Realtime publishers

The Five Essential Elements of

Application Performance Monitoring

Don Jones

sponsored by

QUEST SOFTWARE®

Introduction to Realtime Publishers

by Don Jones, Series Editor

For several years now, Realtime has produced dozens and dozens of high-quality books that just happen to be delivered in electronic format—at no cost to you, the reader. We've made this unique publishing model work through the generous support and cooperation of our sponsors, who agree to bear each book's production expenses for the benefit of our readers.

Although we've always offered our publications to you for free, don't think for a moment that quality is anything less than our top priority. My job is to make sure that our books are as good as—and in most cases better than—any printed book that would cost you $40 or more. Our electronic publishing model offers several advantages over printed books: You receive chapters literally as fast as our authors produce them (hence the "realtime" aspect of our model), and we can update chapters to reflect the latest changes in technology.

I want to point out that our books are by no means paid advertisements or white papers. We're an independent publishing company, and an important aspect of my job is to make sure that our authors are free to voice their expertise and opinions without reservation or restriction. We maintain complete editorial control of our publications, and I'm proud that we've produced so many quality books over the past years.

I want to extend an invitation to visit us at http://nexus.realtimepublishers.com, especially if you've received this publication from a friend or colleague. We have a wide variety of additional books on a range of topics, and you're sure to find something that's of interest to you—and it won't cost you a thing. We hope you'll continue to come to Realtime for your educational needs far into the future.

Until then, enjoy.

Don Jones

Copyright Statement

Chapter 1: In Application Performance, the End User's Experience Is King

Application Performance Monitoring (APM) is a complex set of disciplines designed to give you accurate details on how your business applications are performing. Many businesses rely on APM to tell them whether their internally-developed applications are performing well, and many extend their monitoring to include third-party line-of-business applications. The ultimate goal of APM is to tell you whether an application that is supporting the business is slowing it down, and to provide you with tools and direction for solving application performance problems.

Historically, however, there's been a problem with APM. APM is, after all, something that is utilized by a company's IT staff, and IT often has the wrong perspective.

APM: The Infrastructure Perspective

IT's job is to maintain the business' technology investments. Traditionally, that has included things like the network and its attendant hardware—switches, routers, and so forth—as well as servers, power supplies, data center racks, and so forth. IT also helps maintain the business' client computers, but they don't tend to focus much on what the company's end users are actually *doing* on those client computers. As far as most of the IT team is concerned, IT is about hardware and what's in the data center.

APM of the Past: Performance Is in the Data Center

In the past—and, frankly, in most of today's companies—APM is something that happens in the data center. To measure an application's performance, you measure the performance of its individual parts. You might, for example, measure:

- Network bandwidth and latency
- Database servers' processor utilization
- Middle-tier or application servers' memory utilization
- Application server disk access speeds

A sophisticated business might also be able to probe the performance of individual application components. For example, one way to do so is to have an automated process submit sample database queries directly to the database server, then measure the time it takes to complete that query.

There's nothing inherently wrong with any of these measurement approaches, but they do reflect a very infrastructure-focused perspective. In other words, you're not terribly concerned about what's happening on the end user's computer. Many businesses, in fact, have set up service level agreements (SLAs) that reflect performance goals for individual components: The database must process queries in so many milliseconds, the network must have no more than such-and-such latency, and so on.

Even software developers—who, by definition, are focused on the application—tend to worry more about what's going on in the data center. Developers are commonly working on powerful client computer hardware, and rarely working with back-end servers that are operating under a production load. So they run performance tests designed to measure the length of time a query takes to complete, or the amount of time a particular software component takes to process something and return a result.

In many cases, users end up using their applications in ways that those applications' developers never anticipated—or tested for. Sometimes those unexpected usage patterns can become the most important things the users do with the application, and because those patterns weren't tested during development, they may contribute an inordinate amount of negative performance perception to the application.

Here's the problem: Even companies who have mastered this infrastructure-centric approach to performance *still* have users who complain that their applications run slowly. How often have you been on the phone with a major company and heard, "I'm sorry, it'll be just a moment, I'm waiting on the computer." It's become such a common phrase that our children will probably create a shortcut for it, and you'll just hear "slow computer" from the phone representative, and everyone will know what they're talking about.

Clearly, the infrastructure-centric approach to APM isn't doing the job we need it to do. Despite our SLAs and monitoring efforts, our businesses still aren't functionality the way we want them to because of poor application performance. Why is that?

An Application Is Less than the Sum of Its Parts

Back in the day, applications consisted of one thing: a single, monolithic piece of software running on a single computer. As we started to develop distributed computing, our applications became distributed: back-end database, middle-tier server, client application. Today, applications consist of multiple components that are even shared *between applications.* A Web-based application, for example, might run on a Web server that's also running another application. Our applications have grown from being single, easily-monitored units into complex *systems.*

Take an automobile: You can design a fast engine, a great drivetrain, and low-friction tires, but when you assemble it all into an actual car, you might be less than pleased with the performance. Systems—including our modern applications—can be *less* than the sum of their parts. In other words, good performance observed in every single component does *not* necessarily guarantee good overall performance of the application as a whole. You can apply all the SLAs you want to individual pieces of the application, but you're not meeting the ultimate business need of having an *application* that performs well.

You can probably see where I'm going with this: We need to start focusing a bit less on individual component performance, and start looking at the performance of the entire *system.* And we need to do that from a *business* perspective, which means from the users' perspective.

