Build a Unicenter NSM Application Agent Using the Universal Agent

Integrate any application with Unicenter NSM in just one week.

BY LARRY E. PRESTON

The Universal Agent from OC Systems and its integration with Unicenter NSM provides an extremely effective and fast way to instrument applications not natively managed by Unicenter. For the IT operations group and systems administrators, the flexibility provided by this solution and its integration ensures consistent management is maintained across all applications and processes.

Out-of-the-box Unicenter management provides in-depth monitoring and analysis for Oracle, SAP, PeopleSoft, Microsoft Exchange, and other widely used software applications. When a problem occurs with one of these applications, Unicenter reports, automatically responds according to defined policy, and reports on it to your staff.

But there are many other applications, some of which may be critical to your business, that aren't currently integrated with Unicenter NSM. An undetected failure or abnormal behavior involving any of these applications impedes your ability to ensure business continuity and provide the level of performance your users require.

So, what can you do? Traditionally, you'd use the Unicenter SDK to develop a customized agent for each of the applications. Now, OC Systems is providing a new alternative—the Universal Agent—which requires no special programming knowledge, and supports integration of any application, whether or not source code is available.

In this article, I describe how to use the Universal Agent to quickly and easily build Unicenter NSM integrations for business-critical applications not natively supported by Unicenter solutions.

Decide what to monitor

The first step in building a specialized agent is determining what application information should be monitored. In some ways, this is the most challenging part. Start by defining the problem, being as specific as possible. Why do you need to bring the application under the Unicenter NSM umbrella? What behavior does the application exhibit that adversely affects your business process flow?

Continued

TECHNOLOGY
Universal Agent

Larry E. Preston is the Universal Agent Product Manager at OC Systems, Inc. With nearly 30 years of wide-ranging software development experience, Larry has been involved in virtually every aspect of the software lifecycle, from product design and programming to implementation and support. 703-359-8160 ext. 188 or lep@ocsystems.com.

Then identify what application data could be used to determine the problem has occurred. For example: You have a critical transaction processing application that occasionally becomes unresponsive. What do you need to monitor? You want to know the application is executing (i.e., has not aborted) and that transaction queries are being processed in a timely manner (e.g., in five seconds or less).

What can be monitored?

After you define the data needed to identify the problem, you must determine how to obtain that data. One instrumentation approach for an example application (let's call it ROGUE) is to write another application (let's call it PROXY), that operates totally outside the ROGUE process. PROXY will use whatever system APIs are available to obtain information about process ROGUE. Obviously, the ROGUE-specific data available to PROXY is limited.

The Universal Agent takes a different approach. Instead of instrumenting from outside a process, the Universal Agent instruments within the context of the target process. By injecting the instrumentation directly into the ROGUE process, you have access to all the internal processing flow of the application and can monitor the following data that PROXY cannot access:

- Application function entry and exit points
- Application function parameters and return values
- Runtime API invocations and their parameters and return values
- Business objects, variables, and queues
- Timing of functions, methods, or transactions

Define Resources to be monitored

The Universal Agent monitors Applications and Resources. Applications are the processes the Universal Agent monitors. Resources are the identified, categorized, and instrumented data the Universal Agent monitors. Resources can be anything, and can be defined at any desired level of granularity. This open-ended implementation supports monitoring and organizing instrumentation data in whatever manner you desire.

You should view this step as a mapping exercise, where you assign meaningful names to the raw instrumentation data you've previously determined should be monitored. Ultimately, you will invoke a simple Universal Agent API to communicate two things to Unicenter NSM: the Resource name and its current value.

The Resource name is granular, with each level of granularity identified by a dot (i.e., ".") separator. For example: You are interested in monitoring general memory usage (Resource name: "Memory"). But you want to distinguish between real, virtual, paged, and kernel memory (Resource names: "Memory.Real," "Memory.Virtual," "Memory.Paged," and "Memory.Kernel").

The Universal Agent will automatically create the appropriate managed objects within Unicenter NSM (as viewable in Node view, and propagated to all other Unicenter NSM views). For this example, you'll see a single "Memory" node having four child nodes named "Real," "Virtual," "Paged," and "Kernel."

