Features for Continuous Business Services throughout Application Lifecycles

● Shinya Echigo

(Manuscript received January 11, 2007)

One way to achieve a high-reliability, high-performance system is to implement non-stop operation. This is very important because business chances are lost during system downtime. Until now, maintenance and enhancements of business applications could only be done by stopping an entire system, so these activities are obstacles to non-stop operation. Therefore, a key problem for non-stop system implementation is how business applications can be upgraded without stopping an entire system. This paper describes a dynamic application replacement technology for overcoming this problem.

1. Introduction

Mission-critical systems need high reliability and high performance to provide end users with stable business operations.

Many mission-critical systems are expected to operate without stopping. However, this is difficult to achieve because many of the hardware and software components of a system cannot work continuously. Usually, system administrators prepare maintenance periods during which systems are shut down. However, even if these maintenance periods are short, system shutdowns should be avoided because they lead to lost business opportunities.

Fujitsu's Interstage middleware provides solutions for implementing non-stop (24-hour/7-day) operation. It incorporates Fujitsu's long-term experience in supporting many mission-critical systems in Japan.

This paper describes approaches and technologies for achieving non-stop system operation even when tasks such as applying application program patches, resolving problems, and upgrading services are performed. These are core technologies that support a non-stop system, for example, by enabling non-stop session recovery. This paper also describes some applications of these core technologies.

2. Considerations when implementing a non-stop system

Non-stop operation is essential for constructing highly stable systems such as mission-critical systems that cannot be shut down without causing a huge business disadvantage. Therefore, the shutdown time of such systems must be minimized, even during system maintenance.

Figure 1 shows the following three major obstacles to achieving non-stop operation:

1) Hardware and software problems

When unpredictable problems occur with hardware and software, the problems should be resolved immediately. When the hardware has a problem, it must be replaced. When the software has a problem, the system administrator must classify the cause of the problem, and if there

