# Enterprise J2EE Application Management

Production Monitoring and Performance Management for
BEA WebLogic™ Server, Portal & Integration Applications

ACSERA

## The Agile Enterprise Adopts the J2EE Application Platform Suite

To increase shareholder value, agile companies need to keep their businesses growing and quickly responding to ever-changing market demands. One way companies can increase their revenues is by creating new products and services. These products and services are collections of business units of work such as automating purchase order processing, checking customers' credit, or fulfilling an order. The business units of work may provide customers with self-service capabilities to open new bank accounts, submit a loan application, or to check order status. They may allow employees to enter expense reports, view their paychecks, or efficiently access company information.

These new business units of work require new custom enterprise applications. The common thread in all of these applications is that they need to tie together disparate existing or legacy systems, model critical business processes, and enable remote user access. Today, these enterprise applications are built as service oriented architectures as web portals, business processes, and web services. This provides user, data, and process integration.

Over the last three years, companies have shifted away from just using Web Application Servers to using an Application Platform Suite such as BEA's WebLogic Platform 8.1. WebLogic Platform consists of WebLogic Server (WLS), WebLogic Portal (WLP) and WebLogic Integration (WLI). IT organizations take advantage of this new platform technology to reduce their software development costs and deliver new business applications faster.

## Service Level Agreements

These applications are business-critical and the LOB may even establish Service Level Agreements (SLA) on these business units of work both externally and internally for their partners, customers and employees. Therefore, it is vital to monitor and manage the performance of these systems to ensure high Quality of Service (QoS) to the business so they may meet their SLA. A typical agreement might say that 98% of all wire transfers should take less than 1 minute to execute. Another could specify that an application must be capable of processing thirty thousand dental claims per day. A third agreement could require that end users must experience less than a two second response time from a self-service website.

IT Operation departments need to ensure QoS for the enterprise applications in production; when there are performance problems, the cause needs to be identified and resolved quickly. In many cases system downtime represents loss of revenue, but degraded system performance may also result in revenue loss or decreased customer satisfaction.

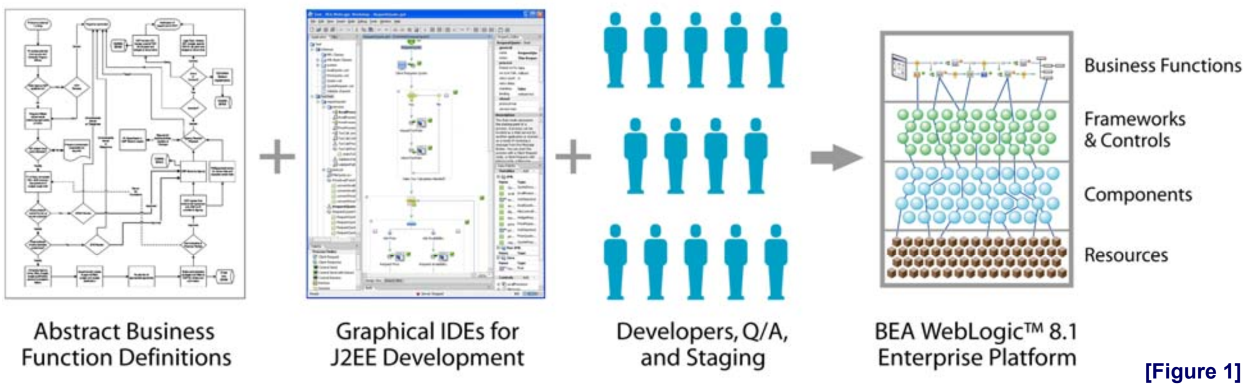
Traditionally, an IT operations department would monitor the health and performance at the system level rather than monitor business units of work. Enterprise Systems Management solutions such as HP OpenView Operations, IBM Tivoli, or BMC Patrol assist with monitoring the performance of critical elements such as the portal, network, operating systems, storage systems, database, load balancing and general system health. But they do not provide visibility into the business processes of a WebLogic Platform production application. This requires a different solution; a monitoring system that understands the business units of work the enterprise application performs. Furthermore the system must understand the individual components that comprise the units of work to be monitored in order to meet the desired SLAs. Monitoring of the JVM memory heap or Java object or method level response times does not result in the achievement of SLA objectives. Therefore, a monitoring solution should watch how many orders were filled, how many loans were processed, or how many accounts were opened. It should monitor and manage the performance of these applications at the business level, not the lowest implementation-dependent level. When there is a performance problem, operations needs to be able to look at all components that are being monitored as described above to quickly determine where the SLA is being violated. There could be many different causes from the network, to the systems involved. However if operations determines that the SLA is being violated because of some condition in the enterprise application, the solution should provide the ability to drill down from the business process to the process nodes to the components that support it to diagnose any bottlenecks or failures.

## New J2EE Platforms – A Greater Level of Power and Complexity

The J2EE Platform applications may consist of hundreds of business units of work; new enterprise applications are being deployed monthly; patches and upgrades may be applied monthly to existing applications. With all this change, production support teams are being inundated with more dynamically changing systems than ever before that they will need to monitor and manage. To keep these applications under control, they will need to understand the system's headroom or capacity, how the system will respond under different loads, and how to identify bottlenecks in the business processes.

