IBM Software Group

# IBM WebSphere® Application Server V5.1.1

## *Request Metrics*

e business on demand.

# Goals

- Describe Request Metrics functionality that is provided with WebSphere
- Introduce Application Response Measurement (ARM) standard
- Outline enhancements introduced in WebSphere V5.1.1
- Describe Request Metrics settings

# Agenda

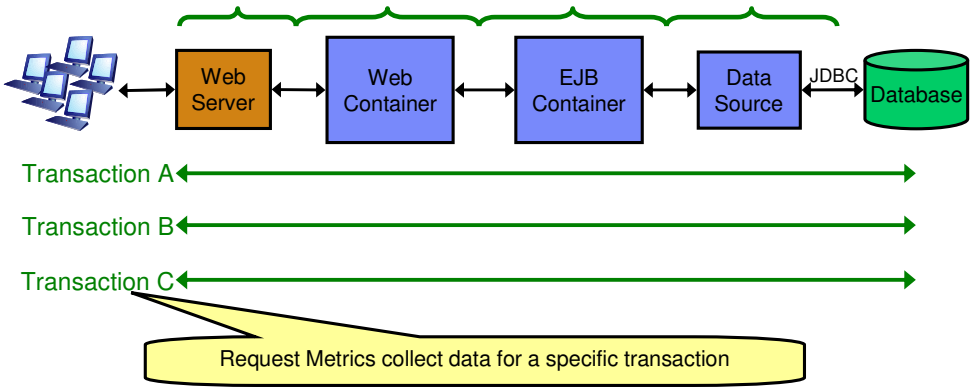
- Overview
- Configuration Options
- WebSphere Application Server V5.1.1 enhancements
- Configuring Request Metrics (Viewlet)

# Section

***Overview***

# Request Metrics

- Request metrics track individual transactions, recording and correlating process time in each of the major components
  - Components supported: Web server plug-ins, web container, EJB™ container, JDBC calls, Web Services
  - Ex: Response time for transaction A





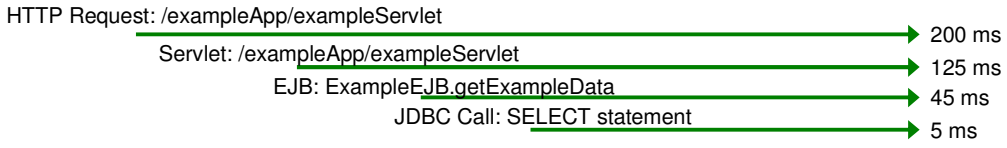


Request Metrics allow you to monitor transactions as they move through the different components in a WebSphere environment (HTTP server plug-in, Web/EJB container and database levels). Request monitoring can follow requests as they fan out across a WebSphere environment, monitoring them even as they cross process and host boundaries.

Requests can only be traced if they start in either an HTTP or enterprise bean remote request. The associated downstream Web\EJB container and database requests are also captured, allowing the user to correlate the amount of time spent executing in the different tiers. This allows a system administrator to quickly close in on the part of the system where performance is being degraded.

## Why Use?

- Allows hierarchal view for each individual transaction by response time
  - Provides data for debugging resource constraints
- Provides filtering mechanisms to monitor synthetic transactions or specific transactions
  - Users can exclusively track artificial transactions to measure performance
- Provides response time on individual transactions, helping determine if Service Level Agreements are being met
  - EWLM tracks and identifies when SLA targets are **not** met



