

Software Building Blocks

Mobile Agent Based Service Oriented Architecture (ASOA)

Copyright © 2005

Mobile Agent Technologies

Table of Contents

TABLE OF CONTENTS	2
INTRODUCTION	3
MASS PRODUCED SOFTWARE	3
SERVICE ORIENTED ARCHITECTURE (SOA)	3
USING MOBILE AGENTS TO IMPLEMENT SOA	4
<i>Advantages of Using an Agent Based SOA</i>	4
LIMITATIONS OF JAVA'S EJB COMPONENT MODEL	5
TIGHT COUPLING OF COMPONENTS	6
<i>Stubs and Skeletons Are Now a Legacy of the Past</i>	7
SAVING MONEY IN THE DATA CENTER	7
<i>Use Existing Hardware More Effectively</i>	7
<i>More Economical Blade Server Hardware</i>	8
<i>Reducing Human Labor</i>	8
CONCLUSION	9
ABOUT MOBILE AGENT TECHNOLOGIES	9

Introduction

How reusable is your source code? From the very beginning of the software industry, engineers have tried to create a set of tools, and a methodology, that would significantly reduce the amount of time needed to write an application. One recent innovation that has a great deal of attention focused upon it, both in software companies as well as corporate IT groups, is known as a service oriented architecture (SOA). The primary goal of an SOA is to create a number of loosely coupled, reusable services. These "services" could then be dynamically assembled and reassembled, into any number of different applications, based upon ever changing business requirements. If properly implemented, an SOA would enable an organization to develop and deploy enterprise applications much more quickly and at a lower cost than previously possible. This new architecture will provide both IT organizations, as well as software companies, the increased flexibility and agility that their customers have been demanding for a long time. Mobile agents, in conjunction with a peer to peer based execution environment, provide a superior solution for building an SOA, when compared to currently used distributed component object models which are based upon application servers and other types of middleware.

Mass Produced Software

Ever since the initial introduction of the stored program and subroutine in the 1960's, software engineers have been trying to devise new approaches to reuse existing program code. In 1968 a visionary computer scientist by the name of Douglas McIlroy gave a speech at a NATO conference in Germany entitled "Mass Produced Software Components". In his lecture McIlroy described a possible future in which software factories would manufacture reusable components, which would subsequently be used as building blocks within larger systems. Much the same way the electronics industry works. He also stated that in the existing software industry there were no "standard parts, much less catalogues of standard parts" and that software currently was being produced using "backward techniques".

Service Oriented Architecture (SOA)

In the early to mid nineties, there was a gradual shift away from traditional procedural programming languages, such as "C", Cobol and Fortran, to newer ones that were object oriented, such as C++ and Java. Two new facilities provided by these improved languages, were polymorphism and inheritance. These features helped to some extent, to increase code reuse. In time, these object oriented languages gave birth to a distributed component model and servers in which to deploy them. The most popular of these models today are CORBA, DCOM and Enterprise Java Beans (EJB). But even as companies have whole heartedly embraced these technologies as the foundation for building enterprise applications, few if any have yet to reach the much anticipated nirvana of code reuse and componentization that McIlroy envisioned.

There are many different opinions in the industry as to the precise definition of a service oriented architecture. But everyone seems to be in agreement about three essential qualities that this architecture should embody. The first property being a set of loosely coupled components. The second course grained interfaces. The third, support for asynchronous processing. Dr. Hao He in an article entitled "What is a Service-Oriented Architecture?", published on the webservices.xml.com website in September of 2003, provided the following definition of SOA. "SOA is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners."

Using Mobile Agents to Implement SOA

Today there is really only one distributed component model that actually fulfills the broad definition provided above, that of autonomous mobile agents. By their very nature, mobile agents provide a set of easily reusable services. Typical examples might be the calculation of a monthly premium on an insurance policy, determining an employee's overall eligibility for health and welfare benefits, or finding a list of airline flights based on specified parameters for arrival and departure. One or more of these services could be combined to form a cohesive business process, such as binding an insurance policy, automatically making benefit selections for an employee, or booking a vacation. A set of one or more business processes would comprise an application as it is currently known today.