In the past, developers wrote enterprise apps using teams of expert Java 2, Enterprise Edition (J2EE) programmers on top of technologies such as WebLogic Server (WLS). The development process could take up to a year. Now using a Service Oriented Architecture such as BEA WebLogic Platform, companies are developing and deploying these business-critical enterprise applications in months. Software developers can create extremely complex enterprise applications that integrate existing information systems using WebLogic Workshop. WebLogic Workshop hides the complexity of low-level J2EE programming thereby enhancing developer productivity and reducing the dependency on J2EE experts in your organization. WebLogic Platform applications are composed of thousands of very low-level components that work together to perform a business function.

Due to its complexity, J2EE has been the realm of expert programmers, leaving many lesser experienced software developers out of the picture. BEA solved this problem by creating WebLogic Platform 8.1 as an abstraction layer on top of J2EE. This opened the application development space to a wider audience. In this new paradigm, system programmers construct custom WebLogic Java Controls that provide such capabilities as data access, legacy system integration, or data computation. These controls encapsulate and abstract low-level J2EE components, so that they may be used as a library of application building blocks. These J2EE components include Enterprise Java Beans (EJB), Java Servlets, Java ServerPages (JSP), JMS queues, JDBC connection pools and JDBC drivers to name a few.


Business Functions
Frameworks & Controls
Components
Resources
Abstract Business Function Definitions
Graphical IDEs for J2EE Development
Developers, Q/A, and Staging
BEA WebLogic™ 8.1 Enterprise Platform

**[Figure 1]**

Application programmers implement business requirements by designing Java Processes, Portals, Pageflows, and Java Web Services by assembling the Java Controls using WebLogic Workshop. [Figure 1] When the applications are deployed on WebLogic Platform, they are essentially "configuring" the new application's business units of work in a runtime framework provided by BEA. The framework architecture is quite complicated and encapsulates many best practices of J2EE design. The beauty is that the developer need not worry about the framework, just the application business logic. The developer's code is compiled with the framework code to create the runtime application. The power of this approach is that business logic may be efficiently modified and redeployed repeatedly over time. Application developers do not have to change low level source code, they simply drag-and-drop new functionality or edit properties. Below the covers, the WebLogic Runtime framework handles all of the heavy lifting and the application is translated into pure J2EE code. Traditional monitoring applications have taken the approach of monitoring J2EE components. This level of focus does not equate to monitoring the business units of work as these components do not represent business functions—they are at the wrong level of abstraction.

## New Solution to Harness Complexity

A solution that monitors and manages the performance of business units of work will first need to identify them in a deployed environment without requiring the business applications to be taken off line. Therefore, a monitoring solution will need to detect all of the enterprise applications on the production clustered servers in a WebLogic domain. There could be an arbitrary number of applications in production and they will grow over time as usage increases and as the business deploys new systems to respond to market demand. This means that a WebLogic Platform monitoring solution will have to understand the architectural frameworks that support the Java Processes, Portal, Pageflows, and Java Web Services. Additionally, it will need to discover and synthesize the relationships between the Java Controls as orchestrated by these frameworks. The solution will also need to understand the WebLogic Java Controls framework and which low-level J2EE components are encapsulated in which Java Controls.

By identifying the Processes, Portals, Pageflows, Struts Modules and Java Web Services, we are finally at the business units of work, but they may change at anytime. Because of this, the monitoring solution will need to be able to detect and track the changes to the production applications in real time. If there is a performance problem in production, the solution should provide a means of doing root-cause analysis. Time is critical when resolving issues with these types of applications; it is too expensive to move the application back to a staging environment to determine where the failures occur. To diagnose problems in real time, production support will need to determine which process is having an issue, which Java Controls are used by the offending process, and which Java method is causing the error or bottleneck. Therefore, the monitoring solution will need a user interface that provides not only a monitoring view for Operations teams, but also a diagnostics view to the Platform Engineering teams for problem resolution. Since operational security can also be of concern, this user interface should provide complete role based access.

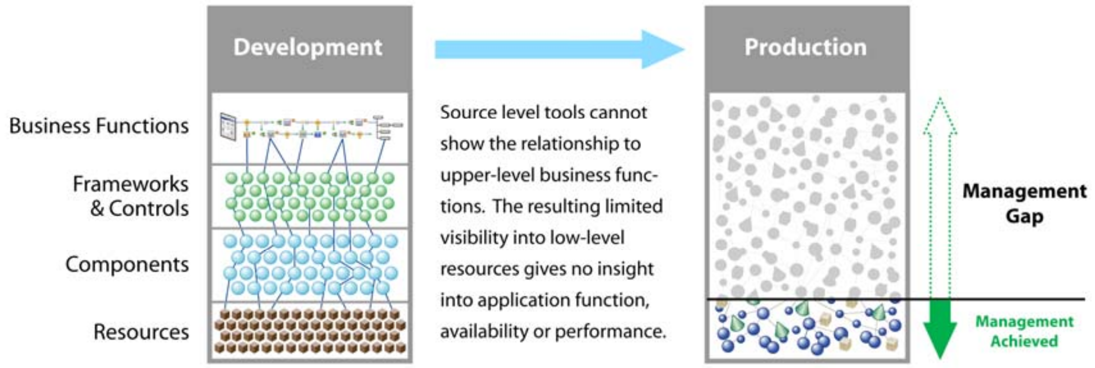
While the monitoring of the business process level SLAs is critical, a SOA also relies on a set of common services. This new class of monitoring application should have the ability to set Service Level Objectives (SLO) on any metric associated with the business units of work all the way down to the supporting J2EE components. When the SLO events are triggered, the monitoring solution should be capable of taking different actions: sending out email alerts, logging a message, running a script or send an alarm to an Enterprise Systems Management application. Because this system has to run in a production environment, it will need to have an extremely low impact on the performance of the deployed WebLogic Platform applications. There should also be a means of controlling the amount of data collected and published from the monitored servers. The monitoring solution should work across firewalls and provide secure connections between the user interface, managed servers, and the processes that collect the data. It should support high availability management configurations and be able to fail over to backup data collection servers. Finally, the system should have the ability to be upgraded in production with zero system downtime.

