WHITE PAPER

Rapid problem resolution for complex WebSphere MQ applications

Introduction

At many organizations, WebSphere MQ is the glue that connects mission-critical applications and drives key business initiatives. Despite this importance and high visibility, when problems occur in MQ-related application programs, the resolution process often involves an ad-hoc combination of end-user input, jerry-rigged system tools, on-the-fly debugging code and intuition.

This paper offers a structured approach for resolving MQ application problems. The emphasis here is to use application tools to resolve application problems. We walk through an MQ-related problem, and using the Compuware product suite, show how one might build an organized process to rapidly detect, understand and correct problems.

Problem resolution involves three major objectives:

- Awareness. Make the problem-solvers aware of a problem as quickly as possible.
- Characterization. Provide the necessary background information so the problem-solvers can confidently understand the problem.
- Resolution. Leverage background information to correct the problem—quickly and accurately.

Sample application

Our application, as shown in Figure 1, is a fairly typical MQ-based system. A request comes into a mid-tier server, in this case a Windows server, from the Internet or an intranet. The mid-tier server converses with mainframe applications via MQ messages to build the desired response. The response is then returned to the requester. In this case, the mid-tier server is running MQ client to the mainframe.

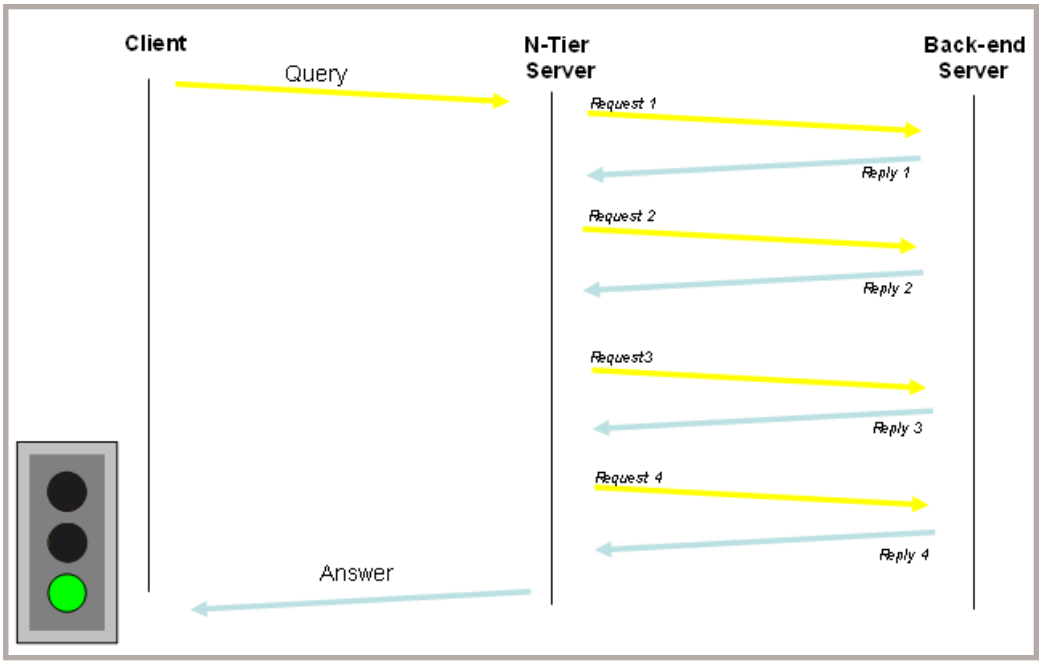

Figure 1. Sample application

Awareness: Real-time notification when a problem occurs

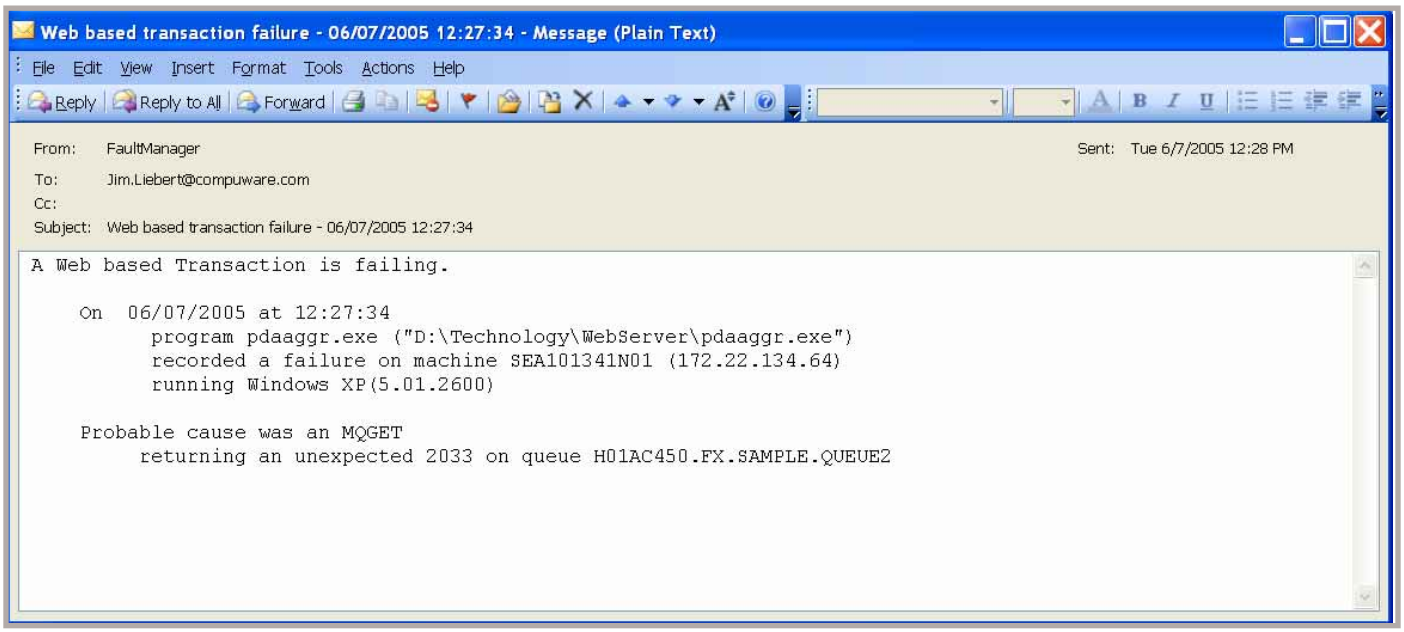

Figure 2. Real-time problem notification via e-mail

Often the most time-consuming step in the critical path of problem resolution is for the appropriate personnel to become aware that a problem even exists. This is particularly true as mainframe data extends out into the distributed world; often the battle-tested mainframe mechanisms for problem notification fall short in these wider environments. This can be exacerbated by symptom uncertainty. What symptom is presented to the end user when problems arise?