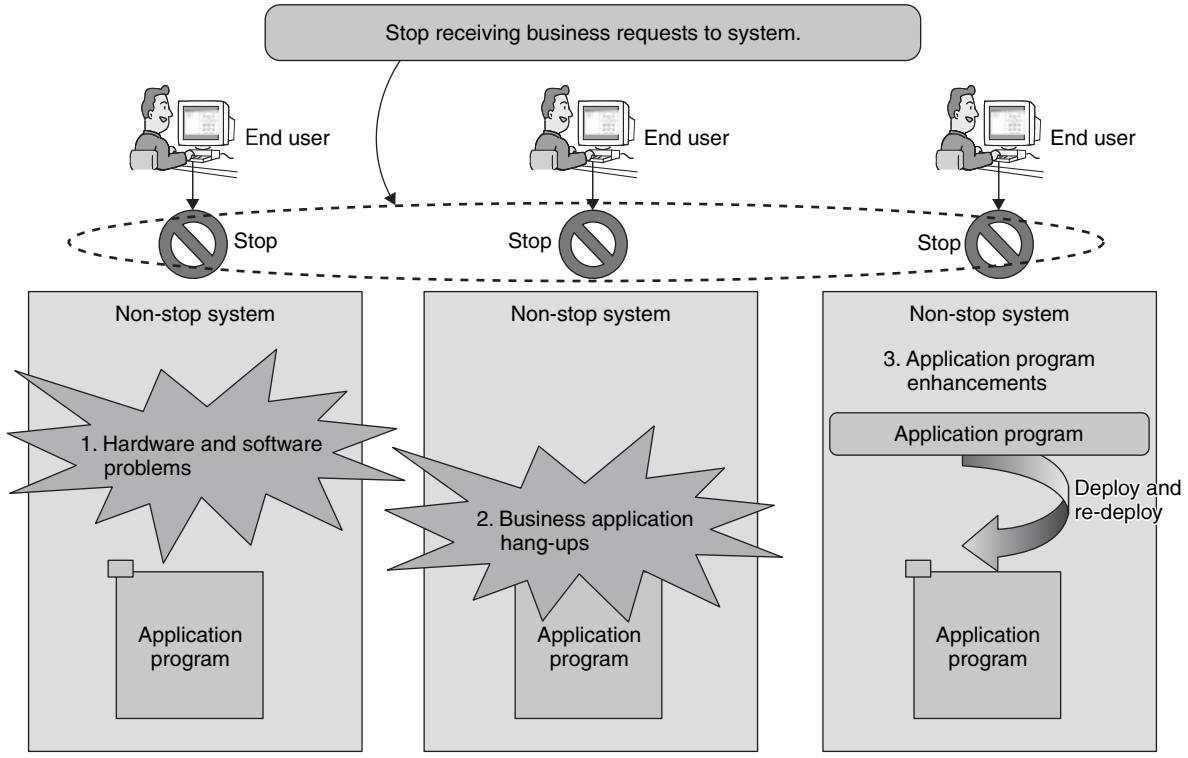

Figure 1
Obstacles to non-stop operation.

is a problem in the middleware, a patch must be applied. However, until these actions are completed, the system cannot respond to business requests.

2) Business application hang-ups

Business application programs hang up due to insufficient resources, program bugs, and other causes. When a hang-up occurs, the system cannot respond to business requests and the business applications' programs must be restarted after the cause of the hang-up has been cleared.

3) Application program enhancements

When application developers notice a bug or add a new service in an application program that implements business logic, the application program will be changed. In these cases, the application program may need to be stopped.

Fujitsu's Interstage middleware provides software solutions for realizing non-stop systems.

For example, regarding the second obstacle of business application hang-ups, Interstage provides solutions for anticipating troubles by way of alarm notifications and solutions for reducing the down time of business applications by auto-restarting their programs.

Similarly, regarding the third obstacle of application program enhancements, Interstage provides the solutions introduced in the next section.

This paper also describes how these solutions can be applied to overcome the first obstacle of hardware and software problems.

3. Solutions for application program modifications and enhancements

As mentioned in Section 2, application program modifications and enhancements are issues that must be resolved when implementing

a non-stop system.

There are two type of solutions for making application program modifications and enhancements without stopping a system:

- Solutions using load balancers
- Solutions using Interstage functions

Both types have merits and demerits and the system administrator must choose the best solution for the system architecture and requirements. **Table 1** summarizes the characteristics of five solutions, and the following sections explain them in detail.

3.1 Solutions using load balancers

In large-scale systems, load balancers such as Fujitsu's IPCOM[1)] network server are usually installed. Load balancers can improve an entire system's throughput by managing several servers in parallel.

By using load balancers, a system administrator can change the load balancing policy of a system's servers to temporarily stop the dispatch of business requests to a server so it can be safely maintained. During the maintenance, the remaining servers can continue processing business requests so the system does not stop (**Figure 2**).

These solutions can be applied for hardware and software problems (e.g., hard-disk problems and middleware program bugs) because they can isolate problem parts. With these solutions, the system administrator separates the target server, safely fixes the hardware and software, and then builds the server back into the system.

However, these solutions are only applicable to systems that employ load balancers, and they require the system administrator to perform tasks such as changing the policy of the load balancers. These disadvantages make these solutions difficult to apply in general.

Moreover, when maintenance is performed using these solutions, the number of servers is reduced, so the system administrator must ensure that the remaining servers have sufficient capacity to cope with the extra load.

3.2 Solutions using Interstage functions

Interstage Application Server, which is the foundation product of Interstage, provides three functions for non-stop application maintenance that are similar to those of load balancers:

1) The HotDeploy function
2) The Class Auto-reload function
3) A function for active changing of server applications

3.2.1 HotDeploy function

Interstage Application Server provides the Interstage Java Server (hereafter called the IJServer), which is a platform for executing Java 2 Platform, Enterprise Edition[2)] (hereafter called J2EE) applications. The IJServer provides several functions for achieving a non-stop system. One of them, the HotDeploy function, allows J2EE applications to be dynamically added (deployed), changed (redeployed), and deleted

Table 1
Summary of five solutions for application program enhancement.