Some companies have tried to use J2EE monitoring tools for performance management of their WebLogic Platform applications in products and have found many problems with this approach. These tools rely on software developers manually adding instrumentation to the application's Java byte-code before it can be monitored. If instrumentation is added to all of the Java code, the monitoring tool drastically affects the application's performance by generating millions of metrics. To be effective, developers need to pick only the "right" code for instrumentation, this can take up to two weeks of time as they must first understand the application before they even start the manual process. Next, they have to create a custom dashboard for each application. This takes more valuable engineering time as you need to map the metrics to screen components. This manual instrumentation is effectively new code development, and as such needs to be tested for coverage. The monitoring will be only as good as that programmers understanding of the deployed system. When the application or a service that the application depends on changes, the process must be repeated, and the same time expense is incurred before the application is ready to be monitored.

But, the biggest problem with using J2EE monitoring tools is the type and volume of metrics they collect. These tools do not meet the requirement for monitoring business units of work in WebLogic Platform applications. They are good for telling a software developer how many times a Java method was called on which object, but the metrics are far too low-level for system administrators. For this approach to work, it would require the production support team be fluent Java programmers and familiar with the design of the application source code. Otherwise, this would require software developers to be engaged in monitoring the application in production. Clearly, either way this is a very costly endeavor.

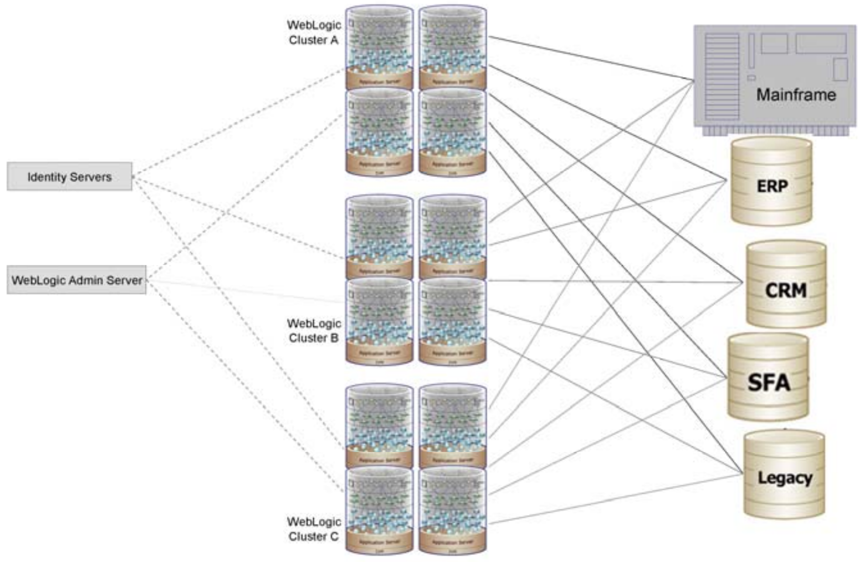
## Current Solutions Provide Low-Level Metrics without Hierarchy and Context

The main problem with J2EE level monitoring tools is that they do not show you any metric hierarchy. [Figure 2] It is important for supporting J2EE component metrics to be grouped with the Java Controls that use them. One J2EE component could be called in many different ways so it is critical to understand the context in which the component was utilized.


Development
Production
Business Functions
Frameworks & Controls
Components
Resources
Source level tools cannot show the relationship to upper-level business functions. The resulting limited visibility into low-level resources gives no insight into application function, availability or performance.
Management Gap
Management Achieved

**[Figure 2]**

Additionally, these code monitoring tools cannot discern user code from the code generated by the WebLogic framework; they expose the entire framework code mixed in with the application code. Administrators should not be required to understand how the framework was implemented in order to monitor the system. Finally, these tools cannot deal with asynchronous, loosely-coupled, distributed service-based systems. [Figure 3]