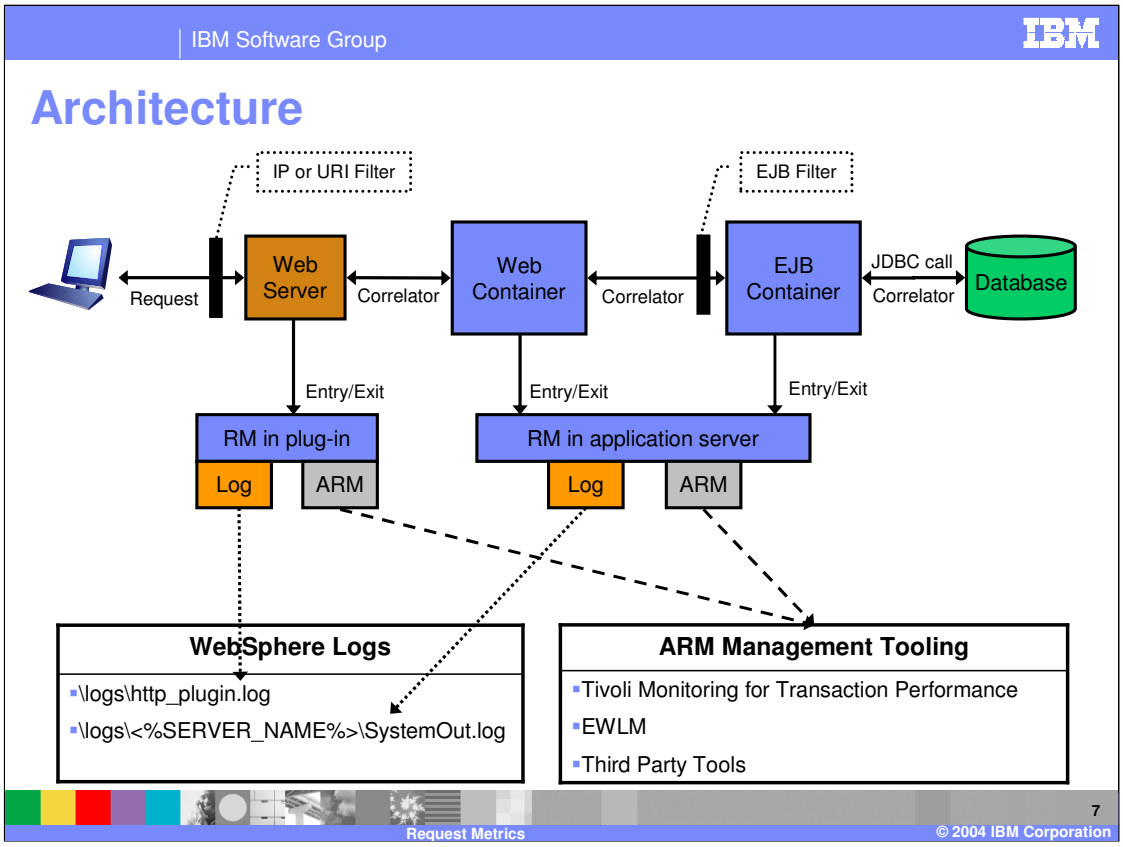




Request metrics can be used to monitor the health of your application through the use of synthetic transactions. Synthetic transactions are transactions that the are initiated from a known source with the purpose of monitoring specific transaction times. Synthetic transactions allow you to monitor your application's responsiveness and health. With the addition of system monitoring software (e.g. Tivoli® Monitoring for Transaction Performance), a system administrator can be alerted when response times are too low or if transaction are failing.

Request metrics provide valuable hierarchal data on transactions in your application. It allows you to see which part of your application calls another part – allowing you to determine whether the application is behaving according to design. The hierarchal view also allows for the quick determination of which part or portion of your application may be responsible for poor performance.

Request metrics also provide valuable measurements for determining whether Service Level Agreements (SLAs) are being achieved. Because request metrics can show trace data from specific transactions in the system, it is an invaluable tool in determining whether you are meeting your SLA targets. In addition, request metrics can be paired with other tools, such as Enterprise Workload Management (EWLM), which can be used to monitor and warn when certain targets are not being met.




The diagram above shows the different parts to a simple, request-metrics-enabled WebSphere application server. The diagram assumes that request metrics has been enabled, logging and ARM reporting has been enabled, and appropriated filters are in place.