The Danger of the Data Center View

That said, we *need* the data center perspective. If the application isn't performing well, it's because one or more components isn't performing well. That infrastructure perspective is useful in finding the root cause of a problem; it's just not useful in telling us whether the overall *application* is healthy.

The danger of relying *solely* on the data center-centric view is that things can look perfectly fine in that view, but be absolutely horrible out in the real, production world. Again, consider that phone call where you're told, "The computer is slow." When that happens, do you suspect that the company you've called has people running around the data center, gnashing their teeth and looking for the cause of the problem? No. They have people sitting in the data center, calmly pointing to their performance meters, saying, "We're meeting our SLA for performance," even though the end users would completely disagree.

Without losing those data center skills, we need to change our perspective.

Application Performance Monitoring: The End User Perspective

When it comes to measuring the overall performance and health of an entire application system, there is only one perspective that matters—that of the end users. Think about it: What goes on in the data center isn't the company's *business.* It's what the end users are doing that matters to the company. They're the ones *conducting* the company's business; the data center is simply supporting them in their efforts. If the end users think an application is slow—well, then it's slow. It doesn't matter how fast the database processes queries.

The 5D Framework: Performance Is in the Eyes of the Beholders

In a recent Gartner report entitled "Magic Quadrant for Application Performance Monitoring," author Will Cappelli outlined a five-dimensional model for APM. These contribute to what I call a "Five-Dimensional Framework" or "5D Framework," or approach, for more modern and beneficial APM. The five dimensions are:

- End-user experience (EUE) monitoring
- User-defined transaction profiling
- Application component discovery and modeling
- Application component deep-dive monitoring
- Application performance management database

EUE monitoring is listed first for a reason: It's the most important high-level metric you can have for APM (we'll cover the other four dimensions later in this chapter). The EUE tells you, literally, what your end users are dealing with. If the "computer is slow today," you'll *know* because you're monitoring what your users are actually encountering in the real world. It doesn't matter whether all your back-end metrics look fine—one or more of them are obviously not good enough if the end users are seeing a slow application.

Why the EUE Is All That Matters

Let's think about a slightly different type of application. Consider a Web application that runs on the public Internet—perhaps an e-commerce application such as Amazon.com.

If your approach to APM is sitting in the data center and monitoring database response times and application server performance, you're completely missing out on the single most important measurement to your business: Whether users can quickly check out and submit their payment. Ultimately, the business survives on users being able to complete their purchases, and if that process is slow, many users will give up and go someplace else. That won't be reflected in your database performance—you *have* to have a way to measure the actual experience of your end users. You have to know that the checkout process only slows down when the user is typing in their credit card number, and not because something is slow on your end.

End user experience = business. Period.

Elements that Contribute to the EUE

So what contributes to the experience your end users have with an application? Generally, *every single component of the application*—including components that you may well have overlooked.

For example, one common cause of end user performance complaints is the end users' own client computers, which may be running less-powerful hardware than that which was used to develop and test the application. Network latency can be another common cause for slow applications—although the network itself is often overlooked as a component of an application. One of the reasons that APM can be so complex is the sheer number of individual components involved. Consider a fairly simple Web application, and its components:

- The end user's Web browser, which may have to execute client-side scripts and other code
- The network connections between the user and the Web server
- The Web server hardware (or, in the case of a virtual server, the virtualization host and *its* hardware)
- The Web server software, such as Apache or IIS
- The application code, which may be ASP.NET, Java, PHP, or something else
- The connection between the Web server and the database server

- The database server hardware, including its storage subsystems
- The database server software, such as Oracle, MySQL, or SQL Server
- The database code—queries—that are executed on the server

Even that isn't a complete list. Those "network connections" may consist of multiple routers, switches, proxy servers, firewalls, and more, all of which can contribute to poor application performance.

This complexity is ultimately why the EUE is such an important metric. It would be very difficult to continuously monitor *each* of these components manually, and to correlate component-level performance with the performance of any given application—especially because many of these components would be shared by multiple applications.

Measuring the EUE

So how do you go about measuring the EUE? A conceptually simple approach might be to install a piece of agent software on a few end user client computers, and have it measure the time specific tasks take to complete. It could report that data back to a central server for correlation. Unfortunately, "conceptually simple" doesn't translate to "easily implemented;" very few APM systems rely on this kind of agent software. Actually making that work would be pretty tricky, and it would be impossible or impractical in some situations—like when you're trying to monitor the EUE of Web customers. They're not likely to let you install agent software on their computers, after all! Instead, let's define a few goals for EUE monitoring, and define—or redefine—an important term.

Goals for EUE Monitoring

If we're going to monitor and measure the EUE, what is it that we expect to achieve? Primarily, we want to measure the time it takes to complete specific key tasks, excluding any time it may take the user to provide input (we don't necessarily want to measure the speed with which they type). In some ways, you can say that we want to measure the time a user spends waiting on the application, rather than interacting with it. Our measurements need to include all the time starting from the user's final interaction—such as clicking a "submit" button—all the way to the time when the user can begin interacting again—such as when a screen of information is displayed for them. Our measurements should therefore include *everything*—the time needed to transmit data from the client application all the way to the time needed to transmit any response and assemble it on the screen for the user to see.