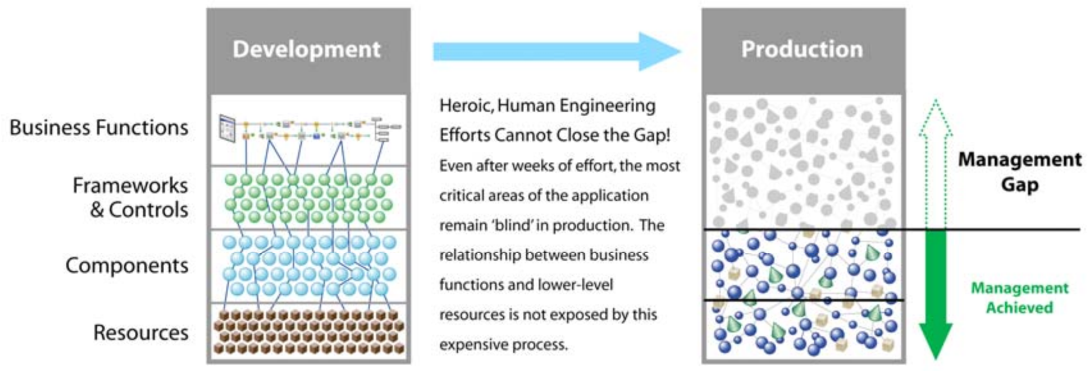


Asynchronous, Loosely-Coupled, Distributed Service-Based Deployment Architecture

**[Figure 3]**

In order to close the management gap, operations experts now engage in significant pre-deployment tooling projects. They are using a combination of Java profilers, JMX tools, code-level monitors and staff to make their applications manageable. A typical project involves people from the development team, people from operations, someone from Q/A and several outside consultants.

In their attempts to manage an application, these teams try to create a mapping of upper-level functions with low-level components and resources. [Figure 4] People work around-the-clock to get their applications ready for production. They bring in experts to analyze programs and hand instrument the system in as many places as they think are needed. Companies are spending many weeks on these efforts and the price tag often exceeds a quarter of a million dollars (and climbing). More important, they are substantially increasing risk by altering software that has already passed Q/A.

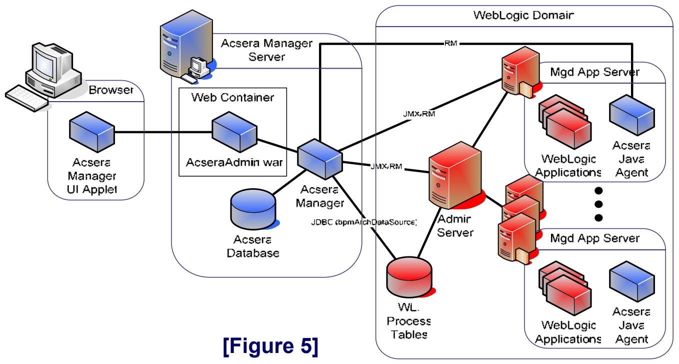

Development
Production
Business Functions
Frameworks & Controls
Components
Resources
Heroic, Human Engineering Efforts Cannot Close the Gap!
Even after weeks of effort, the most critical areas of the application remain 'blind' in production. The relationship between business functions and lower-level resources is not exposed by this expensive process.
Management Gap
Management Achieved

[Figure 4]

## Acsera Manager

Many companies have realized this functional gap in monitoring and managing WebLogic Platform applications with the current toolset and needed a way to overcome this lack of visibility into their production systems. Acsera provides the solution to this problem. Acsera Manager was designed to meet the requirements of monitoring Application Platform Suite-based enterprise applications in production at the proper level of abstraction: business units of work. Developers write processes, Acsera monitors processes. The same is true for the Pageflow, Portal, Portlet, Web Services and Struts frameworks.

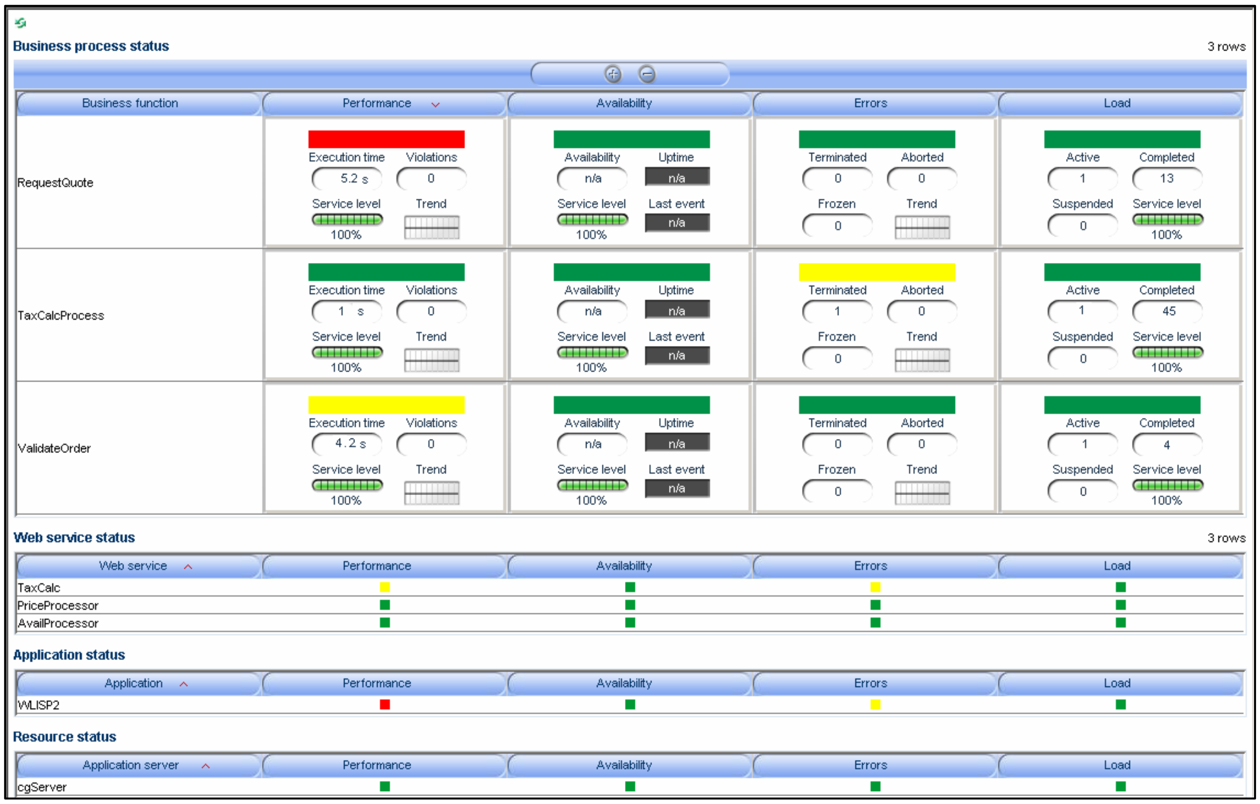
Acsera Manager connects to a WebLogic Admin server and uses JMS to determine all of the members of the domain. Most monitoring systems available today do this as well. However the Acsera Manager takes further steps to contact the Acsera EJB deployed on these domains to begin the business unit of work discovery process. The Acsera EJB communicates information about the deployed applications on that managed server back to the Acsera Manager. [Figure 5] The Manager analyzes this data to construct a model of the deployed applications. The model is mapped into a hierarchy to provide structure for the metrics that will be collected. This modeling process understands the WebLogic Frameworks which enables it to identify all of the business units of work: the processes, process nodes, portals, portlets, pageflows, web services, and struts modules.