Solution	Target	Situation
Using a load balancing product	Any application	Any: From server hardware problems to small modifications to application programs
HotDeploy function	J2EE application only	Large modification and enhancement at service level
Class Auto-reload function	J2EE application only	Small modifications
Active changing of a server application	Transaction application only	For transaction applications
Coordination with Session Recovery function	J2EE application only	No effect on continuous business processing for end users

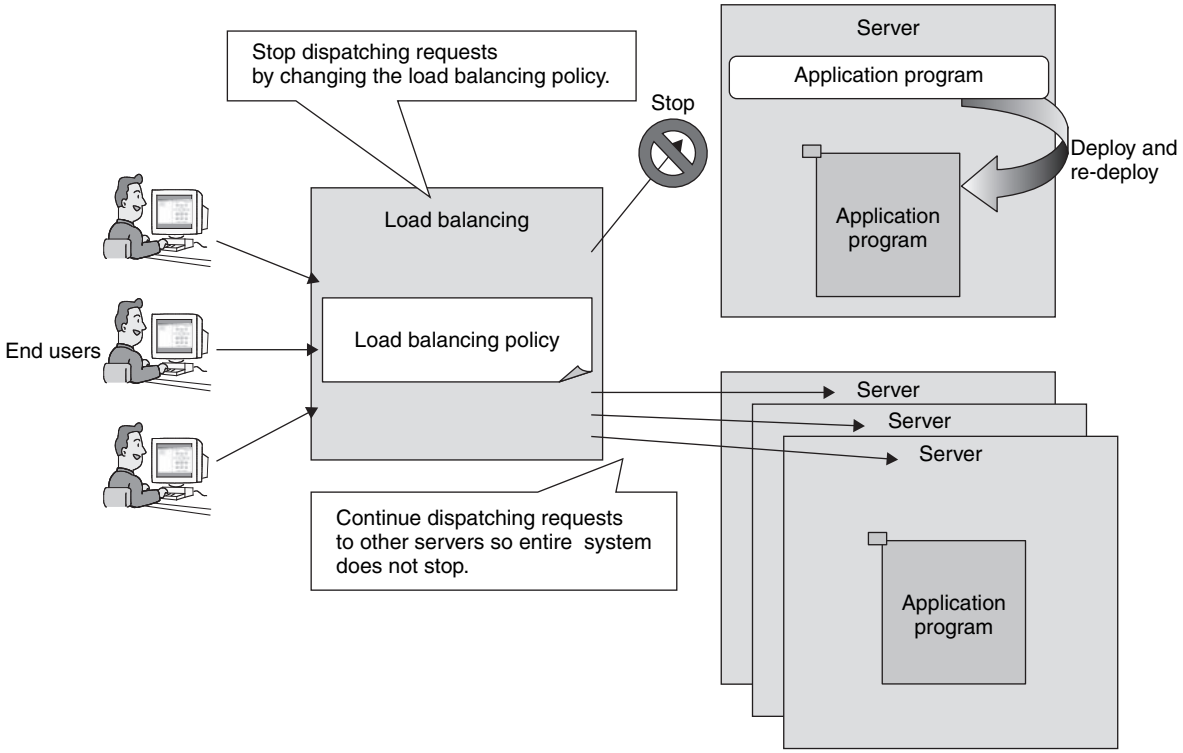

Figure 2
Solution using load balancing product.

(undeployed) without stopping the IJServer.

The HotDeploy function deploys, redeploys, and undeploys J2EE applications by using files with ear, war, and jar extensions. These are J2EE archive files that contain the class files and configuration files (called deployment descriptors) of J2EE applications and are used as units of J2EE applications to be delivered. Because these archive files (hereafter called application modules) are created in units of business services, the HotDeploy function allows business services to be dynamically added, changed, and deleted.