Re-Defining "Transaction"

Perhaps the most important concept in EUE monitoring—and in fact, in APM in general—is the idea of a *transaction.* In the data center-centric view of the world, a *transaction* consists of several independent operations that have to be performed *atomically,* meaning either *all* of the operations are completed or *none* of them complete. It's a common term in relational database management systems, for example.

In APM, *transaction* has a vaguely similar meaning. It does consist of a number of independent operations or tasks, but in APM, the idea is that the operations are all part of some real-world task. Clicking the "checkout" button on a Web site, for example, is the beginning of a transaction that is finished only when the "completed" screen is shown to the user. These *user transactions*, as they're sometimes called, may extend over a significant amount of real-world time—several minutes, perhaps—and may consist of several discrete steps.

You define what a user transaction looks like in your applications. For example, in an order-entry application, a transaction might start when a sales representative clicks a button to submit an order, and might finish when the completed order number is displayed to the representative.

Note

It's a common practice to define transactions in such a way that they do not span user input. That is, the transaction starts when the user is finished providing input and ends when more user input is needed. That way, slow-typing users aren't perceived as a slow application. However, it may also be common to "wrap up" several of these transactions into a larger task—such as completing a multi-step checkout process in an online store.

Here's a form definition of *transaction* from the Gartner paper:

> …sequences of user activities and system responses that are perceived by the user to be a single, logical unit of work; this only rarely corresponds to a transaction or logical unit of work in the traditional sense of a consistent change of database state.

Perceived by the user is the operative phrase, here: A transaction in the APM world is something that the end user will consider to be a single unit of work. Again, the end user perspective is king.

Measurement Techniques

We know what we're measuring, and we know why we're measuring it. Now we just need to figure out *how* we're going to measure it. This is where commercial APM solutions come into play, and it's where they have some of the most differentiation. Although there are a few common approaches—which I'll cover in the next sections—not every APM solution offers every one of these. Some of these techniques are only applicable to specific types of applications, while other techniques are generally more effective than others. As you start to review APM solutions for use in your environment, understanding how they measure the EUE is one of the first things you should ask about. Also make sure you understand *why* a solution uses a particular measurement technique: Most solution vendors have invested considerable time and effort in their approach, and understanding why they've chosen that approach can be very helpful when comparing potential solutions.

Robots

The first technique is to create a robot—that is, a piece of automated software—that behaves like an end user. Conceptually, imagine a piece of software that actually uses your application, clicking on things and entering data according to a pre-written script. In fact, solutions exist that can do exactly that, and they're often used in software testing and quality assurance. In APM, robots don't necessarily need to *use* the application. After all, you're typically not measuring how fast the user is typing—you start measuring once the user *submits* something to the application. So an APM robot can simply submit a pre-designed set of data *as if* it had been typed into a client application. These so-called *synthetic transactions* allow the robot to probe your entire application's performance as a single unit, by simply measuring the time it takes to get a completed response back from the application.

Some APM solutions may even allow you to capture real-world traffic from users and use that to pre-load a robot so that the robot is using "real" data. In many cases, you'll modify the data in some fashion so that you can easily pick it out of your back-end databases and delete it, invalidate it, or take some other action to separate it from the actual production data in the system.

Which brings up an important point: *You have to measure EUE on your production systems.* That means you will, to a degree, be mingling test data with real data; that's unavoidable, and APM solutions can include tools to help you deal with that. But you *cannot* accurately measure the EUE by injecting synthetic transactions *into a test system*. The EUE will vary based on your application's current workload, so you need to test the EUE on the real, production-loaded system.

Packet Capture and Analysis

A similar, and potentially less-intrusive approach, is to capture the network traffic of transactions in process. This can be implemented in the form of a software agent that runs on your network, or even as a dedicated appliance that attaches to your network. These systems don't necessarily attempt to capture and monitor every transaction passing through your system, although you could potentially do so. However, in many cases, these systems capture traffic and look for transactions with specific characteristics—such as a certain transaction amount or other detail. This enables the systems to analyze a *sampling* of transactions, which may be real end user transactions or could be synthetic transactions injected into the system solely for measurement purposes.

The approach here isn't all that different from a robot, although rather than generating transactions and waiting for the results, the system simply measures the time it takes to complete transactions that are happening anyway. These systems can also provide more granular detail, perhaps by also capturing traffic between your back-end systems. That's moving beyond the realm of EUE, however, so we'll save that discussion for later.

Note

Gartner's report notes that packet capture and analysis is used by some of the more-effective APM solutions currently on the market.

Endpoint Instrumentation

This is a classic approach to APM: Installing measurement agents directly on endpoints. That doesn't *necessarily* include the end user's computer, although it could; you can just as easily measure a user transaction from the back end as you can from the front end—although there may need to be some adjusting or additional measuring in order to also capture things like network latency between the end user and the back end.