[Figure 5]

Continuing deeper into the application, the Java Controls that support these constructs are discovered next and the methods on the controls are identified for dynamic and automatic instrumentation. Using a set of heuristics the Acsera Manager identifies the necessary probe points needed to achieve the desired monitoring. These probe points are sent back to the Acsera EJB, which tells the Acsera Agent where to collect metrics from the application's Java byte-code. Other monitoring solutions will attempt to collect all metrics for further analysis, this mass collection results in an undue burden on the performance of the application in production.

As the application executes, the Acsera Agent will batch send the metrics to the Acsera Manager. In addition to using byte-code instrumentation, the Manager also collects data from the managed server through JMX and runtime information from the WebLogic JDBC data sources. Acsera Manager correlates these metrics into the hierarchical model where it is analyzed and stored into a database.



**Business process status** 3 rows

| Business function | Performance | | | | | Availability | | | | | Errors | | | | | Load | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Execution time | Violations | Service level | Trend | | Availability | Uptime | Service level | Last event | | Terminated | Aborted | Frozen | Trend | | Active | Completed | Suspended | Service level |
| RequestQuote | 🟥 | 5.2 s | 0 | 100% | | 🟩 | n/a | n/a | 100% | n/a | 🟩 | 0 | 0 | 0 | | 🟩 | 1 | 13 | 0 | 100% |
| TaxCalcProcess | 🟩 | 1 s | 0 | 100% | | 🟩 | n/a | n/a | 100% | n/a | 🟨 | 1 | 0 | 0 | | 🟩 | 1 | 45 | 0 | 100% |
| ValidateOrder | 🟨 | 4.2 s | 0 | 100% | | 🟩 | n/a | n/a | 100% | n/a | 🟩 | 0 | 0 | 0 | | 🟩 | 1 | 4 | 0 | 100% |

**Web service status** 3 rows

| Web service | Performance | Availability | Errors | Load |
|---|---|---|---|---|
| TaxCalc | 🟨 | 🟩 | 🟨 | 🟩 |
| PriceProcessor | 🟩 | 🟩 | 🟩 | 🟩 |
| AvailProcessor | 🟩 | 🟩 | 🟩 | 🟩 |

**Application status**

| Application | Performance | Availability | Errors | Load |
|---|---|---|---|---|
| WLISP2 | 🟥 | 🟩 | 🟨 | 🟩 |