- If a program upstream abends, what does the end user experience?
- If a program upstream fails, is the end-user symptom the same or different?
- Is the onus on the end user to make IS aware of failing applications?
- If so, does the error present itself in such a way that the end user would be likely to report the problem?
- Are the end users fellow employees or customers? Would a customer report a problem?

The answers to these questions are likely to be vague; even the responsible programmer may be unsure as to the personality of their programs when problems occur. The preferred solution would be a consistent notification process that removes the end user from this loop. This is the first benefit that the Compuware tool set can bring to this problem resolution process: the ability to generate real-time notification (via e-mail, pager, help ticket) should a problem occur. This notification can be initiated via a program failure (such as a gpf or data exception) or can be initiated programmatically when the program detects the problem (such as a negative SQLCODE during a SQL call or, as in this case, an unexpected MQ call failure).

A closer look reveals other benefits to this approach: real-time notification; the e-mail was generated at 12:28 and reports a problem that occurred at 12:27:34. Even if the end users were apt to report a problem, they could never achieve this level of early response. Secondly, the e-mail can be tailored to include some early characterization information; in our case, the server where the error occurred (SEA101341N01), the name and path to the offending program, and supporting information indicating the error was related to an unexpected 2033 returned against queue H01AC450.FX.SAMPLE.QUEUE2. Compuware Fault Manager provides a sophisticated real-time notification process without necessitating any significant in-house-written debug support routines.

Tip #1. Use Abend-AID's Fault Manager to provide real-time problem notification—and take the end user out of the loop.

Characterization: Early MQ background information

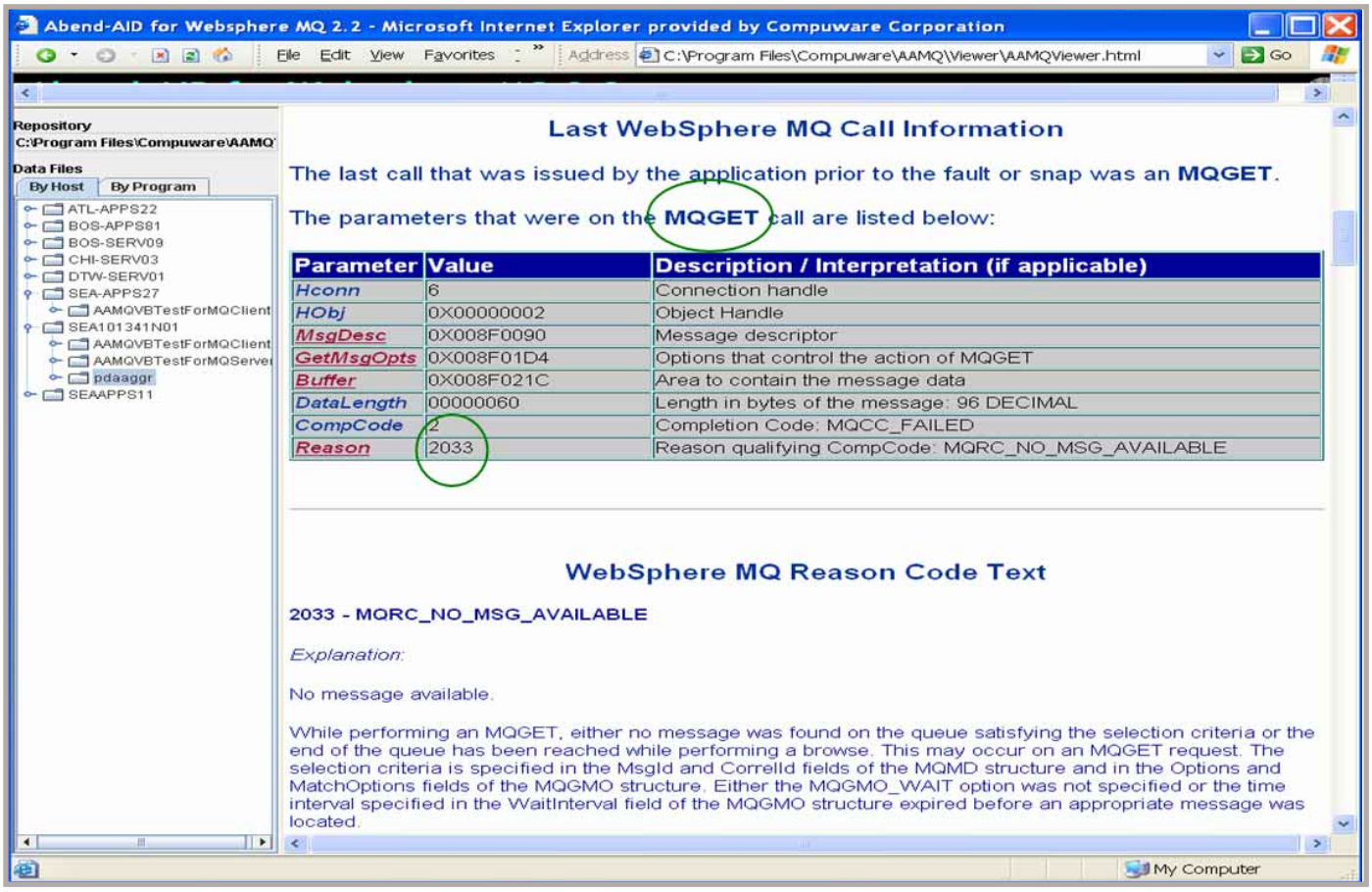

Figure 3. Early MQ detail information

Once we become aware that a problem exists, the next step is to better understand the problem. Because our sample problem involves MQ, Compuware Abend-AID *for WebSphere MQ* (for Windows or mainframe) can also provide a detailed look into the last MQ call and the MQ environment at the time of the problem. Here we see the completely rebuilt MQGET associated with the last MQ call (the returned 2033)—including some suggestions on how to correct the problem. Hyperlinked parms indicate more information is available; for instance, clicking on the MsgDesc will show the completely formatted MQ message descriptor; clicking on the Buffer would show the last retrieved message (had this MQGET succeeded).

Again, this information is provided without the need for any extensive in-house modifications. And it is not tied to a MQ failure but merely the presence or absence of MQ when the diagnostic information was generated (either via a program fault or by calling a provided API). In many cases, this will be enough detail, captured at the point of first failure, to allow the programmer to understand and address the root cause and resolve the problem.

The objective here is rapid problem resolution; to accumulate the maximum amount of information at the point of failure and present it in the most meaningful way—and quickly close the problem resolution circle.

Based on information we have gathered so far, let's review what we currently know about our sample application problem.

Tip #2. Use Abend-AID *for WebSphere MQ* to provide and isolate the MQ detail from the program—allowing the programmer to quickly identify or eliminate MQ as a suspect during problem resolution.