Note

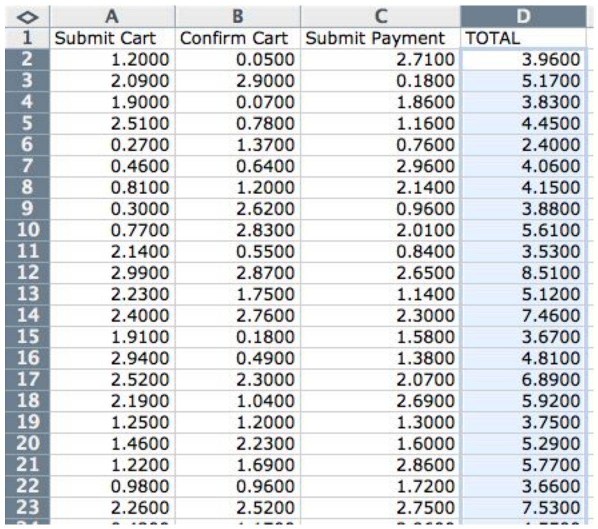
It's especially easy to forget about, or ignore, the network and other "physical" application components. In some scenarios, those components may not contribute any significant impact to the application and *can* be safely ignored. In others, they *do* impact the application and should be considered. APM solutions that have the ability to capture, or can be extended to capture, some information about the physical infrastructure can be very useful. That also includes *virtual* infrastructure components, as more and more environments adopt virtualization in various application layers.

Special-Purpose Systems

Finally, there's a whole class of techniques designed to deal with very specific applications, such as Voice over Internet Protocol (VoIP) transactions or other complex, multi-stage IP-based services. These are measurement systems you consider only if you have a specific need to do so, and in many cases, you will be looking at specialized APM software and hardware to accomplish your EUE monitoring goals.

Visualizing the EUE

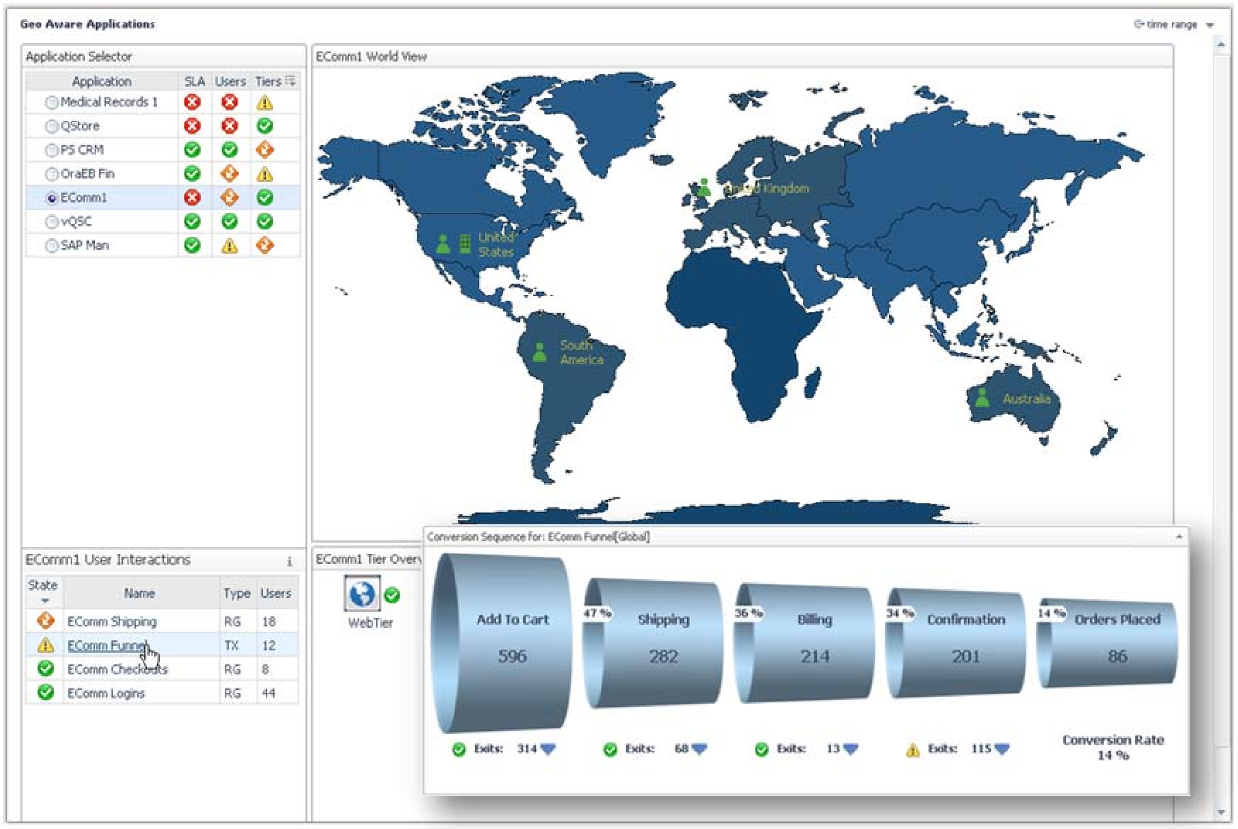
When the EUE monitoring numbers start rolling in, how will you use them? One obvious solution is to plug the numbers into a spreadsheet and start looking for trends, highs, and lows. Figure 1.1 shows what that might look like for a sample Web application.