**Resource status**

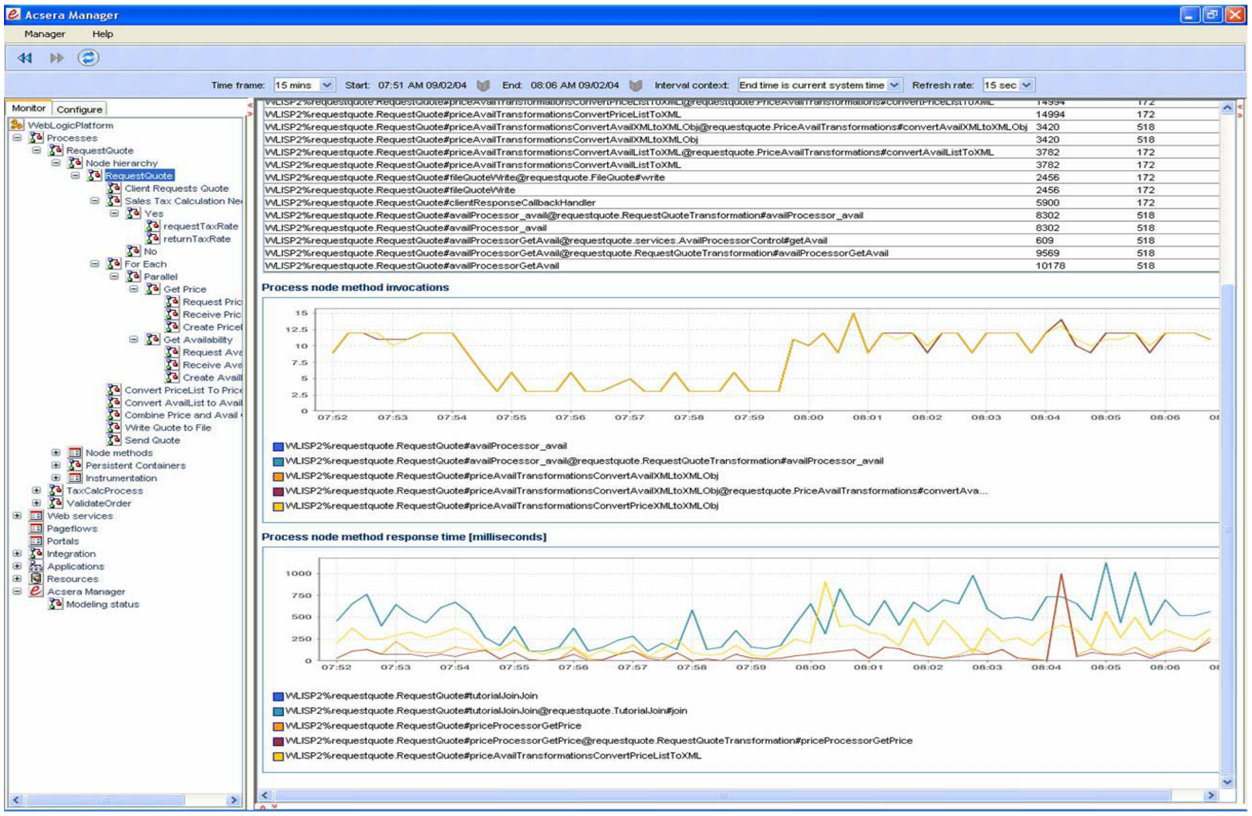
| Application server | Performance | Availability | Errors | Load |
|---|---|---|---|---|
| cgServer | 🟩 | 🟩 | 🟩 | 🟩 |

**[Figure 6]**

When an application administrator logs into the Acsera user interface via a web browser, they are provided with an operational view of the WebLogic Platform domain.[Figure 6] The screen (shown above) lists all of the deployed WLI processes, web services, pageflows, portals, applications and servers in the domain. They can view at a glance the performance, availability, errors and load on these components. By double clicking on any of the process names, a new window opens that shows the process node hierarchy and the associated performance on those nodes.

## Problem Diagnosis and Root Cause Analysis in Production

If the application displays performance anomalies or unusual error conditions, the user can go immediately to the Platform Engineering views and the Root Cause Analysis views to do immediate diagnosis of problems. The result is complete monitoring and performance management of WLI applications in production without any application specific knowledge on the part of the product team. Acsera provides 100% visibility, monitoring and performance management for the new complex enterprise applications deployed on WLI, eliminating mystery performance problems and downtime surprises. [Figure 7]



Time frame: 15 mins Start: 07:51 AM 09/02/04 End: 08:06 AM 09/02/04 Interval context: End time is current system time Refresh rate: 15 sec

| | | |
|---|---|---|
| WLISP2%requestquote.RequestQuote#priceAvailTransformationsConvertPriceListToXML@requestquote.PriceAvailTransformations#convertPriceListToXML | 14994 | 172 |
| WLISP2%requestquote.RequestQuote#priceAvailTransformationsConvertPriceListToXML | 14994 | 172 |
| WLISP2%requestquote.RequestQuote#priceAvailTransformationsConvertAvailXMLtoXMLObj@requestquote.PriceAvailTransformations#convertAvailXMLtoXMLObj | 3420 | 518 |
| WLISP2%requestquote.RequestQuote#priceAvailTransformationsConvertAvailXMLtoXMLObj | 3420 | 518 |
| WLISP2%requestquote.RequestQuote#priceAvailTransformationsConvertAvailListToXML@requestquote.PriceAvailTransformations#convertAvailListToXML | 3782 | 172 |
| WLISP2%requestquote.RequestQuote#priceAvailTransformationsConvertAvailListToXML | 3782 | 172 |
| WLISP2%requestquote.RequestQuote#fileQuoteWrite@requestquote.FileQuote#write | 2456 | 172 |
| WLISP2%requestquote.RequestQuote#fileQuoteWrite | 2456 | 172 |
| WLISP2%requestquote.RequestQuote#clientResponseCallbackHandler | 5900 | 172 |
| WLISP2%requestquote.RequestQuote#availProcessor_avail@requestquote.RequestQuoteTransformation#availProcessor_avail | 8302 | 518 |
| WLISP2%requestquote.RequestQuote#availProcessor_avail | 8302 | 518 |
| WLISP2%requestquote.RequestQuote#availProcessorGetAvail@requestquote.services.AvailProcessorControl#getAvail | 609 | 518 |
| WLISP2%requestquote.RequestQuote#availProcessorGetAvail@requestquote.RequestQuoteTransformation#availProcessorGetAvail | 9569 | 518 |
| WLISP2%requestquote.RequestQuote#availProcessorGetAvail | 10178 | 518 |