As a request enters the system, it will be examined by the filter settings in the web server plug-in. If the request matches one of the associated filters, an ARM transaction will be started and a correlator will be assigned in the web server plug-in. When the request moves into the Servlet Container, another ARM transaction will be started and a correlator value will be assigned to the transaction. Additionally, the correlator value that was passed from the upstream container will be assigned as the ARM transaction's parent correlator – this allows the two separate transactions to later be correlated together. This behavior continues until the last point in the transactional chain is reached.

As the response leaves the system, it will stop the ARM transaction timers in each of the containers. It also records the appropriate transaction information and time in each of the activated logging mechanisms.

In addition to the simple architecture shown above, the client could also pass the correlator from a request metrics transaction to another downstream ARM-enabled software that is not directly controlled by WebSphere. An example of this is when a transaction would move into a legacy backend, such as CICS® server.

## Section

# *Configuration Options*

# Request Filtering

- Use filters to limit the amount of transactions that are logged
  - Example: Use IP address filters to monitor synthetic transactions
- Available filters
  - HTTP requests can be filtered by IP address, URI or both
  - Enterprise Bean requests can be filtered by method name
- Performance impact ~5% when ~20% of transactions pass through filters
- Administrative Console path: Troubleshooting > PMI Request Metrics




Filters can be specified to limit who and what performance data is collected on. Collecting PMI Request Metric information is a resource intensive operation. In a production environment it is highly recommended that filters are enabled to only capture metrics on key business processes/methods or from a specific IP address.

The following filtering options are available:

•Filtering incoming HTTP requests (one or both of the options below can be active at the same time)

- •Client IP Address Filters: Use client IP address filters in the data center to filter known addresses and provide a mechanism to make a request or set of requests while the system is under normal load.
- •URI Filters: URI filtering provides a mechanism to filter, based on the URI of the incoming HTTP request.

•Filtering incoming enterprise bean requests

- •Enterprise bean method name filters are specified with the fully package name qualified enterprise bean method name.

## Trace Levels

- Use trace levels to limit the depth of a transaction that is logged
  - Higher trace level = greater performance hit
- Available trace levels
  - None – No data captured
  - Hops – Process boundaries (web server, servlet, EJB over RMI-IIOP)
  - Performance Debug – Hops + 1 level of intra-process calls
  - Debug – Full capture (all cross-process/intra-process calls)
- Administrative Console path: Troubleshooting > PMI Request Metrics




Trace levels control the depth of information collected. For most purposes the setting of "HOPS" will collect enough information – HOPS collects trace information for requests that jumps between processes. When more information is needed, you can either choose to collect just the first intra-process call (PERFORMANCE DEBUG) or all intra-process calls (DEBUG setting).

•Intra-process calls include servlet-to-EJB, EJB-to-EJB, servlet include/forward calls.

When setting the Request Metrics trace level, the same trace level setting is used to set both the ARM and logging settings.

# Request Metrics Output

- Output to WebSphere's logs (default behavior)
  - HTTP requests output to <%WAS_INST%>\logs\http_plugin.log
  - Servlet/EJB/JDBC requests output to <%WAS_INST%>\logs\<%SERVER_NAME%>\SystemOut.log
- Output to Application Response Measurement (ARM) agent and visualized using ARM management software
  - IBM Tivoli Monitoring for Transaction Performance (TMTP)
  - IBM Enterprise Workload Management (EWLM)
  - Other third party tools
- Output can be directed to either location or both at the same time
  - Recommended practice – disable request metric logging when implementing an ARM agent to reduce disk I/O




Request Metric data can be recorded to two different places - either directly to WebSphere's logs or to an ARM agent which allows it to be viewed through ARM management software.

•Application Response Measurement (ARM) - Provides a mechanism whereby applications can provide response time measurement to a centralized reporting and management facility.

•For best performance and utilization of the collected data, write it to an ARM agent and disable logging. To do this, create the following Customer Process for WebSphere's JVM.

•com.ibm.websphere.pmi.reqmetrics.loggingEnabled = false

When request metrics logging is enabled, the request metrics entries can be found by looking for the following:

•http_plugin.log: Look for entries prefixed with "PLUGIN"

•<%SERVER_NAME%>\SystemOut.log: Look for entries prefixed with "PMRM00031"

WebSphere Application Server does not provide an ARM agent, but can be used with Tivoli's ARM 2.0 agent and any ARM agent written to Open Group's ARM 4.0 standard.