The HotDeploy function is achieved using the Java class loader mechanism. By using separate class loaders, several generations (before modification and after modification) of the same J2EE application implementation can be managed on the same process. The business requests received after executing the HotDeploy function are dispatched to the new J2EE application implementation as shown in **Figure 3**. Old implementations in memory are automatically discarded by the Java garbage collection function.

The HotDeploy function is designed to minimize the losses incurred when reception of business requests is temporarily stopped. For example, to reduce the usage of system resources, several ear files are usually deployed on a single IJServer to process several different services. However, the effect of the HotDeploy function is limited to the target application module, so the other J2EE applications continue processing business requests without interruption. Additionally, the HotDeploy function prevents business requests from being lost as follows:

1) Reception of new business requests is temporarily stopped while the HotDeploy function is executed (① and ② in Figure 3).
2) Before replacing the new J2EE application implementations, the IJServer waits until all the received requests have been

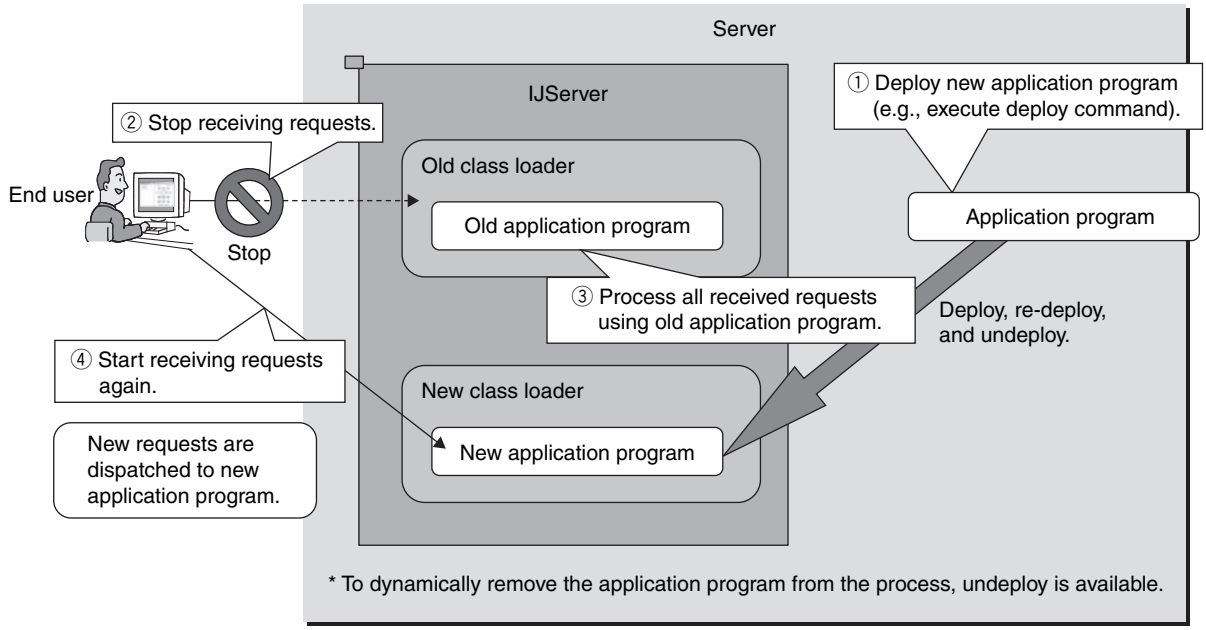

Figure 3
HotDeploy function.

completely processed (③).

3) Reception of new business requests is restarted (④)

An application program developer can manage the timing of the HotDeploy function. For example, a simple batch command can be created that executes the HotDeploy function at midnight. This allows scheduled service modifications.