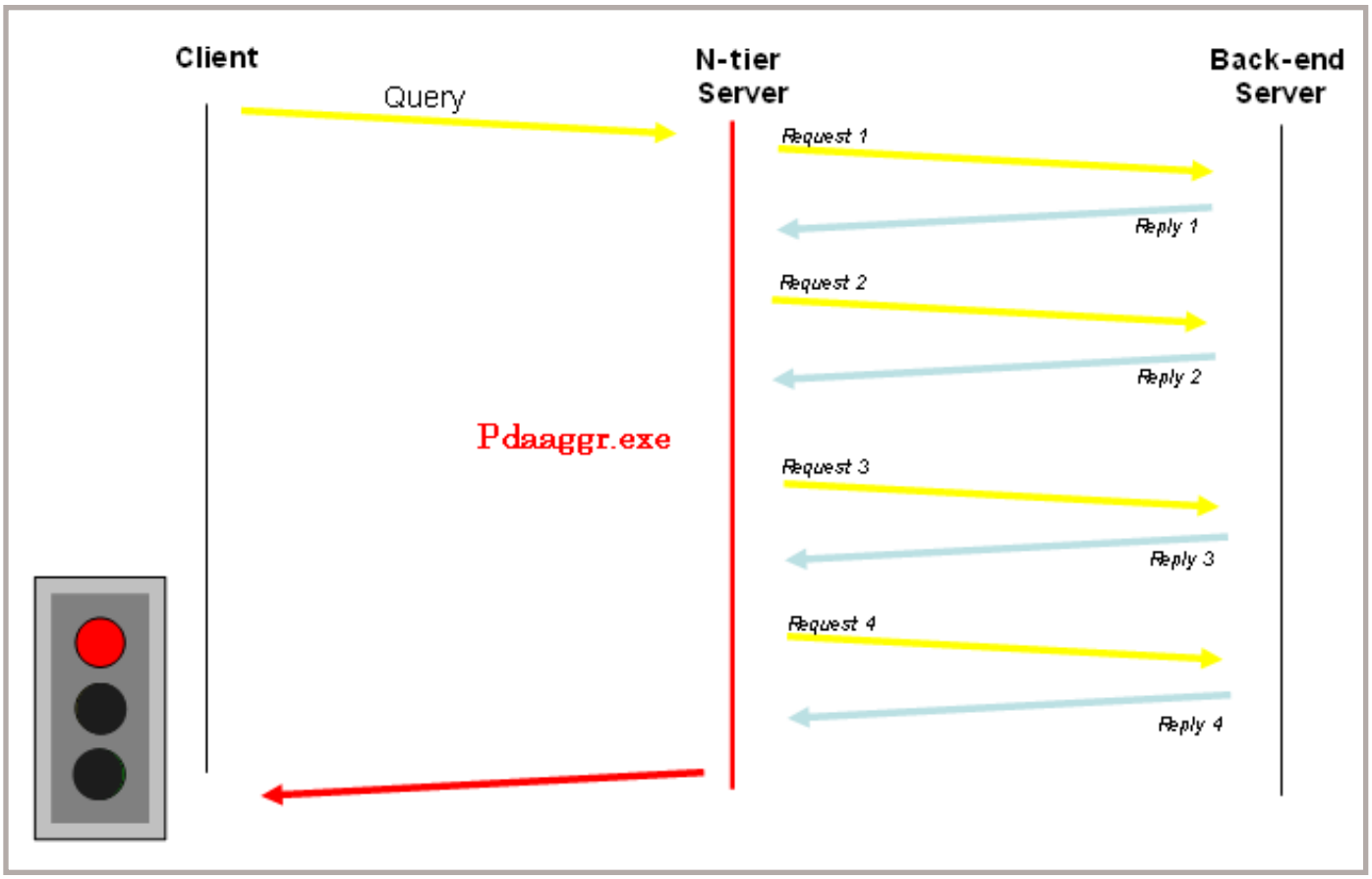

Figure 4. Sample application problem status after notification and early classification

At this point, we know several things about our application:

- It occasionally fails.
- The failure is reflected in program pdaaggr.exe running on the Windows server.
- The problem seems to be related to an unexpected 2033 (*Message not found*) on an MQGET.

And there are also several things we do not know about the problem:

- What is the end-user symptom? (*"Page Not Found"? "Please try again later"? Incorrect or stale output?*)
- How frequently is the application failing?
- What is the scenario that leads to the 2033?

While in some situations our current characterization information might be enough to resolve the problem, in this case, further research is needed. That is another byproduct of this structured, application-driven, problem-resolution approach: to create a clear next step even as the problem grows more complex.

Here the next objective on our problem-resolution critical path is to better understand the message flow between the mid-tier server and the back-end server. Understanding the message flow will help us to better pinpoint the 2033.

Characterization: Record and analyze MQ message flow

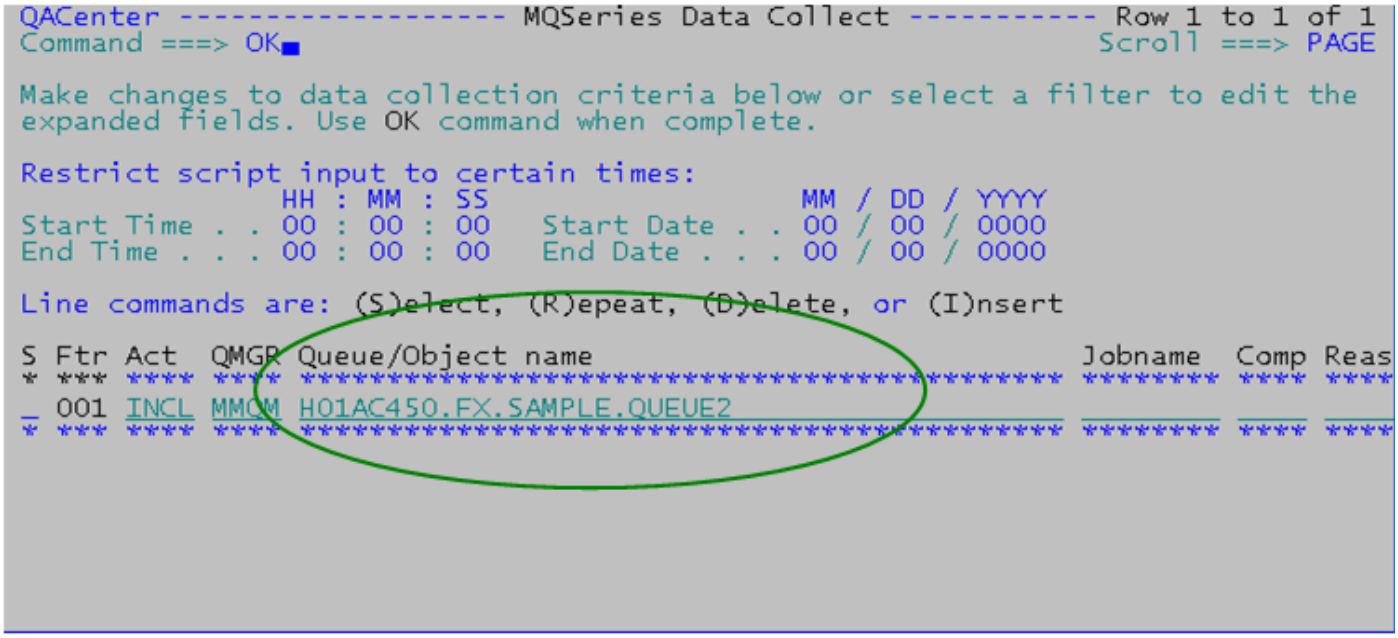

Figure 5. Record MQ message flow on the queue in question

The question "Where is my message?" is at the crux of many MQ-related problems. Here we know the application that was waiting for the message, the related queue manager and queue, and the associated error. What we don't know is the event or sequence of events that caused the error. Since our message was not found, where did it go?