Research what to instrument

After you identify and categorize the required application instrumentation data, you have to do a bit of research to determine the proper data instrumentation points.

The first place to look is the runtime API layer. Whether you have access to source code or not, the APIs are always visible, and can be easily instrumented. The Universal Agent exposes all runtime and system-level APIs upon which the application is built. You can instrument entry to and exit from any API, access all parameters that are passed, as well as all values that are returned.

When source code is available, you'll analyze the code to identify instrumentation points that can be exploited. However, you will often find the most elegant instrumentation points are still found in the API layer.

Code the instrumentation

Having identified the required instrumentation points and developed meaningful Resource names to represent their data, you're now ready to develop the integration code.

For Java applications, you'll write the integration code in Java. For all other cases, you'll write the integration code in simple C. In both cases, you'll use a simple Universal Agent API to send the Resource name and its current value to Unicenter NSM.

The Universal Agent performs the Unicenter NSM integration by instrumenting the in-memory image of the application. No application files (e.g., source, libraries, executable) are modified. Neither is the manner in which the application is launched. There are obvious advantages to this approach: No application-related files (including source code) are required and you avoid configuration management issues.

When you have source code

Integrating an application for which you have source code is simple and straightforward.

Figure 1 shows a Node view of the Universal Agent integration of application roulette.exe. The fully qualified Resources that are being monitored are roulette.Bet, roulette.BankTotal, and roulette.Debt. These Resources represent program variables within the application.

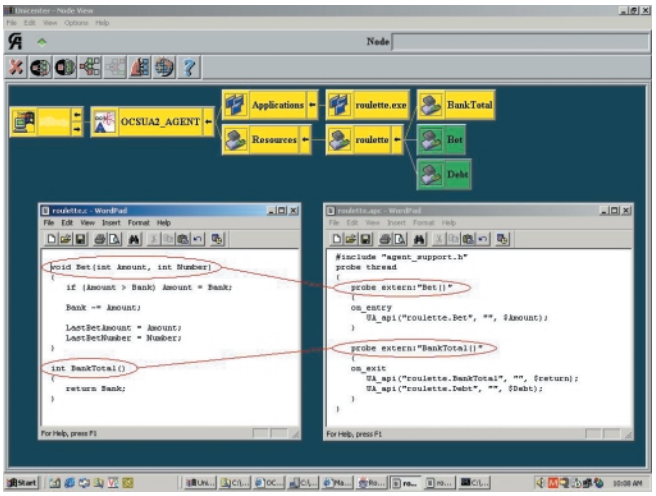

Figure 1: When source code is available—Instrumentation code, written in simple C or Java, can be inserted at any point.

At the bottom left is a portion of file roulette.c, the source code for the application. You want to extract the following program variables:

- The parameter "Amount" that is passed to function "Bet()."

Continued

- The return value from function "BankTotal()."
- Global Variable "Debt" (definition not shown).

At the bottom right are the complete contents of file roulette.apc. This is the code that was developed to instrument the application and integrate it into Unicenter NSM. Take a look at what the code does, from top to bottom:

- "probe extern:Bet()" indicates that function "Bet()" is to be instrumented. The "on_entry" clause brackets the injected instrumentation code to be executed when this function is called.
- "probe extern:BankTotal()" indicates that function "BankTotal()" is to be instrumented. The "on_exit" clause brackets the injected instrumentation code to be executed when this function returns.

The injected code that is executed in these instrumentation probes is the Universal Agent API. Let's look at the UA_api() calling sequence. The first parameter identifies the fully qualified Resource name; the third parameter specifies its value. Notice how each individual portion (separated by dots) of the fully qualified Resource name is mapped into distinct nodes in Node view.

The Resource values being passed in the UA_api() calls are as follows:

- "$Amount" is the value of parameter "Amount" in function "Bet()."
- "$return" is the value returned by function "BankTotal()."
- "$Debt" is the value of global variable "Debt," as it exists when function "BankTotal()" returns.