3.2.2 Class Auto-reload function

The Class Auto-reload function is another of the functions of the IJServer platform for J2EE applications. This function is used when replacing class files (executable files on the Java Virtual Machine) of J2EE application implementations. The IJServer detects the replacement of class files automatically. When this function is used, replacement becomes easier because the application program developer does not need to recreate the ear files or execute commands and only needs to copy the new files to a suitable directory. Therefore, this function is more effective when a modification is limited to just a few class files, which is common during the application development phase.

Like the HotDeploy function, the Class Auto-reload function is implemented based on the Java class loader mechanism. The application program developer copies the class files to the server machine (① in **Figure 4**). Then, the IJServer periodically checks if any class files have been replaced (②) and manages the information of the class files so the replacements are automatically detected when the HotDeploy function needs to execute a command. When a replacement is detected, the class loaders are automatically replaced after stopping the acceptance of requests (③). After the class loaders have been replaced, reception of new requests is restarted (④).

3.2.3 Function for actively changing a server application

There is one more function for dynamically performing application program enhancements. In addition to the J2EE platform, Interstage Application Server also supports platforms on

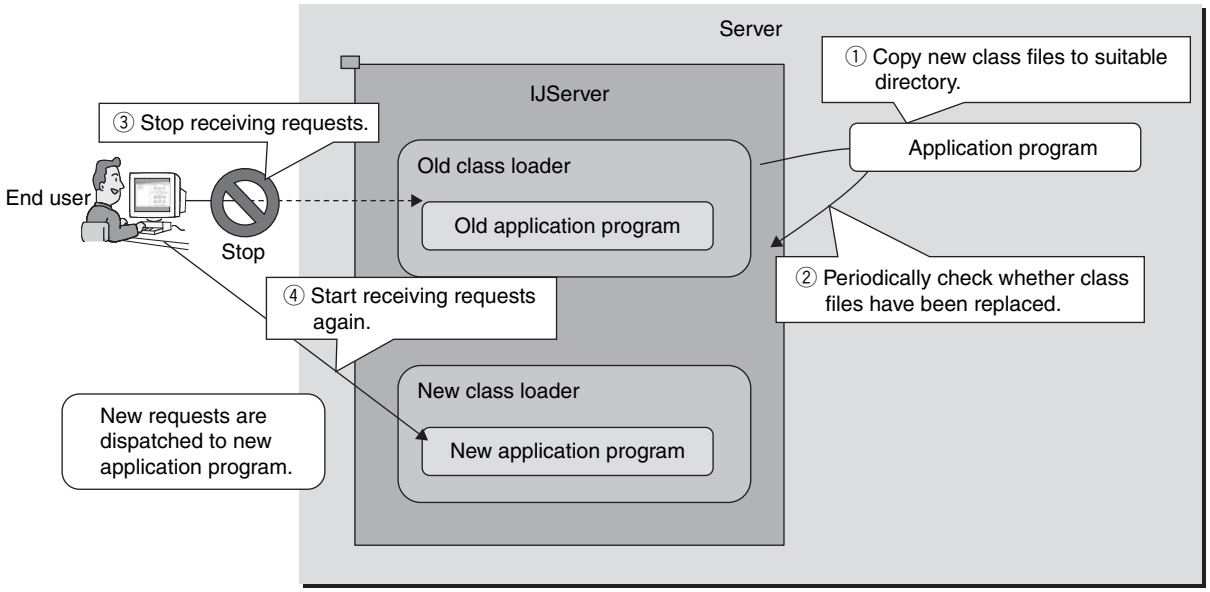

Figure 4
Class Auto-reload function.

which server application programs written in C and COBOL are managed. In Interstage Application Server, such application programs are called transaction applications. Interstage Application Server introduces the WorkUnit, which is a function for managing several application programs as a unit of business services in a one-shot, synchronous manner. Transaction applications are one of the application programs that WorkUnits can manage.

Active changing of server applications enables application programs and the operational environment definitions (WorkUnit definitions) of transaction applications to be dynamically changed.

This function uses a request queue, which is also provided by Interstage Application Server. The business requests from the end users are queued before they are dispatched to the appropriate application process. During active changing of server applications, reception of requests from the queue is temporarily stopped.