Compuware QACenter *for WebSphere MQ* can accurately answer that exact question. We can record the message flow in and out of a Queue Manager for all queues or down to one specific queue. As shown in Figure 5, we are activating the record function for the queue manager MMQM and concentrating on our "queue of interest," H01AC450.FX.SAMPLE.QUEUE2—the queue that received the 2033 that started the problem. We can now leave this low-overhead recording active until the problem reoccurs, at which point we can analyze the traffic and possibly identify the cause of the 2033.

Tip #3. Use *QA*Center *for WebSphere MQ* to record MQ message flow between, among and through MQ queues.

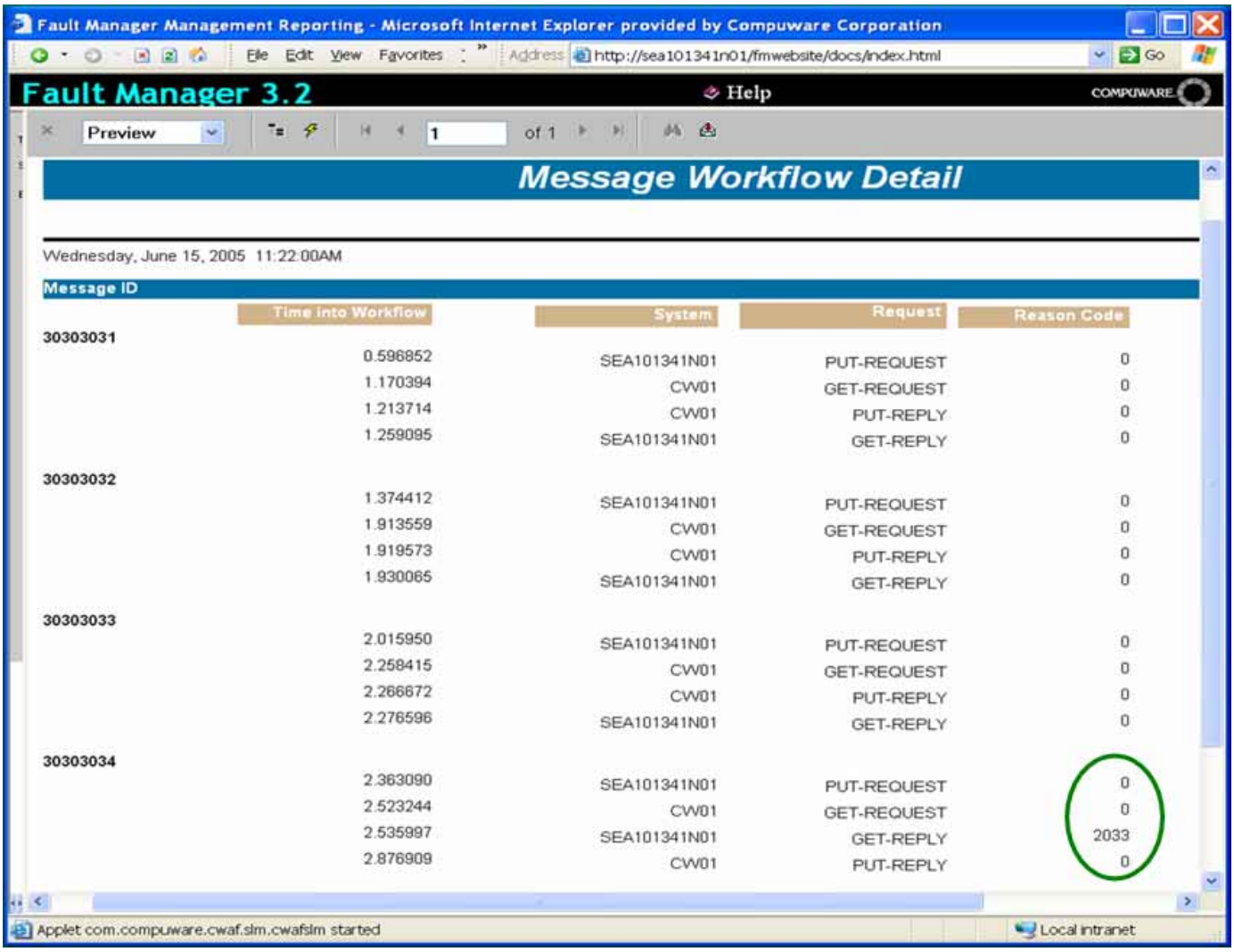

Figure 6. Recorded MQ message flow on the queue in question

Figure 6 shows the results of our MQ message flow recording. We have isolated the message flow associated with the 2033 and presented the messages in chronological order. These four messages correlate with the conversation as shown in Figure 1, with SEA101341N01 as our N-tier server and CW01 as our back-end mainframe server.

We can see the first three messages work perfectly; SEA101341N01 puts the request, while CW01 gets the request, processes it and puts the reply. Finally SEA101341N01 gets the reply. That sequence is interrupted with the fourth message. The request goes to the mainframe but the reply comes in too late; the MQGET has already failed before the reply is MQPUT!

So now we can add to our knowledge about our problem:

- It occasionally fails.
- The failure is in program pdaaggr.exe running on the Windows server.
- The problem seems to be related to an unexpected 2033 (*Message not found*) on an MQGET.
- The 2033 is not tied to a program failing or abending but rather to a response coming in too late.
- Because the reply comes in (albeit late), when the error occurs we can expect an orphaned message on the reply queue.

This gives us yet other avenues of research: Take a closer look at the orphaned messages on the reply queue to understand the business implications of our problem and take a more detailed look at the failing MQGET.