◇	A	B	C	D
1	Submit Cart	Confirm Cart	Submit Payment	TOTAL
2	1.2000	0.0500	2.7100	3.9600
3	2.0900	2.9000	0.1800	5.1700
4	1.9000	0.0700	1.8600	3.8300
5	2.5100	0.7800	1.1600	4.4500
6	0.2700	1.3700	0.7600	2.4000
7	0.4600	0.6400	2.9600	4.0600
8	0.8100	1.2000	2.1400	4.1500
9	0.3000	2.6200	0.9600	3.8800
10	0.7700	2.8300	2.0100	5.6100
11	2.1400	0.5500	0.8400	3.5300
12	2.9900	2.8700	2.6500	8.5100
13	2.2300	1.7500	1.1400	5.1200
14	2.4000	2.7600	2.3000	7.4600
15	1.9100	0.1800	1.5800	3.6700
16	2.9400	0.4900	1.3800	4.8100
17	2.5200	2.3000	2.0700	6.8900
18	2.1900	1.0400	2.6900	5.9200
19	1.2500	1.2000	1.3000	3.7500
20	1.4600	2.2300	1.6000	5.2900
21	1.2200	1.6900	2.8600	5.7700
22	0.9800	0.9600	1.7200	3.6600
23	2.2600	2.5200	2.7500	7.5300

Figure 1.1: Reviewing EUE response times.

That's not a very effective way to look at things, though. Humans don't work as well with numbers as they do with visual representations—that's why Excel comes with charting capabilities. A good APM solution, however, won't make you turn to Excel to construct your own charts. Instead, it will provide its own visualizations of EUE, such as the example shown in Figure 1.2.



Application Selector

Application	SLA	Users	Tiers
Medical Records 1	✖	✖	⚠
QStore	✖	✖	✓
PS CRM	✓	✓	◆
OraEB Fin	✓	◆	⚠
● EComm1	✖	◆	✓
vQSC	✓	✓	✓
SAP Man	✓	⚠	◆

EComm1 User Interactions

State	Name	Type	Users
◆	EComm Shipping	RG	18
⚠	EComm Funnel	TX	12
✓	EComm Checkouts	RG	8
✓	EComm Logins	RG	44

Figure 1.2: Visualizing the EUE.

That's a much more efficient and effective way to review your EUE data. In fact, because these visualizations can be *specific* to the EUE, they can be much more effective than even a great Excel chart. For example, you might be able to see a map, showing average EUE times from various regions—with unacceptable values shown in an alternative color to attract your attention. Or you might view an entire series of transactions that make up a larger process—such as a Web-based checkout process—showing end user wait times for each discrete step in the process. Color coding and predefined thresholds can help draw your attention to steps that aren't performing as quickly as you need them to, helping direct top-level attention to problem areas and enabling immediate follow-up action.

Visualizations are *key.* In fact, as you evaluate APM solutions, ask to see examples of the visualizations each one offers. Effective visual displays that can focus your attention and drive troubleshooting are one of the main things you're paying for in an APM solution, and the types of displays offered by each solution will be one of the major competitive differences that you need to consider.

Re-Defining the SLA

With all of the previous discussion in mind, you should already be thinking that your concept of an SLA is going to need to change. Ideally, your SLAs will represent the experiences of the end user.. "When a user clicks Submit, the next screen should display within 3 seconds," for example. *That's* what you manage to, and that's a number you program into your APM solution so that it can help you monitor and manage to that response time.

So what about your old, data center-focused SLAs? They become less important as performance contracts, but they can remain important as back-end performance *thresholds.* That is, if you know a database response time of .5 seconds is needed to support an EUE of 3 seconds, you can program .5 seconds as a threshold. When the EUE starts to turn yellow in your APM solution, you can dig a little deeper If the database performance is nudging over that .5 second threshold, then that might be where you start troubleshooting the problem. In other words, your old data center-focused SLAs become *symptoms* of a problem rather than the problem themselves.

How the EUE Drives 5D APM

To summarize the main points so far:

- The EUE is what really matters to the business. All the underlying metrics simply support that EUE.
- The EUE is what you *manage.* It's the goal. It's what you look at every morning and several times throughout the day; it's what you want to be alerted about when it isn't what you want it to be.
- In APM, problems flow *up* the application stack. In other words, a deep-down problem with an application component will ultimately be reflected in the EUE. Causes, however, flow *down* the stack: You start with the EUE, then dig deeper until you find the root cause.

EUE, then, drives the entire 5D approach to APM. When the EUE looks good, you don't worry about what's happening under the hood—you're meeting your goals. When the EUE doesn't look good, you use the other four dimensions of the 5D approach to determine the root cause. In fact, the entire function of the 5D approach is to help you quickly drill down to the root cause. Ideally, you'll see your EUE begin to decline *before* it becomes entirely horrible, and you'll dig into the root cause and solve the problem before users become severely impacted.

Let me share a brief example from my past to illustrate the importance of this top-down approach.

> I used to work on an IT team that supported a complex, multi-tier application. Invariably, under heavy workload, the system would start to slow down and users would complain. When things (that is, complaints) got bad enough, we would have to take action.
>
> "Taking action" usually meant all of us jumping on whatever our particular piece of the puzzle was. I pulled out my server performance monitors and starting looking at CPU, memory, network, and disk performance. The DBA started looking at database performance. The application developers started running test transactions against their middle-tier components. The desktop support guys would pick a client computer and start measuring its processor utilization.
>
> Equally invariably, all of us came to the conclusion that "our" bit was performing fine, and that the problem must lie elsewhere. Unfortunately, since we'd *all* come to that conclusion, there was no place else to look—so we'd sit in a conference room and argue, pointing fingers at each other.

