Realtime publishers

The Definitive Guide™ *To*

Application Performance Management

sponsored by

Compuware
We make IT rock around the world

Greg Shields

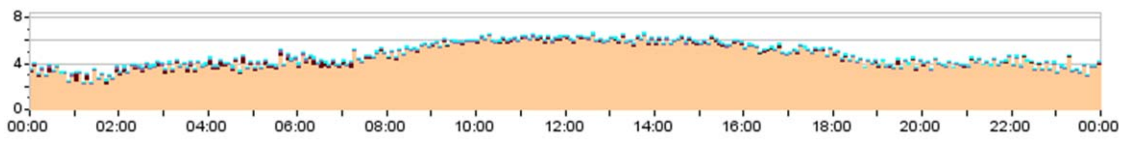
Copyright Statement

Chapter 5: Understanding the End User's Perspective

TicketsRUs.com IT Director John Brown is feeling a bit…overwhelmed…with charts today. Maybe it was the late night last night, or the early rise this morning. Or, maybe someone secretly switched the black lid with the red one on the coffee pot again, which, he thinks to himself, is not a very funny joke.

In any case, John finds himself staring blankly at a set of charts from his recently-implemented APM solution, finding little in their meaning this morning. He looks through charts of individual server performance. He flips through those relating to his networking behaviors and finds nothing there of interest either. He even peeks into the transaction breakdowns between his servers, ridiculously comprehensive in their level of captured data, but ultimately too low-level for his management-oriented mind to comprehend. Those charts are for a developer's brain, not his.

Leaning forward in his chair, he fixates on one in particular:



In looking at this graphic, he finds that he cares little about what that chart actually represents. "Some measurement of the ticketing system had this little bump during the workday hours," he thinks to himself, "that's interesting, I guess."

"These charts are giving us the information we want," he thinks to himself, "They help my admins find and fix problems. They help my network engineers track down bottlenecks. Heck, they even found the piece of code that caused that big problem a few months ago. We'd have never tracked that down without the transactional views. Yet something still nags me…

"…I want to know how my users are doing."

John's problem today has nothing to do with lack of monitoring. With his new APM solution in place, quite the opposite is true. Fully implemented, John's APM solution now gathers metrics from servers and their individual applications. The network itself is represented, both from the perspective of individual servers as well as the WAN as a whole. Yet in all of these metrics he's gathering, he's missing the one piece that ultimately represents success: His users' experience.

Just then the phone rings. It's Dan Bishop, the COO and John's direct superior. Things are never good when Dan calls, "I'm hearing scattered reports that our wait time on the system has spiked to over 20 minutes per purchase. What's up?"

"Nothing that I can see here, Dan," John reports.

"Well, your numbers might not show it," Dan continues, "but an old golfing buddy of mine just called in and reported the same. Track it down and let me know. Oh, and put two tickets for next week's concert at the arena on hold for him, will you?"

Why the End User's Perspective?

Chapter 3 of this guide walked you through the entire history of IT monitoring as we know it. Starting with the basics of "ping" responses, through SNMP polls, agent and agentless perspectives, and concluding with application analytics and transaction gathering, the history of monitoring has evolved dramatically over time. With each evolution, the areas in which monitoring integrates with your systems grow richer while their data grows more useful to the business. As continued in Chapter 4, each successive approach adds yet another layer to the overall view into a computing environment.

Yet Chapter 3 and 4's discussion concluded at the very point where experience-based monitoring actually starts to get interesting. With the development of End User Experience Monitoring (EUE), automated solutions for watching your business systems get their first looks into the actual behaviors experienced by an application's users. Gathering metrics from the perspective of the user themselves brings a level of objective analysis to what has traditionally been a subjective problem. If you've ever dealt with the dreaded "the servers are slow today" phone call, you understand this problem.

What Is Perspective?

This guide has used the term "perspective" over and over in relation to the types of data that can be provided by a particular monitoring integration. But what really is perspective, and what does it mean to the monitoring environment?

It is perhaps easiest to consider the idea of perspective as relating to the orientation of a monitors view, which determines the kinds of data that it can see and report on. Although the computing environment is the same no matter where a monitor is positioned, different monitors in different positions will "see" different parts of the environment.