Characterization: Browse MQ queues and messages

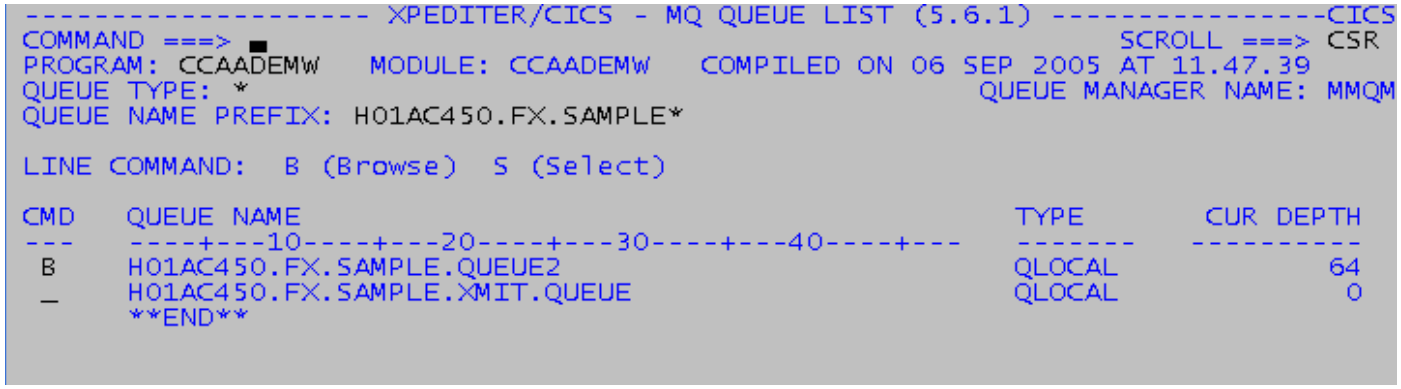

Figure 7. Browse MQ queues

The Compuware Xpediter/CICS File Utility allows you to browse queues. Figure 7 shows the selection list for the browse. One thing of note is the queue depth of 64; this implies 64 failures of our sample application (leaving 64 orphaned messages).

A particularly useful feature of this browse function is to map a message on a queue to a COBOL copybook. Because the application programmer is much more likely to be familiar with the business side of an application, this feature allows them to view the orphaned message and get a better understanding of where it fits in the business logic of the application and the business impact of the problem. Figure 8 shows an example of browsing a message on a queue and mapping it to a COBOL copybook.

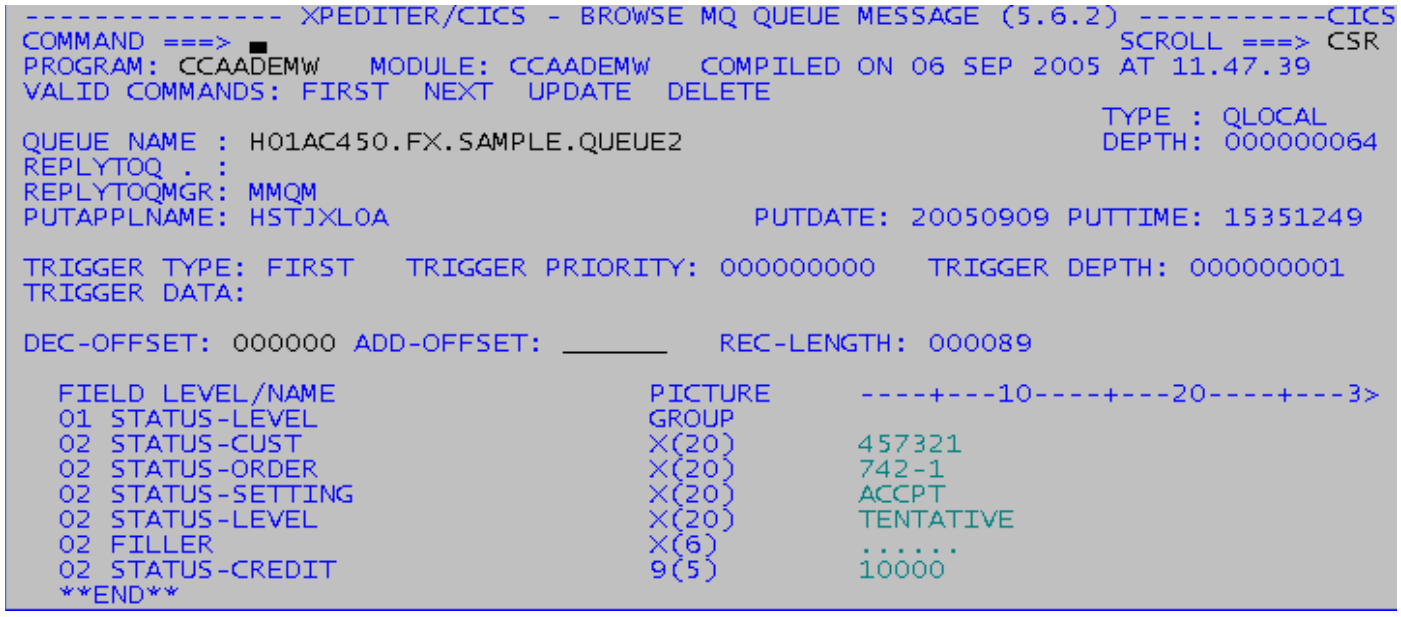

Figure 8. Map MQ message to a copybook

Tip #4. Use Xpediter/CICS to browse MQ queues and view MQ messages, including mapping the message to a copybook.

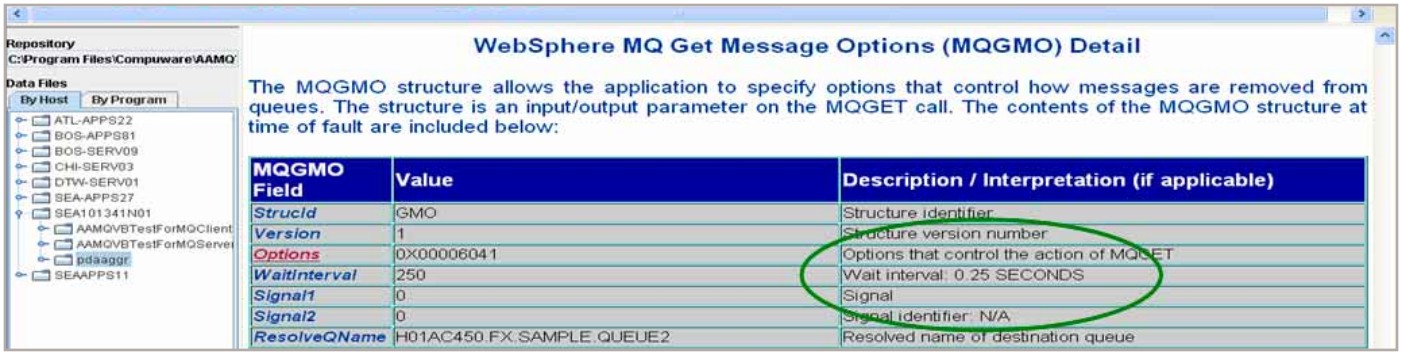

Figure 9. Get message details from the failed MQGET

If we go back and review the Get options in effect for the failed MQGET, we can see that the MQGET had a wait interval of .25 seconds. That means the mainframe has to complete the request within that elapsed time. The 2033 indicates that the mainframe application is not always capable of achieving that service level. One inclination might be to increase the wait time; in some cases, this might be an acceptable solution. But if our problem is actually a creeping performance problem, the true cause and the problem would be masked and the problem would likely occur again.