Inherent characteristics of mobile agents allow them to move about freely on a computer network looking for hosts with excess CPU cycles, attaching themselves to these machines, and then beginning execution. The ability of mobile agents to do their work in parallel, across a potentially large number of low cost computers, such as blade servers, provide cost savings that can not be easily realized by today's popular distributed component models. A mobile agent based SOA easily supports the dynamic and rapid assembly of software applications which are extensible, scalable and outperform those created using today's current enterprise information architecture (EIA). Additionally, a broad number of software initiatives within one organization can all leverage the same set of agent based services. This holds true across organizational boundaries as well, such as between business partners, governmental agencies or military branches.

Advantages of Using an Agent Based SOA

By anyone's estimate, enterprise application software is very expensive to develop and maintain. The rule of thumb in the software industry says that only one third of the total cost of a developing an application is borne prior to its initial general availability (GA) release, while two thirds of the costs though are incurred while enhancing the features, improving the underlying infrastructure, and fixing defects over its useful life.

An agent based SOA provides a very easy way to reduce the initial development costs associated with a complex application. An application no longer has to be written from scratch, but rather assembled from existing off the shelf parts. It was this very same business concept, that of assembling products using standard off the shelf parts, which allowed the IBM corporation to quickly respond to the perceived threat from Apple and introduce a personal computer of their own. These reusable components will be stored on a standard web server until they are needed, at which point they are downloaded into an agent execution environment. Besides lowering the cost of an initial general availability product release, these software building blocks can also help to make an application more extensible, and easier to maintain, over its useful life. Since all of the source code for any given service is cleanly encapsulated within a single mobile agent, it is much easier for a developer to find bugs and enhance existing features.

Limitations of Java's EJB Component Model

There are notable limitations in using today's leading distributed component models as a foundation for building service oriented architectures. Taking a look at the EJB model, the component itself is statically bound to the machine upon which it is installed. It lacks the ability to move about on a network and attach itself to an available host processor. The inability of these components to move about freely on a heterogeneous computer network, directly limits this technology's ability to support the dynamic assembly of applications from a collection of available services. Just as important though, the EJB, unlike a mobile agent, does not offer a single service, but rather most commonly exposes a set of very fine grained interfaces. This approach is similar to writing traditional programming libraries and frameworks, except for the fact that the method is remotely accessible. This technical approach causes complex interdependencies to arise between EJB's. Software architects refer to these interdependencies as "surface area", with their desire being to reduce it to a minimum amount. As a result the overriding goal of creating small manageable pieces of functionality, which can be quickly and easily assembled into a robust application, is made much more difficult.

At the implementation level there are other key factors which limit the usefulness of EJB's as an enterprise component model. These limitations are part of the EJB specification itself. The original creators of this specification envisioned a hypothetical world in which programmers had nothing to worry about except coding pure business logic, but they failed to build facilities into the application server, which would compensate for the loss of coding options available.

First of all EJB's can not read from, nor write to, the local file system. Throughout the history of software development, different components within a system have frequently taken advantage of the local file system to exchange needed information, or to store the results of intermediate computations. This sadly limits the EJB only to deployment time configuration. Secondly an EJB can't accept connections on a network socket, which means a server level daemon, such as a web, mail, or FTP server couldn't be implemented using the EJB component model. This could easily be accomplished using mobile agents. Next, from a lower level programming perspective, an EJB can't read or write non-final static fields within a class. This coding technique is most commonly used

as a mechanism to achieve rapid access to data, the static field acting as a shared memory cache. Lastly, and probably the most significant, is that an EJB can not spawn a separate thread of execution. This responsibility has been delegated solely to the J2EE container in which the component is deployed. This limitation prevents the Java developer from taking advantage of time-slicing, a facility found in modern day pre-emptive multitasking operating systems. This allows the CPU to switch back and forth between multiple concurrently running threads, by swapping each one in and out of memory, giving each one its own "slice" of processor time. This last restriction takes away a developer's ability to execute algorithms, or other CPU intensive tasks, at the same time that IO bound operations are taking place. This directly hinders their ability to create robust, high performance, enterprise applications.