Consider, for example, a set of fans watching a baseball game. If you and a friend are both watching the game but sitting in different parts of the stadium, you're sure to capture different things in your view. Your friend who is sitting in the good seats down by the batter is likely to pick up on more subtle non-verbal conversations between pitcher and catcher. In contrast, your seats deep in the outfield are far more likely to see the big picture of the game—the positioning of outfielders, the impact of wind speed on the ball, the emotion and effects of the crowd on the players—than is possible through your friend's close-in view.

Relating this back to applications and performance, it is for this reason that multiple perspectives are necessary. Their combination assists the business with truly understanding application behaviors across the entire environment. An agent that is installed to an individual server will report in great detail about that server's gross processing utilization. That same agent, however, is fully incapable of measuring the level of communication between two completely separate servers elsewhere in the system.

Why the End User?

Thus far, this guide has discussed how the vast count of different monitors enables metrics from a vast number of perspectives: Server-focused counters are gathered by agents, network statistics are gathered through probes and device integrations such as Cisco NetFlow, transactions and application-focused metrics are gathered through application analytics; the list goes on. Yet, it should be obvious that this guide's conversation on monitoring remains incomplete without a look at what the end users see in their interactions with the system.

This view is critically necessary because it is not possible—or, at the very least, exceptionally difficult—to construct this experience using the data from other metrics. Relating this back to the baseball example, no matter how much data you gather from your seat in the outfield, it remains very unlikely that you'll extrapolate from it what the pitcher is likely to throw next.

For the needs of the business application, end user experience (EUE) enables administrators, developers, and even management to understand how an application's users are faring. First and foremost, this data is critical for discovering how successful that application is in servicing its customers. Applications whose users experience excessive delays, drop off before accomplishing tasks, and don't fulfill the use case model aren't meeting their users' needs. And those that don't meet user needs will ultimately result in a failure to the business.

The Use Cases for EUE

Though failure is a strong word, the reality is that EUE is important most especially for those applications that service customers outside the business. As with the ongoing story of TicketsRus.com, when an outward-facing application stops performing to expectations, customers go elsewhere, with results that are often disastrous to the business.

This line of thinking introduces a number of potential use cases where EUE monitoring can benefit an application's quality of service. EUE monitoring works for valuating the experience of the absolute end user as well as in other ways:

- Quantifying the performance characteristics of connected users as well as differences in performance between users in different geographic locales
- Simulating user behaviors through the use of robots for the purpose of predicting service quality degradations
- Identifying where internal users, as opposed to the absolute end user, are seeing a loss of service
- Keeping external service providers honest through independent measurements of their services

Figure 5.1 shows a reproduction of Chapter 4's example e-commerce system. In this version of the image, however, four areas where EUE monitoring can potentially be integrated are highlighted. Those four areas correspond to the four bullets previously mentioned, and are explained in greater detail in the next sections.