That's the "bottom up" approach. The EUE might informally drive effort, but the effort starts at a detailed, granular level, and it isn't specifically driven by any measured EUE. By measuring that experience, however, the top-down 5D approach lets us start at the top and dig to the *root causes of measured, observed problems.* We're not guessing—we're solving problems.

The 5D Approach to APM

This "5D approach" I've been talking about comes directly from the five APM dimensions defined by Gartner. Understand that these dimensions are really a model, or framework; the real-world implementation of this approach may not necessarily equal five different products that you have to buy, or not even necessarily mean five aspects of one product.

The Five Dimensions of 5D

These five dimensions can best be thought of as *capabilities*—things you need to be able to do in your environment. Gartner says:

> …the first generation of application performance technology treated applications as just another set of components of the IT infrastructure. Thresholds were set in advance by vendor or user… [and] polling agents were periodically put in place to determine whether those thresholds were close to getting crossed or, worse, had been crossed….

However:

> In a world of highly modular, highly distributed, volatile and fuzzy-edged applications, the number of agents that would be required to deliver a holistic view of application performance would likely cripple the performance of the applications being monitored. Furthermore, the interactions among the various modules of modern applications have become so complex and multidimensional that it is likely that valid inferences from local to global application states would be almost impossible to carry out with any regularity.

That's why APM vendors have slowly developed this five-dimensional approach that I'm referring to simply as the "5D approach." This approach deals with the complexity of modern application architecture. The first four dimensions capture specific views of application behavior, while the last dimension correlates and analyzes the data generated by the first four—which often consists of extremely large sets of data.

EUE

This is the dimension we've covered in this chapter: Tracking the actual experience of the end user who is operating the application and executing user-level transactions. The order of these dimensions is not accidental: The order represents a top-down approach, starting with the most-visible element of the application and the element that is most important and impactful to the business.

User Transaction Profiling

Next is user transaction profiling, which involves following specific user transactions—such as submitting a form or completing a checkout—in more granular detail. Here, we're not concerned so much with the total transaction time-to-complete—that's what the EUE is all about—but rather are concerned with what bits of the application are spending time processing the transaction. The idea is to get a general sense of where time is being consumed, and to determine which broad layer of the application is consuming enough time to push the EUE number over its threshold of acceptability.

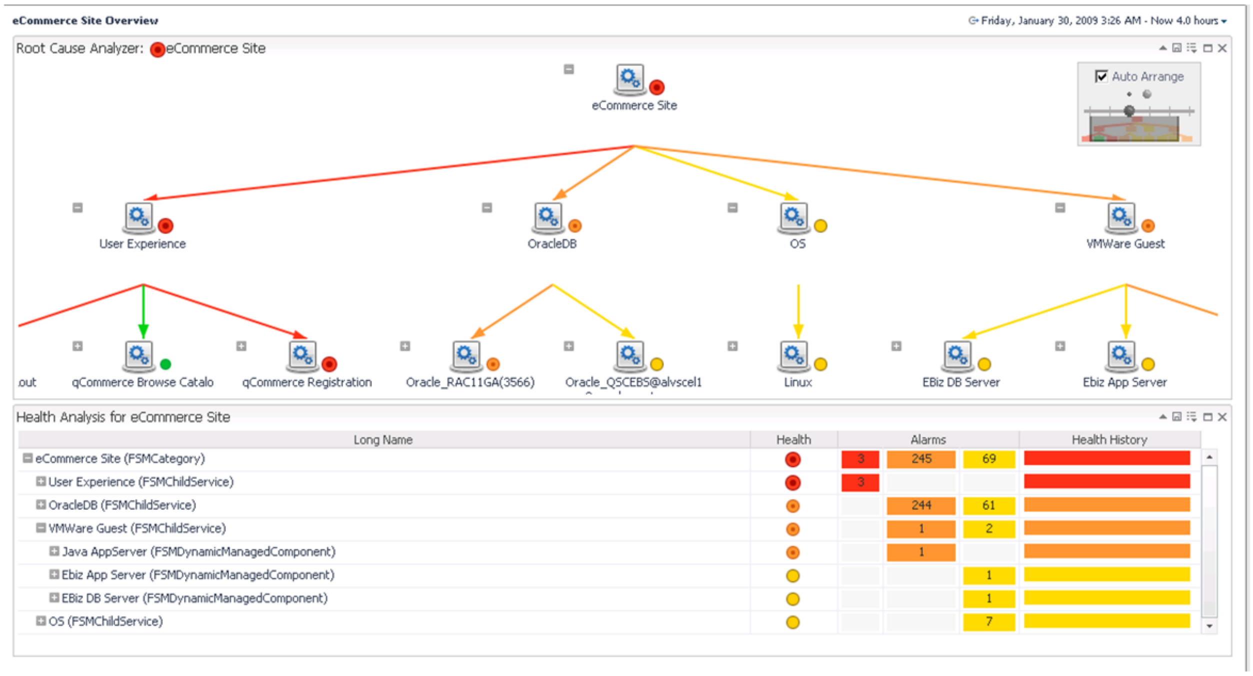
> **Cross Reference**
> Chapter 2 will focus primarily on user transaction profiling.

Application Component Discovery and Modeling

The next dimension is designed to discover the specific components that contribute to an application's performance in any way. This may include both hardware and software components. For example, a database server may be broken down into components like processor, memory, disk, query execution time, and so forth. The level of granularity will depend on the APM solution being used.