A better solution would be to research the mainframe application to determine why it occasionally fails to meet this response time requirement. But, which mainframe application? For that information we return to the message and browse the message descriptor as shown in Figure 10. Here we can see the mainframe job of interest (the job that put the message onto the queue) is HSTJXL0A.

A logical next step is to research that job to determine why it occasionally is too sluggish.

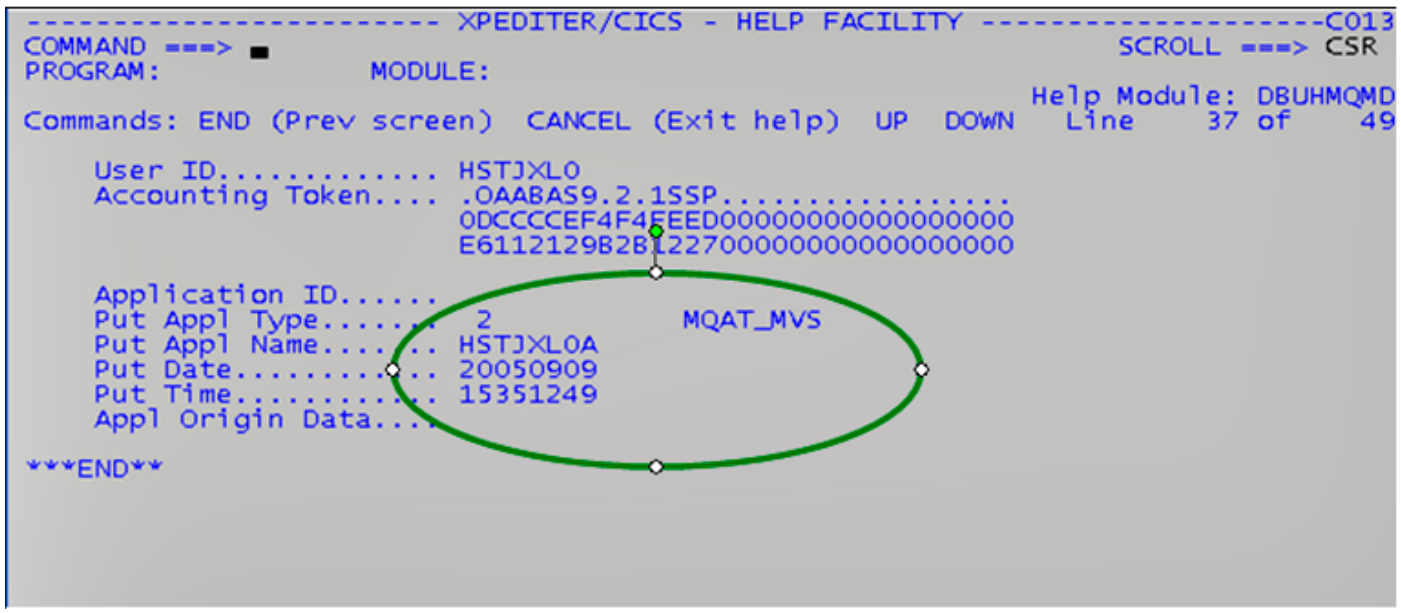

Figure 10. Browse Message Descriptor

Characterization: Analyze the performance of MQ application programs

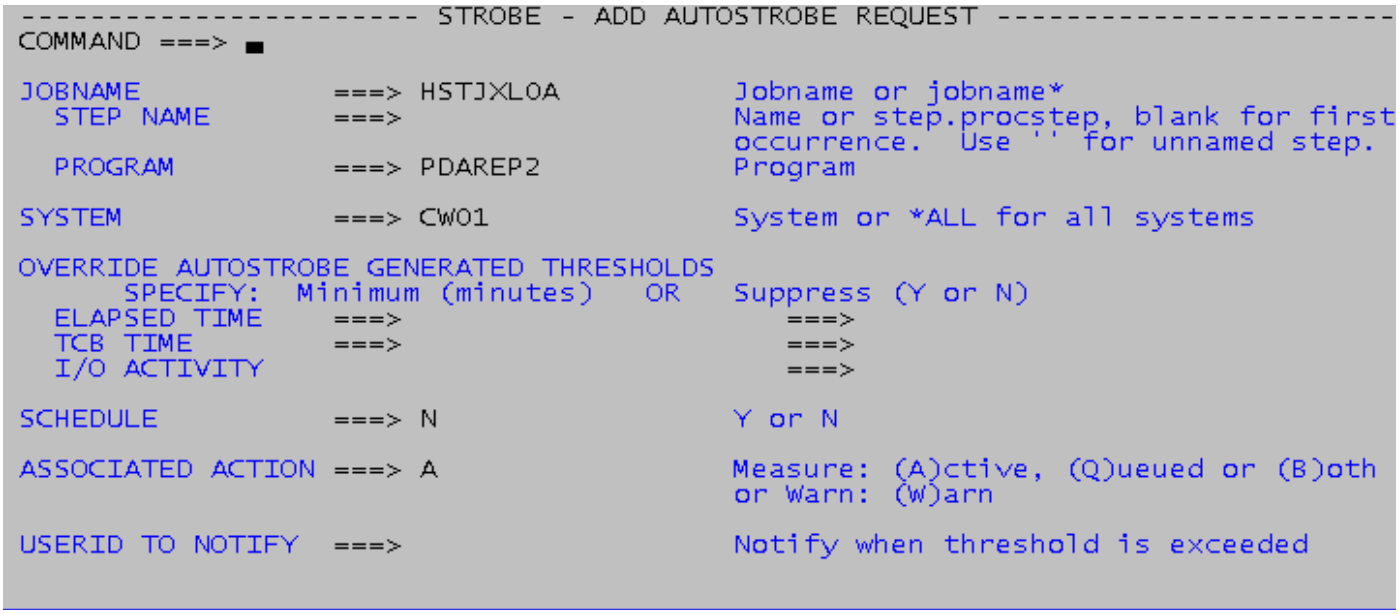

Figure 11. Set up for automatically analyzing a program, should its performance degrade

Compuware AutoStrobe provides a capability to automatically analyze a mainframe job when it exceeds its usual performance thresholds. Compuware iStrobe and Strobe *for WebSphere MQ* will then allow us to analyze the program performance exactly at the point of the excessive elapsed time issue.

One point of note is that once again this research is seamless to the application; we have not had to implement any debugging code to accomplish this analysis.

Tip #5. Use AutoStrobe, iStrobe and Strobe *for WebSphere MQ* to identify and correct performance issues within your WebSphere MQ-based applications.