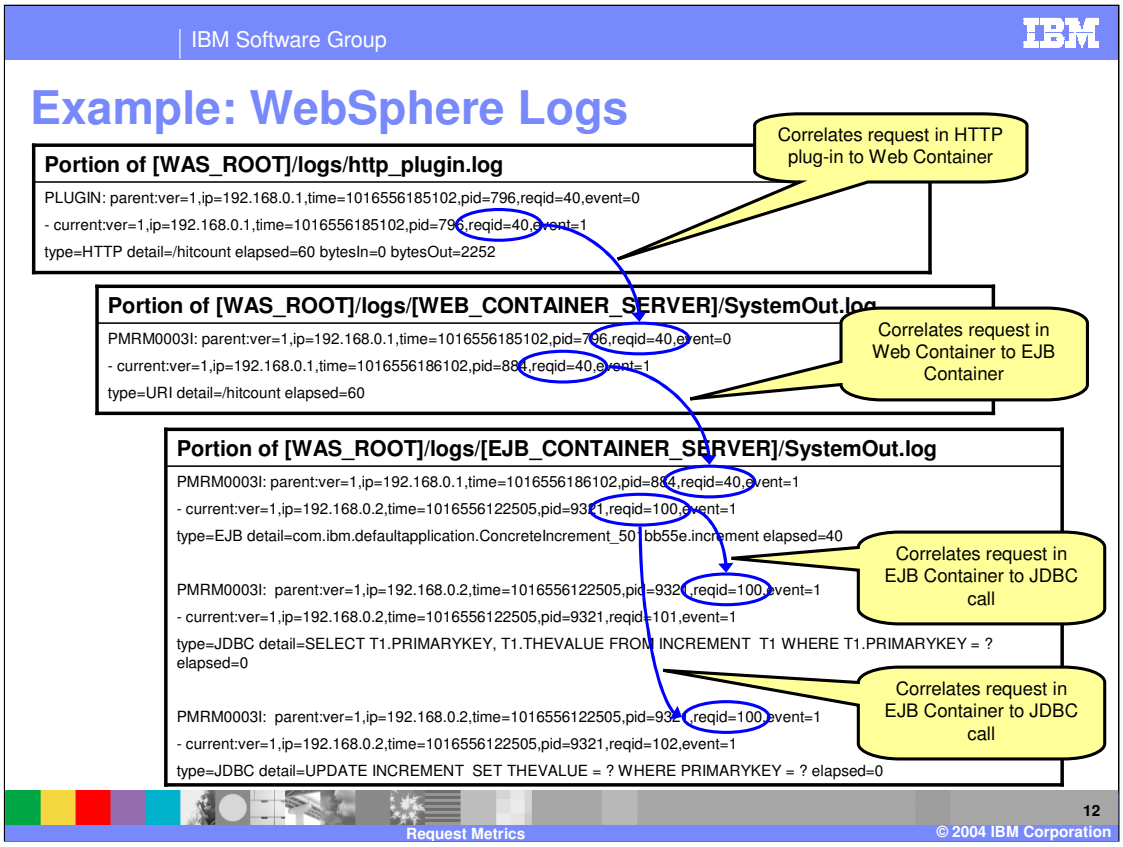
•Examples of software providing ARM 4.0 agents: Tivoli Monitoring for Transaction Performance (TMTP) and Enterprise Workload Management (EWLM).

The following software are IBM solutions that utilize request metrics in their monitoring solutions.

•IBM Tivoli Monitoring for Transaction Performance (TMTP) – IBM Tivoli monitoring solution that marries synthetic transaction monitoring and PMI request metric data collection. Allows users to correlate gathered data to analyze possible performance problems.

•Link to TMTP: http://www.ibm.com/software/tivoli/products/monitor-transaction/

•Enterprise Workload Management (EWLM) – Will be discussed in a later presentation.