It's impractical, given today's technologies, to try to automatically discover an entire application and create a model, or map, of all its components. In practice, discovery often serves as a starting point, and manual modeling takes over from there. That permits you to construct a very detailed model of what your application looks like, and what components are involved so that you have a basis for further troubleshooting. This model is represented visually within an APM solution. Figure 1.3 shows an example of what the visual map might look like.

Cross Reference
Chapter 3 will focus entirely on this dimension of the 5D approach.



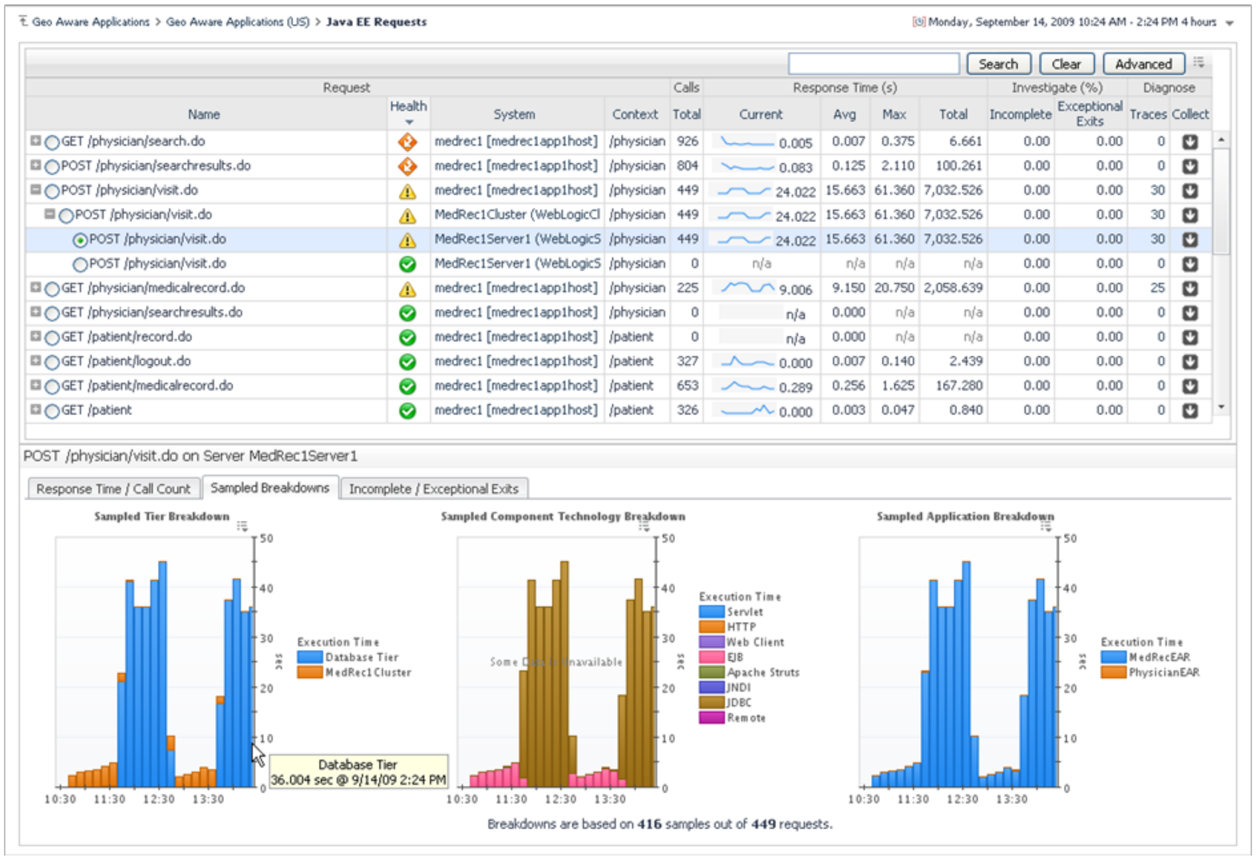
Long Name	Health	Alarms			Health History
eCommerce Site (FSMCategory)	●	3	245	69	🟥
User Experience (FSMChildService)	●	3			🟥
OracleDB (FSMChildService)	●		244	61	🟧
VMWare Guest (FSMChildService)	●		1	2	🟧
Java AppServer (FSMDynamicManagedComponent)	●		1		🟧
Ebiz App Server (FSMDynamicManagedComponent)	●			1	🟨
EBiz DB Server (FSMDynamicManagedComponent)	●			1	🟨
OS (FSMChildService)	●			7	🟨

Figure 1.3: Example application map (or model).

The best APM solutions are very model-driven. That is, they work best when you can provide them with a *very* complete map of the application's components. That map, or model, directly contributes to the next dimension of the 5D approach.

Application Component Deep-Dive Monitoring

With your individual components fully mapped out, the next dimension concerns itself with digging deep into each one to ferret out component-level performance. APM solutions vary widely in how they do so, and in fact, a major competitive difference between APM solutions is the different types of components they work with. Some might only work with Java, for example, and might produce component-level performance displays like the one Figure 1.4 shows.