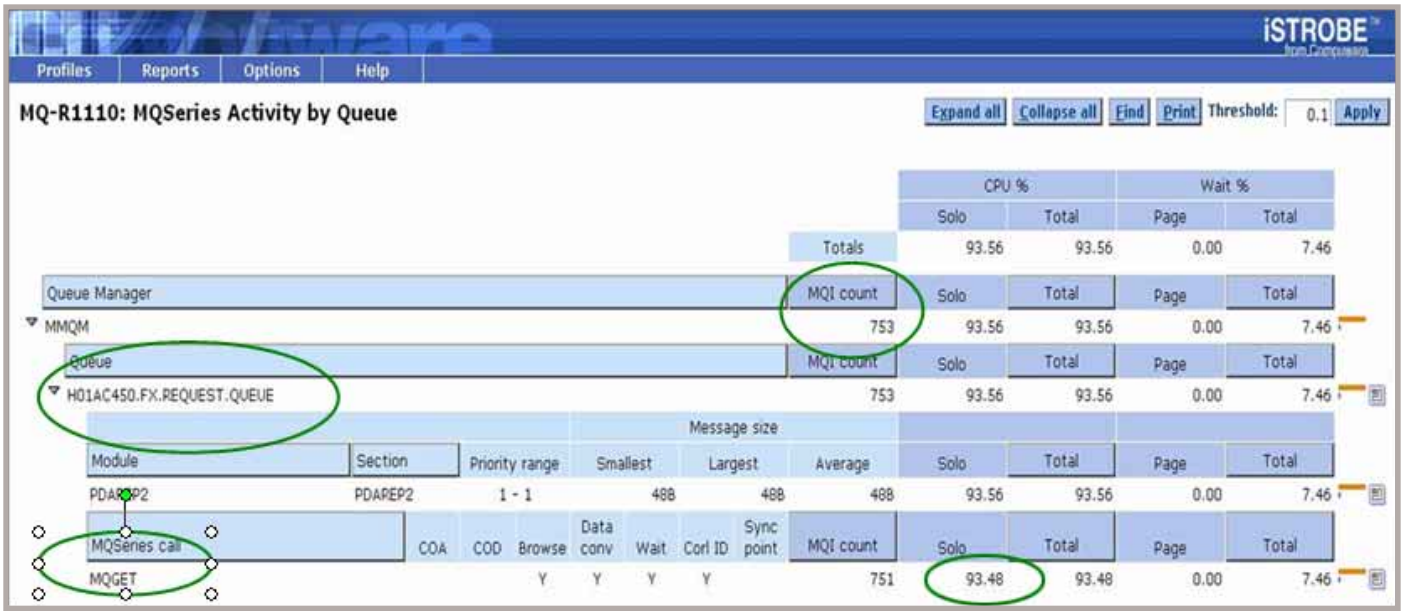

Figure 12. Activity by MQ queue

Figure 12 identifies some critical information we've gathered concerning the elapsed time issue within our mainframe job HSTJXL0A. CPU time is a component of elapsed time and it appears our issue may very well be tied to a burst of activity (notice that we've done 753 MQ calls) and that a huge percentage of our CPU utilization is tied to an MQGET to the request queue (specifically the queue H01AC450.FX.REQUEST.QUEUE). This is our first indication that while the problem symptom pointed us to the reply queue the root cause of our problem might be more closely related to the request queue.

Even more revealing is when we display CPU utilization by module as shown in Figure 13. The biggest consumer is an MQ module CSQWVCOL with over 40 percent of our total CPU usage. Compuware also provides hints as to the issue when this specific IBM module is using a lot of CPU.

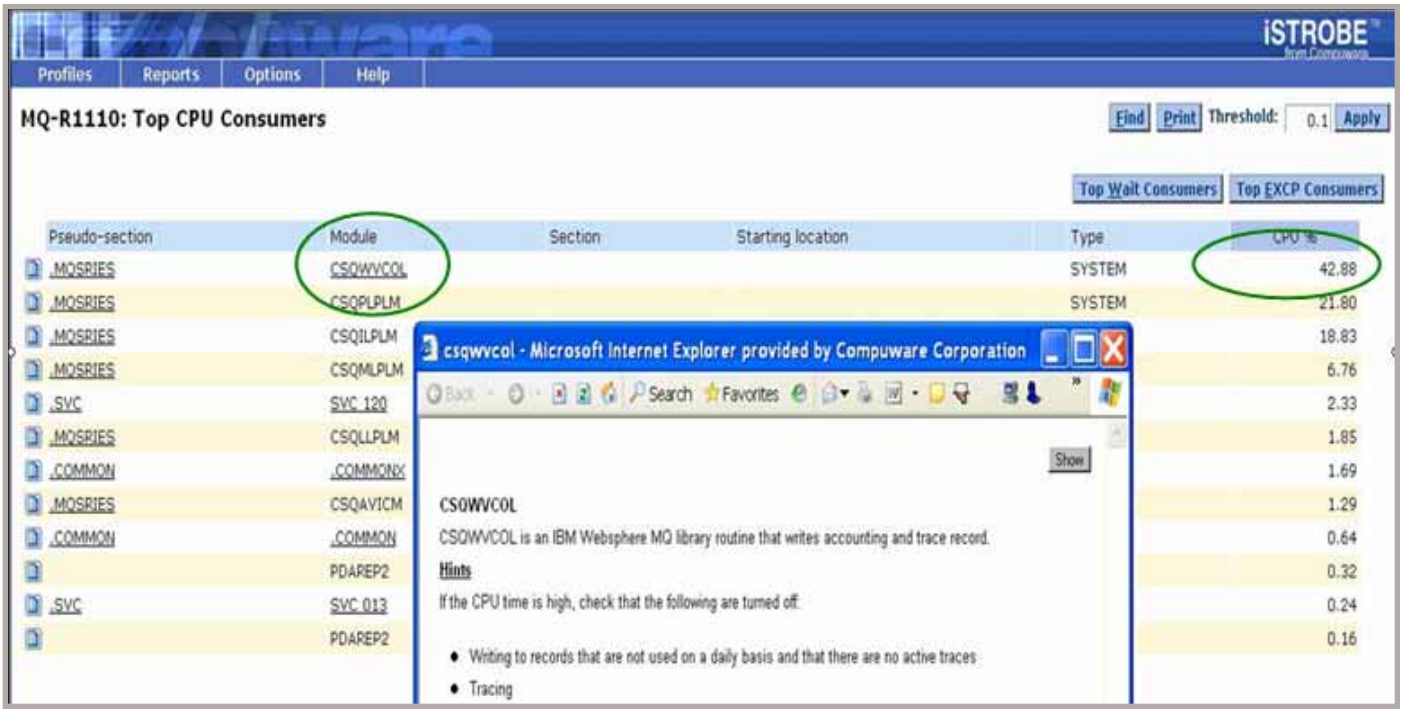

Figure 13. Top CPU consumers

The hint indicates that CPU utilization in this module is indicative of high MQ subsystem trace activity.

So now we have isolated the root cause of our problem: **system-level tracing has led to an application-level failure**. Let's take one more look at our sample application and elaborate our findings.