Mobile agents, to the contrary, suffer from none of these inherent limitations. They can easily cache the results of intermediate computations temporarily within the file system or within static member variables, whichever is more convenient. A mobile agent, unlike an EJB, can read a local configuration file upon arriving at a host, allowing it to be configured dynamically at runtime, varying its behavior based upon its unique execution environment. It also can open a server socket, allowing it to operate as a process daemon capable of accepting remote connections. From these software building blocks, an HTTP, FTP, email or chat server can be built. Or a sever that will take advantage of tomorrow's new communication protocols. But most importantly, it can spawn additional threads of execution, allowing the software developer to take full advantage of the advanced capabilities which today's modern operating systems make available, including several gigabytes of memory, as well as hyper threading (HT), dual-core chips, and 64 bit microprocessors.

Tight Coupling of Components

Another notable characteristic of Enterprise Java Beans, which also hampers the use of this programming paradigm for implementing service oriented architectures, is the fact that EJB's are oftentimes dependent upon one another. This is known as "tight coupling", meaning the behavior of two components are innately tied together. Since EJB's can't spawn additional threads themselves, and commonly expose fine grained interfaces, they unfortunately are forced to rely upon one other to carry out integral tasks. As previously mentioned, these interdependencies lead to a markedly increased surface area within the subsystem they are a part of. Surface area relates to the set way in which two or more objects interact. The greater the surface area the more expensive it is to maintain an application over its useful life. Increased surface as a result of interdependencies works as an obstacle to achieving the core SOA goal of dynamically assembling applications from a set of basic services. Mobile agents on the other hand have no remote interfaces at, providing a set of course grained services which are asynchronous in nature. As such they are inherently loosely coupled, and capable of being "wired together" almost at will.

Stubs and Skeletons Are Now a Legacy of the Past

Today distributed components are most commonly deployed within an object request broker (ORB), or application server. In order for a consumer of a remote service to utilize the distributed object, helper classes referred to as stubs and skeletons must be created. They must then be compiled, and made available within the execution spaces of both the service provider as well as the consumer. These helper classes marshal the remote request at the consumer side and again un-marshal it at the service provider. This approach mandates that a distributed component offer a set of previously defined, well known public interfaces. This is yet another aspect of tight coupling.

A mobile agent based system eliminates the need entirely for stubs and skeletons. As such, there is no rigid pre-defined contract which must be adhered to by a consumer and service provider. This allows mobile agents to deliver their services in an asynchronous fashion, simply forwarding their "results" back to the consumer through a mechanism that is very similar to a callback. While the service is being performed on behalf of the consumer, that component can also continue to execute if needed. To the contrary, EJB and similar distributed component models deliver their services in a synchronous in nature, meaning that once a call is made to a remote bean program execution stops until a response is sent back. Program execution is essentially blocked within the consumer thread, while it is waiting for the call to return.

Saving Money in the Data Center

Ever since distributed components began to be deployed within CORBA compliant object request brokers (ORB) in the early nineties, high end servers have been the hardware of choice for running this type of software infrastructure. The same holds true today for Enterprise Java Beans which are deployed within application servers. In a production environment these application servers are commonly configured within a cluster, providing failover capability. If there are three machines within the cluster, and one goes down, one third of the processing capability of the system would be lost. If this were to happen during a time of peak demand, the performance of the system could be adversely affected.