The Universal Agent instrumentation technology lets you inject code into your application at any location: program entry and exit, function entry and exit, even down to the source code line level. As you can see, the injected code is written as though, and behaves exactly as if, it were a part of the original source code.

When you don't have source code

Although you may have access to source code for some of your applications, it's more common to be dealing with software for which you do not have source. No-source integrations require a slightly different approach.

To integrate an application for which there is no source code, we need to be able to look inside the application to identify what instrumentation points are available. Because we can't instrument the source directly, we'll instrument the runtime APIs upon which the application is built.

The boundary layer between an application and its runtime support is always visible from within the application, and what you find here is all the support APIs the application is using. Note that you can also always instrument application start and termination, because these points are well defined—even when source code is not available.

Figure 2 shows a Node view of the Universal Agent integration of process IEXPLORE.EXE. This is the Microsoft Windows Internet Explorer application, for which source code is obviously not available. This integration monitors 22 distinct Resources (though not all are visible because the "IO" Resource node is collapsed). But let's focus on just the real memory Resources that are being monitored: Memory.Real.Size(K) and Memory.Real.PeakSize(K).

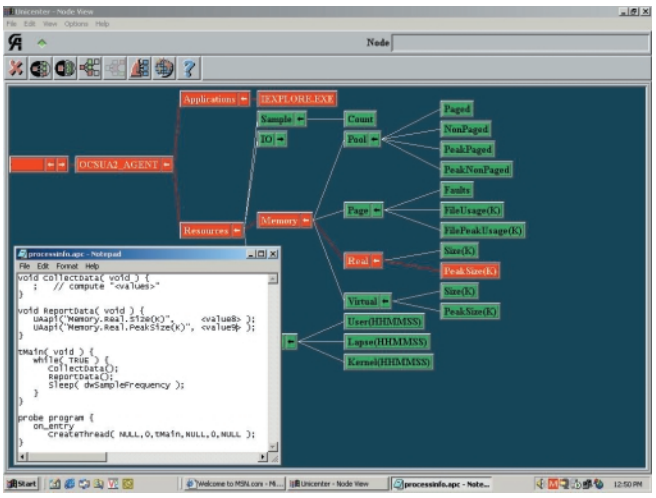

Figure 2: **No source code required**—This shows an integration of Microsoft Internet Explorer, for which source code is not available.

At the bottom left is file ProcessInfo.apc. This is a condensed, pseudo-code representation of the actual code that was developed to instrument the application and integrate it into Unicenter NSM. Only the Universal Agent APIs for the real memory Resources are shown. Take a look at what the code does, from bottom to top:

- "probe program" and "on_entry" indicates instrumentation code is to be inserted and executed once, when the application starts. The injected code (the call to "CreateThread") creates a thread that executes function tMain().
- Function tMain() simply loops, calling CollectData() to gather information and ReportData() to process the data. The Sleep() call controls the sampling frequency.
- Function CollectData() uses standard runtime APIs to obtain information about the process.

The Universal Agent instrumentation technology lets you inject code into your application at any location: program entry and exit, function entry and exit, even down to the source code level.

- Function ReportData() uses the Universal Agent API to forward the Resource data to Unicenter NSM. Note that only the APIs for the real memory Resources are shown.

Let's look at the UA_api() calling sequence. The first parameter identifies the fully qualified Resource name; the second parameter specifies its current value. Notice how each individual portion (separated by dots) of the fully qualified Resource name is mapped into distinct nodes in Node view.

The Universal Agent application instrumentation technology lets you inject code into a no-source application at program entry and exit, and runtime API entry and exit (where all API parameters are also visible). In this example, we applied instrumentation only at program entry.

Continued on page 28

Continued on page 13

No-source instrumentation techniques are useful even when you have source code for an application because it's often easier to extract the desired data by instrumenting the API layer. For example, assume you want the names of all files opened by your application. If you have source, you might instrument all of the open file instructions. However, a more elegant solution is to instrument the runtime API that's called to open all files; you use one instrumentation point instead of many. The power of this technique can be applied in many places.