Geo Aware Applications > Geo Aware Applications (US) > **Java EE Requests**

Monday, September 14, 2009 10:24 AM - 2:24 PM 4 hours

Request				Calls	Response Time (s)				Investigate (%)		Diagnose	
Name	Health	System	Context	Total	Current	Avg	Max	Total	Incomplete	Exceptional Exits	Traces	Collect
GET /physician/search.do	◆	medrec1 [medrec1app1host]	/physician	926	0.005	0.007	0.375	6.661	0.00	0.00	0	
POST /physician/searchresults.do	◆	medrec1 [medrec1app1host]	/physician	804	0.083	0.125	2.110	100.261	0.00	0.00	0	
POST /physician/visit.do	⚠	medrec1 [medrec1app1host]	/physician	449	24.022	15.663	61.360	7,032.526	0.00	0.00	30	
POST /physician/visit.do	⚠	MedRec1Cluster (WebLogicCl	/physician	449	24.022	15.663	61.360	7,032.526	0.00	0.00	30	
POST /physician/visit.do	⚠	MedRec1Server1 (WebLogicS	/physician	449	24.022	15.663	61.360	7,032.526	0.00	0.00	30	
POST /physician/visit.do	✓	MedRec1Server1 (WebLogicS	/physician	0	n/a	n/a	n/a	n/a	0.00	0.00	0	
GET /physician/medicalrecord.do	⚠	medrec1 [medrec1app1host]	/physician	225	9.006	9.150	20.750	2,058.639	0.00	0.00	25	
GET /physician/searchresults.do	✓	medrec1 [medrec1app1host]	/physician	0	n/a	0.000	n/a	n/a	0.00	0.00	0	
GET /patient/record.do	✓	medrec1 [medrec1app1host]	/patient	0	n/a	0.000	n/a	n/a	0.00	0.00	0	
GET /patient/logout.do	✓	medrec1 [medrec1app1host]	/patient	327	0.000	0.007	0.140	2.439	0.00	0.00	0	
GET /patient/medicalrecord.do	✓	medrec1 [medrec1app1host]	/patient	653	0.289	0.256	1.625	167.280	0.00	0.00	0	
GET /patient	✓	medrec1 [medrec1app1host]	/patient	326	0.000	0.003	0.047	0.840	0.00	0.00	0	

POST /physician/visit.do on Server MedRec1Server1

Breakdowns are based on **416** samples out of **449** requests.

Figure 1.4: Component performance deep dive.

The idea is that you can see exactly where the application is spending most of its time, helping you focus on the areas that are consuming the most time and therefore offer the most opportunity for improvement.

A robust APM solution will be much more cross-platform, however. You'll typically want to look for:

- Robust support for different back-end database systems, with (of course) a focus on whatever database brands you're using in your environment
- Robust support for different development frameworks (Java, .NET Framework, and so on), with (again) a focus on whatever frameworks you're using in your applications
- Some ability to consider the physical and virtual infrastructure that your applications run on

Cross Reference

Chapter 4 will concern itself entirely with this dimension.

Application Performance Management Database

Finally, with all that raw data pouring in, you'll need a place to put it and analyze it. APM solutions typically use an *application performance management database* to correlate all the data. The real purpose of this database is to take the raw data from the first four dimensions and produce the visualizations you need to investigate the root cause of a performance problem.

The database is where a lot of APM's "magic" comes into play. A solution that can see specific EUE times, and automatically trace that through your component model to locate components that are contributing to the problem, is a useful tool indeed.

Cross Reference

Chapter 5 will discuss this aspect of APM.

How Vendors Implement 5D

Like many models, the 5D approach isn't always implemented in exactly the dimensions I've discussed here. Some vendors, for example, may implement the entire 5D approach but may break the capabilities out differently—perhaps offering three aspects rather than five. The important thing is to make sure you have *all five* capabilities regardless of how they're packaged. In fact, Chapter 5 will discuss that concept in some detail and provide examples of how you might find APM vendors packaging the five dimensions.

To whet you're appetite, here's one example of how a vendor might package the five dimensions:

- EUE monitoring implemented as a discrete functionality
- User transaction profiling and application component deep-dive implemented in a single combined set of functionality called "application management"
- Additional deep-dive functionality—perhaps specific to databases—implemented as a discrete set of functionality

Coming Up Next

In the next chapter, we'll look at tracking and monitoring user transactions. User transactions are one of the first tools you have to start finding application performance problems: When the EUE isn't looking good, profiling a user transaction can help you start narrowing the root cause.

Chapter 3 will be all about discovering and modeling application components. Although EUE should be your ultimate measure of an application's health, it doesn't do much for helping you find the underlying cause. To search out the root cause of a problem, you need to first know about every component that is contributing to the application—and that's where discovery and modeling come in.

Chapter 4 is where we'll learn how APM tools can dive deep inside those individual components to measure performance at a very detailed and granular level. This is one of your strongest troubleshooting tools for performance problems, and it leverages that infrastructure-centric, data center-focused view of your application—but it does so *after* you've identified a problem at the end user's side of things. In fact, a big part of Chapter 4 will be discussing the "flow up" and "flow down" models for utilizing the 5D approach.

Finally, in Chapter 5, we'll bring everything together and look at how to manage application performance data. The idea is to take all the data generated by the dimensions of the 5D approach and correlate that information so that specific end user problems can be more quickly related to an underlying root cause in one of the application components. We'll also look at the concept of *comprehensive* APM tools, meaning tools that support the *entire* 5D approach rather than just two or three dimensions.

In all, this book should serve as a useful guide to a new, more powerful, and more beneficial way of managing and monitoring application performance.