Use Existing Hardware More Effectively

Within the data center a cluster of application servers is usually dedicated to running one specific business application. In a large enterprise, a "pilot" application that only has been rolled out to a small handful of clients would most likely run at a very low CPU utilization level. While another application in the same hosting center, which had been in production for several years and supporting several thousand users, may barely be able to meet its required service level agreement (SLA) at times of peak demand. The EJB component model has no facility to dynamically provision hardware and software resources across applications within an enterprise, based upon ever changing organizational needs. As a result it can not efficiently use existing capital investments in computer hardware.

The real beauty of a mobile agent based SOA is that all available computing resources within a data center can be harnessed, and shared across a number of different software applications. The system itself can adjust in real-time to ever changing business requirements and varying workloads, by dynamically assigning mobile agents to available host processors. An application that in the past would have been barely able to meet its SLA during times of peak system demand, if rewritten using mobile agents, could now tap into a much larger reservoir of computing power. IT analysts have begun to call this type of software architecture "application virtualization".

More Economical Blade Server Hardware

Since their introduction just four years ago, Blade servers have already begun to supplant and replace rack mounted and stand-alone servers. There are numerous advantages of using this type of hardware in the data center. First of all, the form factor is three to ten times as dense as traditional servers, allowing more computing power to be put in the same amount of space. Secondly, the servers are housed within a chassis, and as a result, key components such as cooling fans, hard drives, power supplies, and switches can be shared very effectively between a set of blades. Another advantage is that they are easier to install and server management software has been greatly improved, when compared to their predecessors. Adding additional computing power can be as simple as sliding a couple of brand new blades into an existing chassis. Lastly, and maybe the most important, is that they consume less power, so the cost of running large numbers of servers, twenty-four hours a day, seven days a week, in a data center will be less than with previous types of servers.

Mobile agents are perfectly suited to take full advantage of the benefits which blade server hardware has to offer, much more so than existing distributed component models. The agent execution environment is extremely lightweight peer to peer environment, which was specifically designed for high performance. Facilities within the environment can monitor CPU utilization so that machines having excess CPU cycles available can be identified, and agents dispatched to run on them. As with all of the other great waves which we have experienced in the short history of computing, it will be the combination of new blade server hardware used in conjunction with mobile agents, that will be used to create the next generation of enterprise applications. These new applications will undoubtedly give the organizations using them a true competitive edge in a global marketplace.

Reducing Human Labor

The impact to operational tasks, within the data center, as the result of a move to an agent based SOA, should be mentioned. Operators will no longer need to deploy and configure distributed components. The agents will be able to roam freely on the network and attach themselves to any available computing host which provides a suitable execution environment. Like their early predecessor the Java applet, mobile agents will be stored within the file system of a standard web server, and downloaded via the HTTP protocol at such point in time that an application needs to use it.

Conclusion

Today, a mobile agent is the closest approximation our industry has to a truly reusable software building block. As a component model it overcomes all of the limitations which were imposed on distributed Java components as a result of the EJB specification. From these loosely coupled, coarse grained, asynchronous services, applications will be dynamically assembled, based upon ever changing business priorities. Systems built upon this new architecture will be far more extensible, reliable, and scaleable than traditional applications built using today's most popular distributed component models. Using an agent based SOA in conjunction with low cost blade servers, will provide today's smart organizations with an extremely effective approach to lowering their total cost of ownership for enterprise application software.

About Mobile Agent Technologies

Mobile Agent Technologies is a leading provider of development tools, and software infrastructure that enables organizations to build agent based service oriented architectures. Our flagship product, called AgentOS, is a mobile agent environment that can help your organization to significantly lower its annual cost for enterprise application development. Future product releases of AgentOS will feature our revolutionary patent pending process for automatic thread migration. This technology allows a running mobile agent, to automatically migrate from one execution environment to another on a heterogeneous network, including a complete execution stack trace, based on runtime environmental conditions. In addition we also offer consulting services related to agent based service oriented architectures, business rules engines and artificial intelligence. To learn more about our products and services please contact us at 201-923-1006, or visit our website www.agentos.net.