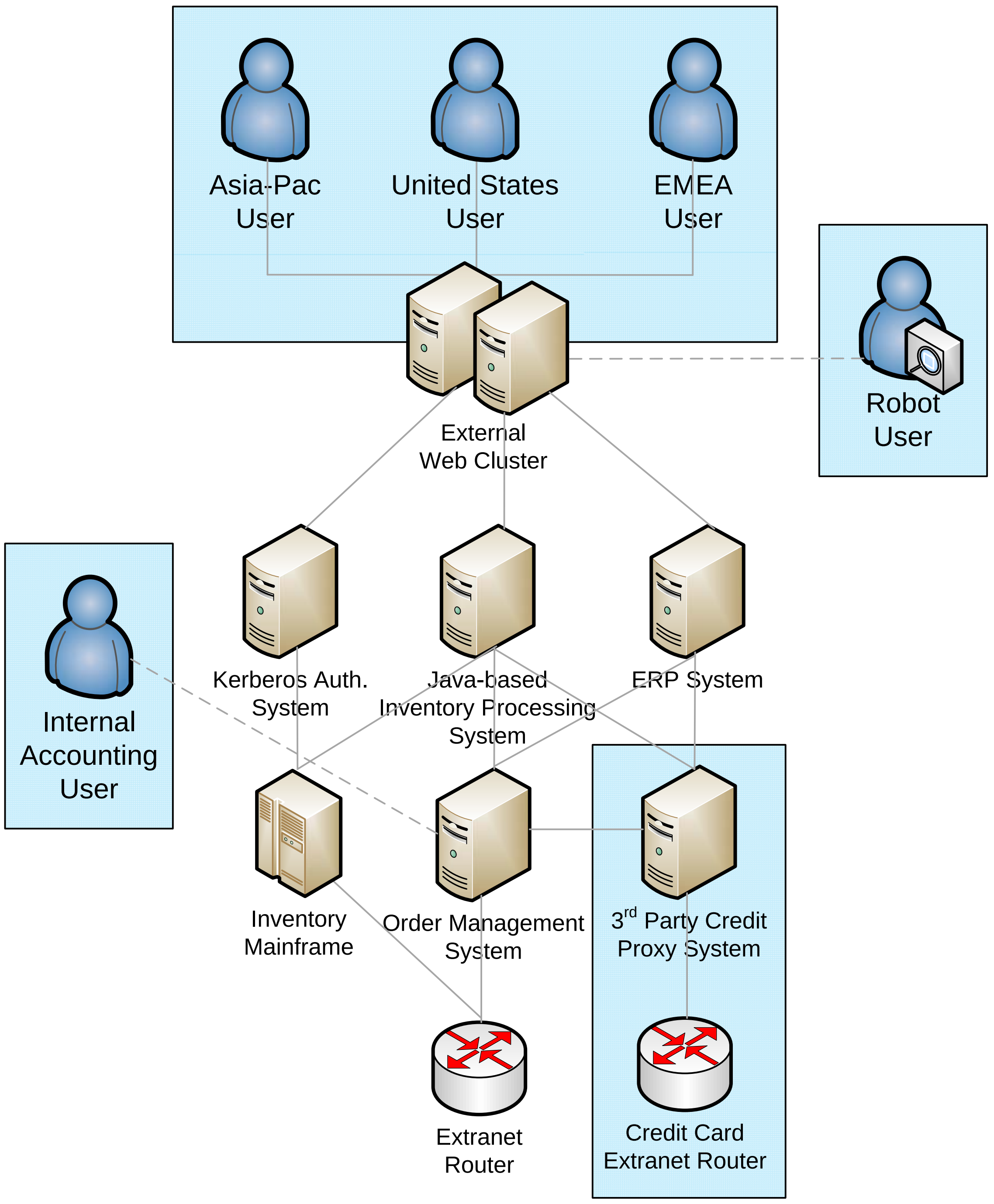


Figure 5.1: Multiple use cases exist for targeting EUE monitoring.

Customer and Multi-Site Perspective

The first and most obvious target for EUE monitoring relates to the actual end users themselves. As you can see in Figure 5.2, this integration occurs with users and across the various geographic locations where users may exist. Here, the behaviors of users are captured by a suite of special monitors that watch for and report on the behaviors of those users.

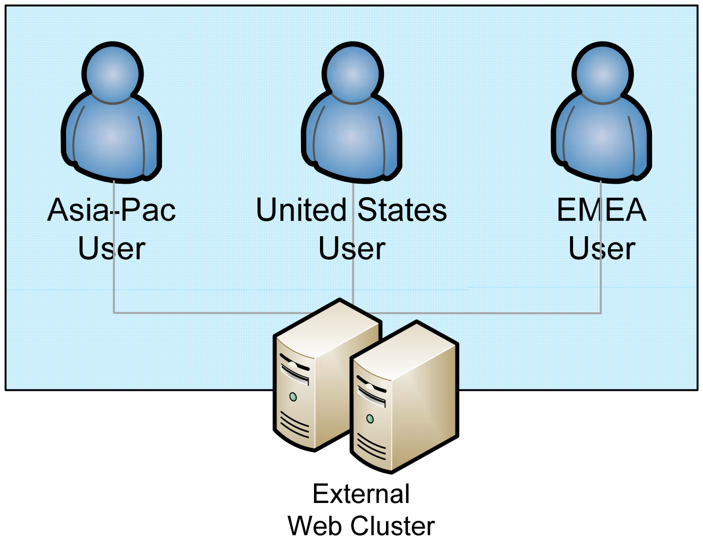


Figure 5.2: EUE can measure user behaviors across multiple connections in multiple geographic locations.

How does this watching and reporting occur? In short, by creating a log of each user's activities. Consider for a moment how an Internet-facing application works. In the example, the application's user interface (UI) is Web-based, served through a front-end Web cluster. For a user to work with that Web-based application, the cluster must generate and present Web pages to the user. The user interacts with those Web pages by clicking in specified locations, with each click resulting in some response returned back to the user.