Process node method invocations

- 🟦 WLISP2%requestquote.RequestQuote#availProcessor_avail
- 🟦 WLISP2%requestquote.RequestQuote#availProcessor_avail@requestquote.RequestQuoteTransformation#availProcessor_avail
- 🟧 WLISP2%requestquote.RequestQuote#priceAvailTransformationsConvertAvailXMLtoXMLObj
- 🟥 WLISP2%requestquote.RequestQuote#priceAvailTransformationsConvertAvailXMLtoXMLObj@requestquote.PriceAvailTransformations#convertAva...
- 🟨 WLISP2%requestquote.RequestQuote#priceAvailTransformationsConvertPriceXMLtoXMLObj

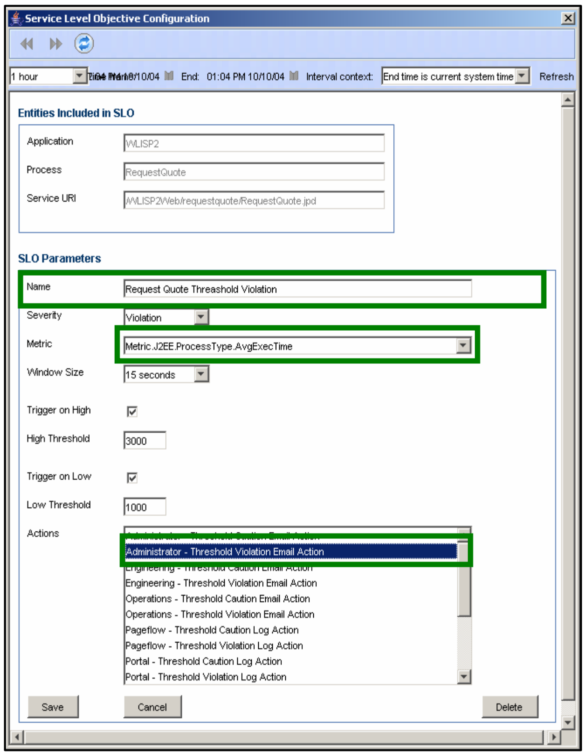
Process node method response time [milliseconds]

- 🟦 WLISP2%requestquote.RequestQuote#tutorialJoinJoin
- 🟦 WLISP2%requestquote.RequestQuote#tutorialJoinJoin@requestquote.TutorialJoin#join
- 🟧 WLISP2%requestquote.RequestQuote#priceProcessorGetPrice
- 🟥 WLISP2%requestquote.RequestQuote#priceProcessorGetPrice@requestquote.RequestQuoteTransformation#priceProcessorGetPrice
- 🟨 WLISP2%requestquote.RequestQuote#priceAvailTransformationsConvertPriceListToXML

**[Figure 7]**

## Integrating Application SLAs with Enterprise Management Systems

A typical enterprise is managed in layers. As noted above, the new J2EE applications are delivering business-critical services and require the advanced technology found in Acsera for true SLA management. In addition, the enterprise Network Operations Center (NOC) is likely to use a horizontal framework such as HP OpenView or IBM Tivoli for an overall picture of the enterprise. Acsera complements these Enterprise Management Systems (EMS) with the ability to trigger alerts, alarms that are sent directly to the EMS platforms. As shown in the below diagram, integration with these ubiquitous EMS platforms is a standard feature of Acsera. [Figure 8]



Service Level Objective Configuration

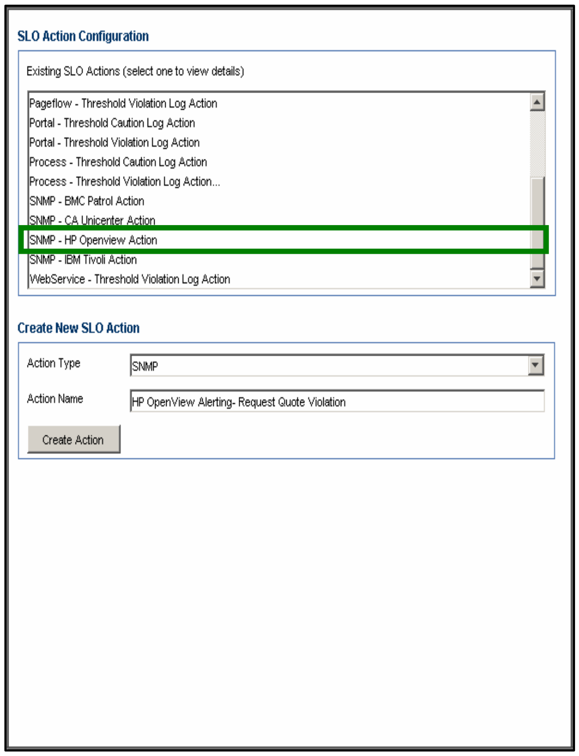
1 hour | Time frame: 12:04 PM 10/10/04 | End: 01:04 PM 10/10/04 | Interval context: End time is current system time | Refresh

**Entities Included in SLO**

| | |
|---|---|
| Application | WLISP2 |
| Process | RequestQuote |
| Service URI | /WLISP2Web/requestquote/RequestQuote.jpd |