The diagram above shows excerpts from http_plugin.log and SystemOut.log log files. The diagram is meant to show how the Request Metric entries in the different logs correlate to each other. Also note that as the request moves through the WebSphere application server, you can view exactly how long each leg of the request took:

•The request took 0 seconds to complete within the HTTP server. This is derived by subtracting the value of “elapsed” from the Web container’s SystemOut.log from the value of “elapsed” in the http_plugin.log (i.e. 60 milliseconds – 60 milliseconds = 0 milliseconds).

•The same steps can be used to find the amount of time spent in the Web Container, EJB Container, and database server.

For more detail on the Request Metric field meanings, see the WebSphere Information Center.

## Section

# *Application Response Measurement (ARM)*

## Application Response Measurement (ARM)

- Open Group standard
- Provides a means through which business and management applications can cooperate to measure selected business transactions
  - Measures service levels of single-system and distributed applications
- For more information see:
  http://www.opengroup.org/tech/management/arm/




The ARM standard was created to simplify the task of managing increasingly complex applications. It helps answer the following questions about your application's performance.

•Are transactions succeeding?

•If a transaction fails, what is the cause of the failure?

•What is the response time experienced by the end-user?

•Which sub-transactions of the user transaction take too long?

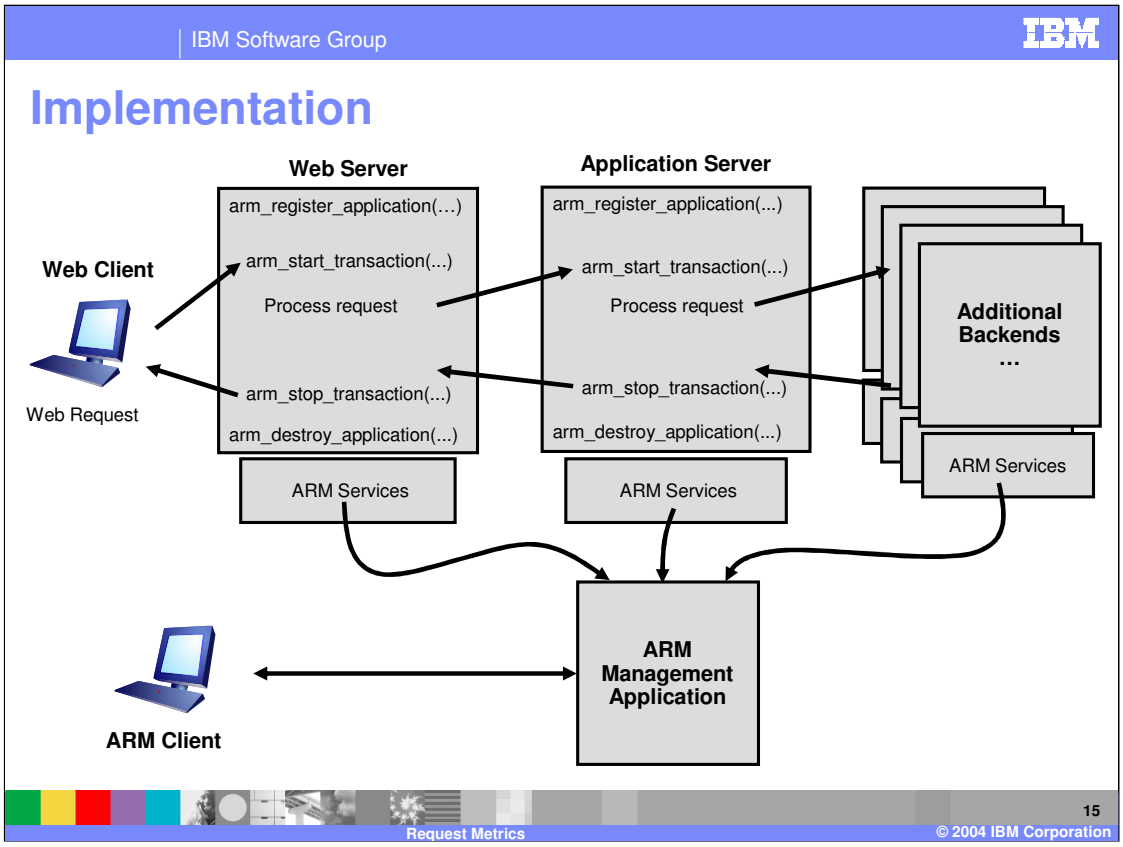
•Where are the bottlenecks?