A benefit of working with Web-based applications is that each click can be encapsulated into its own transaction. When the user clicks on a Web page link, that click begins a long chain of events. The Web server interacts with down-level services to gather necessary data. Those down-level servers may then work with others even further down the application's stack. Eventually, through some combination of effort, the right data is gathered. That data is then passed back to the front-end Web servers, which render new content for the user.

By measuring which links the user clicks on, as well as the response time in receiving and rendering resulting data back to the user, it is possible to identify the quantity of time consumed by each step in the process. Later, this chapter will talk more about the spread of time between the different system elements—client, network, and server—but for now, recognize that EUE monitoring for end users works because the action of each user is encapsulated into a Web transaction that can be measured.

The Impacts of Geography

Although it is commonly accepted that the Internet is equally accessible by every connection, the quality of each connection is in actuality quite different. For example, a user in the United States may find that their experience with an Internet-facing application is acceptable. This may occur due to the high quality of Internet connections in the US as well as the geographic locality between user and application. When an application and user are in a well-connected location of Internet service, such as the same country, their connection tends to be of a higher quality.

Conversely, users that connect to a US-based application from the Asia-Pacific or EMEA regions must route their communication through transcontinental connections and over a much longer distance. The quality of those connections as well as their length of travel can impact the overall experience of the user. By measuring an application's performance from a series of different geographical locations, it is possible to recognize when affecting networking conditions exist.

As with the earlier example, because measurements here are made at the Web server, all inbound connections can be measured against each other. Determining the time required for a full user action to be completed illuminates much about the quality of the connection, and thus the user's experience.

Internal & External Robot Perspective

Yet even with this tracking at the Web server, not all users behave in the same way, and no user behaves with complete predictability. A user's interaction with an Internet-facing application tends to change throughout their use: They walk away from their computers or work on something else for a period of time. They take longer to read through one Web page as opposed to another. They cease their interaction with the application altogether without walking through a full use case.

With Internet-based applications, these non-standard behaviors are more the norm than the exception. Users are used to the "always-on" nature of the Internet, electing to work with its applications as if they too were always on—logging in, logging out, stepping away mid-transaction, moving on to another task, and so on. Because of these erratic behaviors another more predictable "end user" perspective is necessary. That perspective is provided through internally- and externally-placed robots (see Figure 5.3).

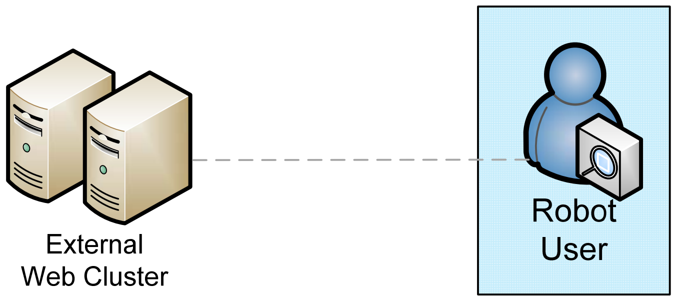


Figure 5.3: Robot users can repeat the same action to derive a baseline of performance and watch for deviations.

The primary job of a robot is to simulate the behavior of a standard user. Such a robot is programmed with a series of actions, the completion of which is accomplished in a known period of time. Repeatedly running through that set of actions creates a baseline response profile for the application. The actions are known and are run over and over, so administrators can then be alerted when performance deviates from the baseline.

For example, if a robot is preconfigured to click through a series of pages on the External Web Cluster with the goal of finding and eventually purchasing an item, it is possible to determine the average period of time required to complete the actions. When that period of time deviates from the baseline at some point later on, it can be assumed that an issue or problem is occurring somewhere in the application.

Although robots alone are not likely to assist in locating the problem, they can operate as a bellwether for downstream problems. Identifying a change in the overall performance back to the user often means that a problem or other issue should be reviewed using other monitoring metrics.

Internal User Perspective

The ultimate end user, however, isn’t the only group of individuals that interact with an application. An entirely different set of users, usually internal to the business, have the job of maintaining that application. These users tend not to be involved in the IT management of the application. Instead, they are associated with managing the workflow related to the products or services being sold by the business organization.

