The new generation of software systems should adapt to such an environment. According to Prof. Daniel Menascé[9], the characteristics of the systems would include: highly distributed; component-based (for reusability); Service-oriented architectures (SOA); unattended operation; potentially unfriendly environments; composed of a large number of “replaceable” components discovered at run-time; and running on a multitude of (unknown and heterogeneous) hardware and network platforms.

In order to support such systems to survive and evolve in such complex environment, the Internetware paradigm emphases five capabilities: open structure, dynamic cooperation, on-line evolution, environment sensitivity, and autonomic adaptation. That means, the structure of the software is more dynamic, the software entity becomes more active and autonomous, the collaboration among software entities will be more flexible and diverse, the software evolution will last along with its whole running life, and software will be delivered as a service in "Web As The Platform".

We think middleware has to address the challenges from the following five aspects which would predominate its future evolution.

Autonomous, the middleware should support the software entities being able to exercise all of the necessary functions of power and adaptive to the environment without intervention from any others. It’s the same for itself.

Cooperative, the middleware should provide mechanisms to support cooperation among software entities though different kinds of patterns, for example static or dynamic, connection-oriented or coordination-oriented, and so on.

Polymorphic, middleware should support the software entities to show multiple possible goals which are consistent with each other. Different kinds of result forms should be satisfied according to some principles on the Internet.

Reactive, middleware should be environment-sensitive, and be capable of reacting to the change of the perceived information.
Evolutive, middleware should support entities to evolve according to the application requirement or running environment, which may result in the change of the number of elements, the architecture, and the cooperation style between elements as well. Moreover middleware should evolve itself.

3.2 An evolutionary model of middleware supporting Internetware

On one hand, middleware was being developed along its own road stimulated by network-based applications. On the other hand, middleware was born to support software paradigm, and the evolution of the latter influences the former very much. With the understanding that current software paradigm is evolving towards Internetware, it is easy to get the statement that current middleware is evolving towards Internetware middleware.

Figure 2 gives an evolutionary model of Internetware middleware whose growing trend is dominant by three axes. X-axis represents machine virtualization, Y-axis represents application abstraction, and Z-axis represents environment-driven factors.

From the view of applications, more and more available and necessary commonalities are abstracted as middleware instead of being re-created for each application. Internetware catches the essence of next generation software. It emphasizes on open, dynamic and intractable characteristics, which should be the key problems to be coped with by next generation middleware, i.e., Internetware middleware.

From the view of the machine, virtualization is more and more fabulous. CORBA is a virtual layer which encapsulates the communication detail (abstracting socket–level network programming in terms of high-level network abstractions matching the application computational model). Java virtual machine is a virtual layer which shields the differences among the operating systems. In the future, Internetware middleware should be a virtual layer which enables a global ubiquitous computer. In fact, that is the same goal of gird computing, pervasive computing and service computing.

At the same time, the five aspects including autonomous, cooperative, polymorphic, reactive, and evolutive described in last section greatly influence the development process of middleware. We summarize them as environment-driven factors, because they embody the complexity of the Internet where the key headstream challenges come from. This stimulates the generation of corresponding mechanisms in middleware, for example, agent runtime environment [5], environment description and acquirement mechanism, architecture dynamic evolution mechanism, trustworthiness evaluation and measurement mechanism, and so on.

Every kind of middleware covers or will cover one or several axes, and one or several aspects along each axis. The final fantasy will be a service grid, an infrastructure wrapper over current hardware that software, providing dependable, consistent, pervasive, and inexpensive access for every user who wants it. The routes leading to it are different, but the principles of evolution are the same like Charles Darwin saw in the diversity of species that operated to generate the species: variation, competition and selection.
Figure 2. An evolutionary model of middleware supporting Internetware

4 Conclusion

Historically, mission-critical applications were built directly atop hardware which is rigid, error-prone, and costly over lifecycles. So middleware was born for reuse by way of abstraction. Middleware was generally known as a middle layer located between the operating systems and applications. But it is a quite general concept for cognition which influences its popular acceptance and further growth.

Some researchers proposed their viewpoints. For example, [7] gave an insightful opinion about middleware layers. [6, 10] considered middleware integrating with software engineering, and suggested middleware-based software processes. [11, 12] considered the requirements for middleware from resource management coming from Grid. We give a perspective on middleware from two views, one is from the application abstraction the other is from the machines virtualization (including both of those located on local machine and distributed over Internet). The evolution of middleware no matter from application abstraction view or from machine virtualization view represents and pushes the development of software paradigm.

Middleware research and development has reached a new phase. The new generation middleware is arising with the new arising requirements. The transition poses the following questions which remain open to resolve.

What kind of abstraction will we need in the next step?
Can we build a general model for all the middleware, and customize the necessities to fit the applications when needed?

Can we build customizable, configurable and scalable middleware framework to adapt to the complex network environment?

How are the characteristics, including autonomous, cooperative, polymorphic, reactive, and evolutive, modeled and incarnated in the middleware?

References

Zhou Minghui was born in 1974. She received the PhD degree in computer science from National University of Defense Technology in 2002. She is a associate professor at Peking University. Her research interests include distributed computing, software engineering, etc.

WANG Qianxiang was born in which year. He received the PhD degree in Computer Science from Northwestern Polytechnic University in 1997. He is a professor at Peking University. His research interests include software engineering, Middleware, etc.
Jiao Wenpin born in 1969. He received the PhD degree in Computer Science from Chinese Academy of Sciences in 2000. He is an associate professor at Peking University. His research interests include Software Engineering, Intelligent Software, etc.

Mei Hong was born in 1963. He received the PhD degree in computer science from Shanghai Jiaotong University in 1992. He is a professor and doctoral supervisor at Peking University. His research interests include Software Engineering, Software Reuse, Distributed Object Technology, Software Production Technology, and Programming Language, etc.