**SLO Parameters**

| | |
|---|---|
| Name | Request Quote Threashold Violation |
| Severity | Violation |
| Metric | Metric.J2EE.ProcessType.AvgExecTime |
| Window Size | 15 seconds |
| Trigger on High | ✓ |
| High Threshold | 3000 |
| Trigger on Low | ✓ |
| Low Threshold | 1000 |
| Actions | Administrator - Threshold Caution Email Action<br>[✓ Administrator - Threshold Violation Email Action]<br>Engineering - Threshold Caution Email Action<br>Engineering - Threshold Violation Email Action<br>Operations - Threshold Caution Email Action<br>Operations - Threshold Violation Email Action<br>Pageflow - Threshold Caution Log Action<br>Pageflow - Threshold Violation Log Action<br>Portal - Threshold Caution Log Action<br>Portal - Threshold Violation Log Action |



**SLO Action Configuration**

Existing SLO Actions (select one to view details)

- Pageflow - Threshold Violation Log Action
- Portal - Threshold Caution Log Action
- Portal - Threshold Violation Log Action
- Process - Threshold Caution Log Action
- Process - Threshold Violation Log Action...
- SNMP - BMC Patrol Action
- SNMP - CA Unicenter Action
- [✓ SNMP - HP Openview Action]
- SNMP - IBM Tivoli Action
- WebService - Threshold Violation Log Action

**Create New SLO Action**

| | |
|---|---|
| Action Type | SNMP |
| Action Name | HP OpenView Alerting- Request Quote Violation |

Complete Integration with Enterprise Management Systems **[Figure 8]**

Acsera supports a true enterprise-class high-availability solution including fail-over manager configurations that support continuous monitoring with no single-point-of-failure. This is in stark contract to hand-built management solutions based on profilers and code-level tools. These systems often embed their primitive alerting systems in a client without any protection from single node failure or network disruption.

## Risk Reduction and Rapid ROI

Acsera lowers the risk of deployment by automating production monitoring and management of complex J2EE applications. Unlike source-level solutions that require weeks of human effort and error prone hand-instrumentation, Acsera accomplishes automated discovery and instrumentation in less than 15 minutes. Far fewer J2EE experts are required to move an application from development into production and applications have greater reliability and performance.

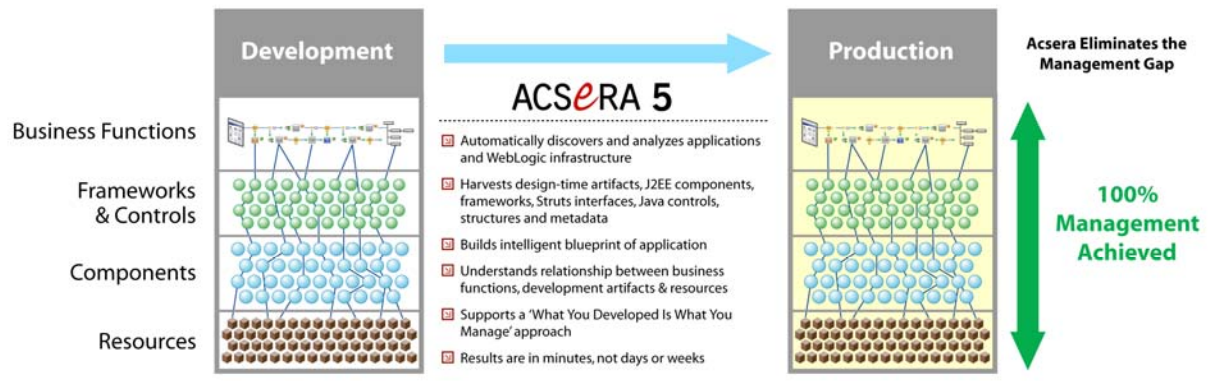
Because Acsera automates a set of complex tasks currently done by teams of engineers, it has the potential to save hundreds-of-thousands of dollars in the first 90 days of operations alone. The ROI 'break even' for Acsera is less than 4 weeks for many customers.

## Lowering the Total Cost of Ownership by Closing the Management Gap

Acsera was built from the ground up to close the 'management gap' described above and to help enterprise customers achieve and maintain optimal performance and availability for complex J2EE applications. It is focused on meeting the total business and technology requirements for large scale enterprise applications deployed in a distributed environment. The results reported by our enterprise customers are compelling.

- Full support for Service Level Agreements in production J2EE applications
- Full visibility of application performance and availability – no blind spots or surprises
- Problems can be diagnosed in hours instead of days in production
- Applications in production have greater uptime and reliability
- The Acsera model 'scales' – IT departments can deploy large numbers of new applications successfully without high-risk human engineering efforts to fill management gaps

Acsera lowers the enterprise cost of ownership for J2EE applications by providing the most advanced and complete solution in the industry for production monitoring and management.


Development
Production
Acsera Eliminates the Management Gap
Business Functions
Frameworks & Controls
Components
Resources
ACSERA 5
Automatically discovers and analyzes applications and WebLogic infrastructure
Harvests design-time artifacts, J2EE components, frameworks, Struts interfaces, Java controls, structures and metadata
Builds intelligent blueprint of application
Understands relationship between business functions, development artifacts & resources
Supports a 'What You Developed Is What You Manage' approach
Results are in minutes, not days or weeks
100% Management Achieved