3.3 More stable solutions for dynamic application program modifications and enhancements

Interstage Application Server also provides a Session Recovery function for J2EE applications. This function makes dynamic modifications and enhancements of application programs more stable.

The Session Recovery function saves Servlet session information, which is conversational information between a Web browser and Web application, on another server called the session registry server. The Servlet session information is then carried over to the other IJServer processes to continue business processing, and the end users of the Web application are unaware of the carry over. The Session Recovery function therefore can be used to prepare for problems with the IJServer such as, for example:

1) An IJServer process or server machine going down due to an unpredictable cause, or
2) An IJServer process being stopped for maintenance.

The Session Recovery function is achieved

by using a session registry server that stores the Java objects of the Servlet session information and facilitates their transfer between IJServers. If an IJServer receives a business request from an end user but does not have the corresponding session information of the request, the IJServer tries to recover the session information from the session registry server. If successful, the IJServer carries over the session and continues processing the business request.

As shown in **Figure 5**, by combining the Session Recovery function with load balancers, dynamic application program modifications and enhancements can be achieved with the IJServer, and processing of business requests can be continued without discarding the sessions created before the application programs are replaced. The Session Recovery function stores sessions to the session registry server (① in Figure 5). When a server needs to be maintained, the system administrator stops dispatching business requests to the server by changing the load balancing policy (② and ③). Even if the IJServer is stopped temporarily to replace an application program, another IJServer process running on another server machine can carry over the created session and continue processing the business request (④). The end user is unaware of the application replacement (⑤).

Furthermore, even if the Session Recovery function is used only on a single server, the effects of an IJServer shutdown are minimized. The session registry server keeps a backup of the created session information before the IJServer is stopped. When the IJServer is restarted and receives a request from an end user that is