Establish Resource thresholds

After your application is integrated and running under the Unicenter NSM umbrella, the next step is to establish meaningful Resource threshold values.

Use Agent view to set thresholds, which are simply a range of values for each Resource that, when breached, will propagate warning and critical events for the responsible Resource and Application. Figure 3 shows the threshold values for Resource Memory.Real.PeakSize(K); this explains why figure 2 is reporting a critical Resource event.

The Universal Agent also has a learning mode that lets you easily monitor an application for abnormal behavior. To use this feature, you allow an integrated application to execute over a period of time, while the Universal Agent automatically records the minimum and maximum values for all monitored Resources. Then you define a percentage above and below the historical range to automatically establish normal behavior threshold values for all Resources simultaneously.

Propagate alerts

When a monitored Resource value exceeds its threshold range, a warning (yellow) or critical (red) condition is propagated to all Unicenter NSM views. You can then use standard Event Management techniques to react accordingly.

For example: Figure 1 shows Resource roulette.BankTotal exceeding a warning threshold in Application roulette.exe, while Figure 2 shows Resource Memory.Real.PeakSize(K) exceeding a critical threshold in Application IEXPLORE.EXE.

Increase availability and reliability

Using the Universal Agent to build a Windows, Solaris, AIX, or Linux application agent for Unicenter NSM is a fast, simple, and straightforward process. After you've done your homework and arrived at the point where you're ready to code the instrumentation, you can usually build a prototype of the application agent in a few hours; you can typically accomplish full integration typically in approximately a week.

The speed and simplicity of the instrumentation and integration process let you bring many more applications under Unicenter NSM, thereby increasing availability and reliability for your business units. ■

Good Migrations!

Continued on page 19

A well developed plan and automated process lets technicians accomplish other objectives during the migration, maximizing efficiency. As a part of their global refresh process, Procter & Gamble migrated an international office's 250 systems with four technicians in two days.

In a large scale upgrade or refresh, you'll want to migrate numerous systems simultaneously. It's imperative to have a way to identify different machines or systems within the process you develop. To accomplish this, the machine name, IP address, phone extension, user name, or ID number could be used as the unique identifier. This identifier could be automatically determined as part of the process—such as reading the registry for the machine name—or could be entered by the technician or user. This identifier ensures migration files and logs are associated with a particular system.

A well-developed automated process can also provide added flexibility like enabling the technician to customize the migration for a particular user or system, while not allowing them full access to the migration program. Enabling parts of Desktop DNA's interface to be used to select or deselect certain elements of a user's PC DNA may be key to managing the unique situations you can encounter during a large-scale change initiative. For example, it may be beneficial to let a technician or user add certain files to the migrated PC DNA but not allow them to modify the data searching setup.

Manage the migration

After you've determined what you're going to migrate, identified the tools, and developed the process, you're ready to execute the migration. Desktop DNA's comprehensive logging, reporting, and diagnostic capabilities, coupled with the enterprise management functionality of Unicenter, let you manage the execution of your migration, assess progress, and verify the success. In the end, you'll have automated the successful transfer of users' PC DNA in concert with your critical change initiative.

Unicenter and Desktop DNA are committed to helping you leverage your existing IT assets and run your organization as cost effectively as possible. By taking advantage of the integration of Desktop DNA and Unicenter Software Delivery (SD), you can not only leverage existing IT investments, but implement a PC DNA management strategy that supports the ongoing reduction of the complications and costs associated with change.

Because you aren't a meteorologist who can get away with just reporting the unpredictable, take advantage of the integration of Desktop DNA and Unicenter SD to realize the benefits of the most effective change management solution available.

Miramar and Computer Associates continue to work closely with one another to complement and extend the Unicenter and Desktop DNA offering. You can obtain further information about Miramar Systems and Desktop DNA at ca.com and miramar.com. In addition, Miramar will be demonstrating the integration at CA World 2004.

For a detailed white paper about the integration of Desktop DNA and Unicenter SD, you can send an e-mail message to gcole@miramar.com and ask for the CA plus Desktop DNA white paper. The white paper is also available from your CA representative and in USD 4.0 SP 1. ■