•How many of which transactions are being used?

•How can the application and environment be tuned to be more robust and perform better?

More importantly it is a standard for measuring service levels of single-system and distributed applications. ARM measures the availability and performance of transactions (any units of work), both those visible to the users of the business application and those visible only within the IT infrastructure, such as client/server requests to a data server.

•ARM provides the standard for software developers to write to. The collection, recording, and analysis of the data is left up to the individual tool-vendors to implement.




The diagram above depicts a typical way in which an ARM environment way be implemented. In this case we are showing a monitored application environment that includes a web server, an application server, and additional backend systems (e.g. database servers, CICS servers, MQSeries® servers, etc.).

A prototypical ARM implementation may include all or some of the parts listed below.

•ARM-instrumented application: The application (web server, application server, etc.) that is being monitored needs to be ARM-instrumented in order to externalize transaction data.

- •Data includes: transaction start/stop times, correlators, additional transaction details, etc.

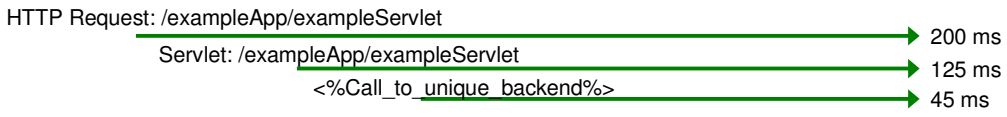
•ARM agent: Agent is necessary to externalize performance data collected within the monitored application to the ARM management application. The implementation of the agent will differ between different software products, however, all agents will be written to the same ARM API.

•ARM management application: This is responsible for collecting and analyzing data from all connected ARM agents. The implementation of this will differ between software products.

•ARM client: Additionally the ARM management application may provide some sort of GUI for viewing analysis information. The implementation of this will differ between software products.

# Request Metrics and ARM

- Supports following ARM interfaces
  - Open Group ARM 4.0 standard
    - Supported in all components
  - Tivoli ARM 2.0
    - Supported in all components **except** web server plug-ins
- API available that allows the ARM correlator to be extracted from the Request Metrics ARM transaction stack
  - Allows customer to integrate downstream ARM calls with ARM transactions that are already in flight within Request Metrics
  - Use when customer applications communicate to a non-WebSphere container downstream along an un-supported protocol







WebSphere supports ARM 4.0 across all of its ARM implemented components. Support for the Tivoli ARM 2.0 agent is **only** available in the servlet container, EJB container, and JDBC drivers.

When enabling the ARM agent within WebSphere you have to additionally specify the following two JVM properties.

•**com.ibm.websphere.pmi.reqmetrics.ARMIMPL**

•Specifies which type of ARM implementation is being used. By default the property is set to ARM 4.0. There are additional settings for Tivoli ARM 2.0 support.

•**ArmTransactionFactory**

•Specifies the ARM factory to be used when creating new objects. Value needs to be set to the fully qualified class name.

The API for accessing the current ARM transaction is useful in a situation where a transaction needs to be traced as it moves from WebSphere to a non-WebSphere backend system. If you want to continue to get accurate transactional timer info, then the correlator can be extracted from the current request metric ARM transaction and passed to the downstream system.