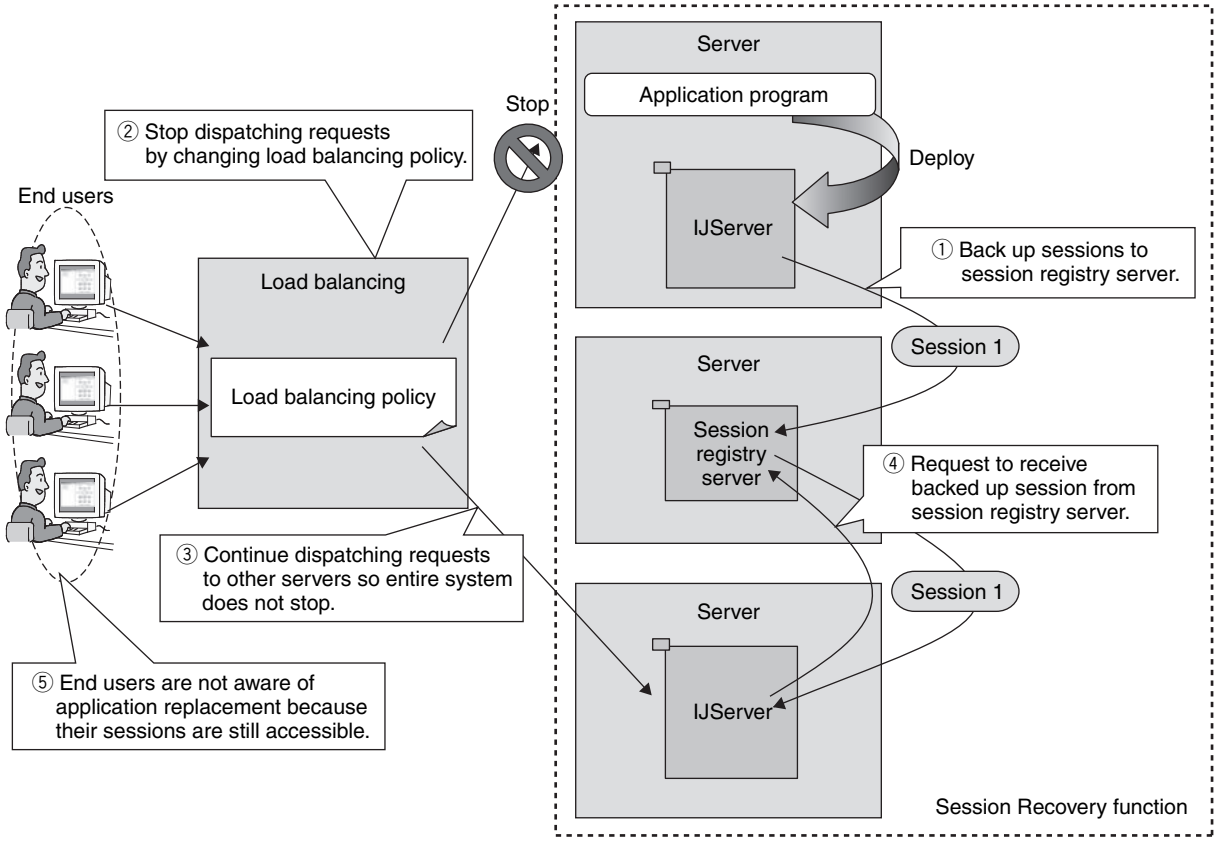

Figure 5
Coordination with Session Recovery function.

related to a previous session, the IJServer recovers the session information from the session registry server.

However, for Interstage products, the use of other solutions such as the HotDeploy function is recommended in this case because the Session Recovery function requires a longer application stop time than other solutions. To reduce losses for the end users, Interstage Application Server should support a request queuing mechanism to avoid the need to stop reception of new requests during shutdown. Such a mechanism could also be made available for dynamic application program modifications and enhancements using the HotDeploy and Auto-reload functions. Developing such a mechanism is one of our future plans.

4. Future plans for dynamic application program modifications and enhancements

To make Interstage Application Server's dynamic application program modifications and enhancements more useful and reliable, it will have to be improved in three ways:

1) When business services are added using the HotDeploy function, the entire system configuration should be checked to see if tuning is necessary. Therefore, the new function must be designed to simultaneously tune the system configurations of the Web servers and the database servers that cooperate with J2EE applications.
2) The Class Auto-reload function detects the replacement of class files; therefore, the replacement timing is managed by Interstage Application Server. However, we must provide a command that enables application program developers to specify the replacement timing. By using such a command, developers could synchronize an entire system by using the Class Auto-reload function to replace J2EE application implementations.
3) The HotDeploy function and Class Auto-reload function temporarily stop reception of business requests. During this time, an error is returned to the clients. If a queuing mechanism such as a transaction application is introduced, the requests are queued so the clients do not receive an error.

5. Conclusion

Interstage Application Server provides three software-based solutions for performing dynamic application program modifications and enhancements, which are essential for achieving non-stop system operation. This paper described how the Session Recovery function of Interstage Application Server enables more stable dynamic application program modifications and enhancements. System administrators can select functions that provide the best solution for the robustness of their system by considering the characteristics of the functions, the system architecture, and the system requirements.

Interstage Application Server provides robust, high-availability technologies for business application programs running on mission-critical systems. By using Interstage Application Server, customers can greatly improve the reliability of the services they provide.

References

1) Fujitsu: Magazine FUJITSU 2006-7 (VOL.57, NO.4).
http://jp.fujitsu.com/about/magazine/backnumber/vol57-4e.html
2) Sun Microsystems: JSR 58: Java™ 2 Platform, Enterprise Edition 1.3 Specification.
http://jcp.org/en/jsr/detail?id=58

Shinya Echigo, *Fujitsu Ltd.*
Mr. Echigo received the B.S. and M.S. degrees in Particle Physics from Kobe University, Kobe, Japan in 1996 and 1998, respectively. He joined Fujitsu Ltd., Kawasaki, Japan in 1998, where he has been engaged in development of middleware software.