Resolution: Correcting the problem

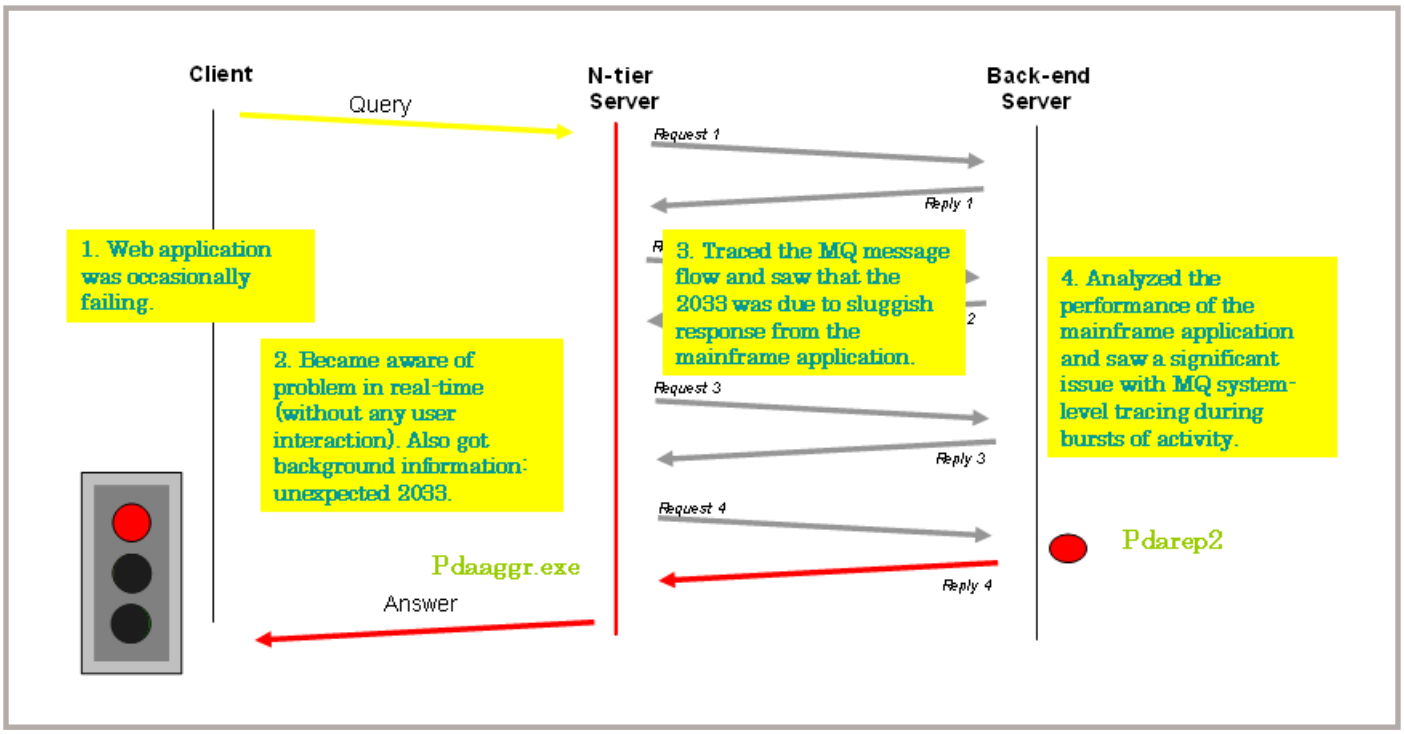

Figure 14. Our application problem revealed

Initial indications of this problem would have indicated that we were deep in an MQ application-related problem. But our research, using the Compuware application tools, has revealed the problem was tied to the amount of MQ system-level tracing on our mainframe. So the problem can be resolved without any programming changes! Let's review what we now know about our application:

- It occasionally fails; but we're still unsure as to the symptom the end-user experiences.
- The failure is reflected in program pdaaggr.exe running on the Windows server.
- The problem seems to be related to an unexpected 2033 (Message not found) on an MQGET.
- The 2033 is not tied to a program failing but instead to a response coming in too late.
- The late response is tied to a burst of MQ activity on the mainframe and the amount of MQ subsystem tracing on the system.

This opens up the discussion on both short- and long-term corrections. In the short term, we can:

1. Review the MQ TRACE settings (*Which ones are enabled? Is the resulting data reported on and used every day?*). While some accounting trace data might be essential, detailed performance data might only need to be captured for a specific performance problem and then disabled after sufficient data has been collected for problem-resolution purposes.
2. Adjust the wait time on the distributed program pdaaggr. (It is currently .25 seconds.)
3. Figure 12 also reveals we are doing the data conversion on the mainframe; it may prove less costly to do data conversion on the other platform.

For the long term, we might consider two possibilities:

1. This type of mainframe application (many short requests) might be better suited for a CICS application rather than a batch application.
2. The problem could possibly have been avoided in production all together had we better tested bursts of activity during development. Again, QACenter *for WebSphere* MQ is the ideal product for testing and load-testing MQ applications.

While this paper walks through a specific MQ example, it also highlights the benefits of the Compuware MQ tool set regardless of the MQ implementation:

- become aware of a problem the first time the problem occurs
- capture the last MQ call and MQ environment at the time of the problem
- automatically test and load MQ applications
- track MQ message flow
- reduce the CPU cost of MQ applications.

The primary emphasis here is the benefit received when one uses the right tools for the job: application tools to resolve application problems and system tools to resolve system problems. The secondary emphasis is how this approach always provides a logical next step; you're no longer reduced to guessing at the problem or adding diagnostic code to production programs.

Establishing a problem resolution process such as this allows organizations to achieve two desirable objectives: resolve problems rapidly and receive maximum benefit from their Compuware investment. Strategic use of the Compuware MQ tools—Abend-AID *for WebSphere* MQ; QACenter *for WebSphere* MQ; the Xpediter/CICS MQ File Utility; and Strobe *for WebSphere* MQ—can help sites avoid those emergency, late-night, war-room meetings by offering a well-constructed MQ problem resolution strategy.

To learn more about Compuware's support for WebSphere MQ, visit www.compuware.com

Compuware products and professional services—delivering IT value

Compuware Corporation (NASDAQ: CPWR) maximizes the value IT brings to the business by helping CIOs more effectively manage the business of IT. Compuware solutions accelerate the development, improve the quality and enhance the performance of critical business systems while enabling CIOs to align and govern the entire IT portfolio, increasing efficiency, cost control and employee productivity throughout the IT organization. Founded in 1973, Compuware serves the world's leading IT organizations, including 95 percent of the Fortune 100 companies. Learn more about Compuware at www.compuware.com.

Compuware Corporation Corporate Headquarters
One Campus Martius
Detroit, MI 48226

For regional and international office contacts, please visit our web site at www.compuware.com

COMPUWARE®
www.compuware.com