An example of how the correlator would be used follows:

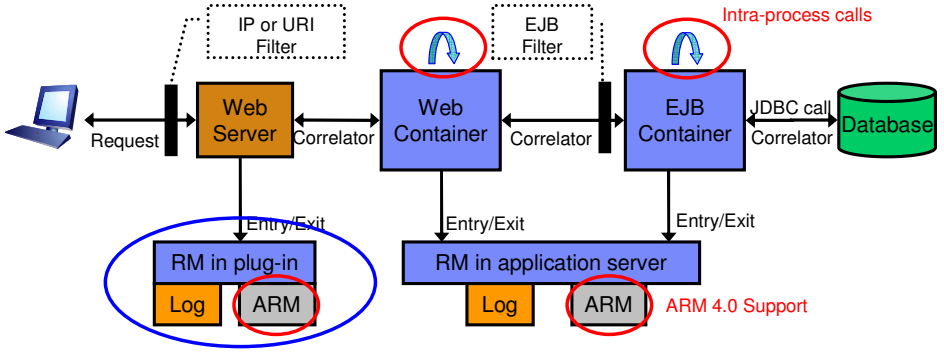
•Within your application you would use the API to extract the correlator from the ARM transaction that is on the top of the stack – this would be the ARM transaction tied to the current Request Metric transaction. You would then use the correlator to tie or correlate any transaction (any unit of work) downstream back to the correct parent transaction. This is helpful because it allows you to correlate any future calls that you may make to ARM with the work that was done upstream in the application.

•This type of scenario could be found in an environment where requests flow through a web server, a web application server, and then into some backend such as MQ Series or CICS. In this case, you would not be able to gather request metrics for the part of the environment that lies out of WebSphere control. Worse yet, without the correlator you would be unable to correlate the transactions that took place in the WebSphere environment with those that took place in the CICS environment. To alleviate this, the correlator should be extracted from the Request Metrics ARM transaction stack it should be flowed downstream. This allows child transactions (initiated on a non-WebSphere platform) to be correlated back to the parent ARM transaction (in this case the last Request Metric ARM transaction). The end result is that you are able to more accurately monitor you application and obtain a better feel of its performance.

## Section

# *V5.1.1 Enhancements*

# Enhancements: Through The Releases




**V5.0**
- Captures process time for EJBs (via IIOP), Servlets, and JDBC calls
- Outputs data to WebSphere logs or ARM agents (Tivoli ARM 2.0 and EWLM DE)

**V5.0.2**
- Captures process time for web server plug-ins and provides data in plug-in logs

**V5.1.1**
- ARM interface updated to ARM 4.0
- Captures process time for intra-process servlets and EJBs
- Captures process time for Web Services and provides data in WebSphere logs
- Provides ARM interface for web server plug-ins
- Opens correlation mechanism to plug-in other response time measurements needed by customer




The diagram shows the different features that have been added and in which WebSphere release they were added in.

## Section

# *Configuring Request Metrics*





The demonstration will show how to access the Request Metrics panel in the Administrative Console. Additionally it will explain the different options that are available.

## Section

# *Summary and Reference*

# Summary

- Request metrics track transactions, recording and correlating the process time in each WebSphere component
- Use with ARM applications to monitor status and response times of key transactions in your business applications
- Set filters and trace levels to minimize performance degradation

# Reference

- To review the ARM 4.0 specification see
  - http://www.opengroup.org/management/arm.htm/
- For more information on the ARM standard see
  - http://www.opengroup.org/pubs/catalog/c807.htm
- For more information on Tivoli Monitoring for Transaction Performance (TMTP) see
  - http://www-306.ibm.com/software/tivoli/products/monitor-transaction/

# Trademarks and Disclaimers



The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

| | | | | |
|---|---|---|---|---|
| IBM | iSeries | OS/400 | Informix | WebSphere |
| IBM(logo) | pSeries | AIX | Cloudscape | MQSeries |
| e(logo)business | xSeries | CICS | DB2 Universal Database | DB2 |
| Tivoli | zSeries | OS/390 | IMS | Lotus |

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both. Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are
trademarks of Intel Corporation in the United States, other countries, or both. UNIX is a registered trademark of The Open Group in the United States and other countries. Linux is a registered trademark of Linus Torvalds. Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program that does not infringe IBM's intellectual property rights may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.