Using an example from TicketsRus.com, the primary mechanism for their ticket sales is through their Internet-based application. Many individuals in the IT organization—administrators, engineers, developers, and so on—are responsible for maintaining the technology that powers that application. Yet a completely different force of individuals is also necessary for ensuring that the right tickets are brokered for the right events and to the right people. These accounting, sales, and management individuals require their own interface into the application that has nothing to do with its ultimate rendering of Web pages for customers.

Figure 5.4 shows how EUE can assist these individuals. Here, an Internal Accounting User interacts with the Order Management System to ensure that the right tickets are always available for purchase. The actor in this figure may be one person or an entire department of individuals that are spread across a country or the globe. Targeting EUE monitoring in this location gives the troubleshooting administrator another set of data to identify user-visible behaviors on the system.

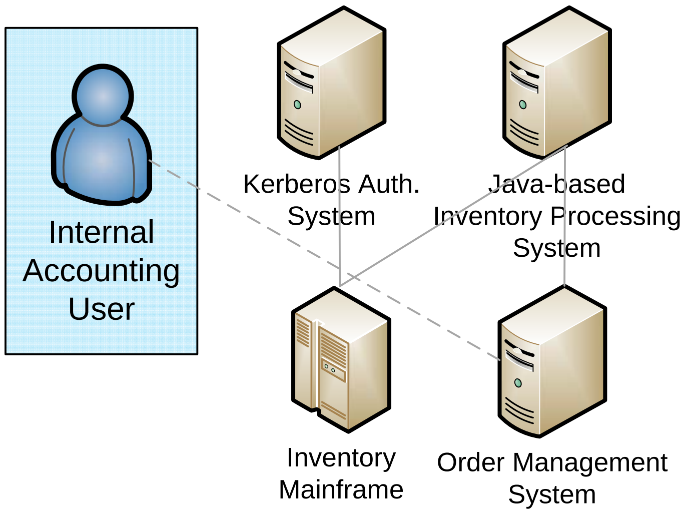


Figure 5.4: EUE can be incorporated into other parts of the application to monitor the behaviors of internal users as well as external.

EUE monitoring at the level of the Order Management System can identify when that down-level system experiences a loss of performance. It is possible to compare the information gathered at this level with other information at the External Web Cluster to isolate potential problems. Such EUE monitoring for internal users can occur at any level in the application's stack where performance is a concern. Each addition of monitoring at different user endpoints provides yet another set of performance measurements that are useful in measuring overall quality of service.

Service Provider Perspective

Today's business applications are rarely atomic in their architecture. One business' application often needs to communicate with others for data, processing, or orchestration of customer orders and requests. Further, all Internet-facing applications depend on their Internet connection for service, and that service must come from somewhere. In all these situations, an external party is responsible for providing a service for the application.

Whenever an outside party is contracted for those services, a binding agreement is usually laid into place to define their quality. An Internet or Cloud Computing service provider guarantees certain levels for bandwidth and latency as part of the price paid for their services. A credit card processing facility guarantees a proscribed level of uptime. Suppliers with direct application connections must meet minimum requirements.

Yet in many cases, the organization doing the monitoring of that service level is the provider itself. For example, an Internet provider guarantees a particular service level but does so based on the metrics that they themselves measure. You can imagine the conflict of interest that occurs in this case when the business providing a service is also in the business of measuring their success with that service.

In this case, another form of EUE integration becomes useful. Targeting EUE here provides a secondary set of measurements when it becomes necessary to independently assess external-party service levels. Figure 5.5 shows another example of an external service that could be monitored by such an EUE integration. There, the third-party Credit Card Proxy along with its Extranet Router is contracted to handle payment card services for the e-commerce system. This is a common service that is contracted out to external parties because of the complexities of payment card handling.

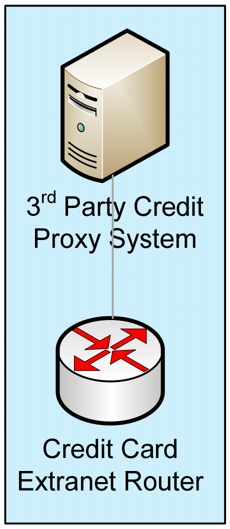


Figure 5.5: Validating service provider connections is another effective use of EUE monitoring.

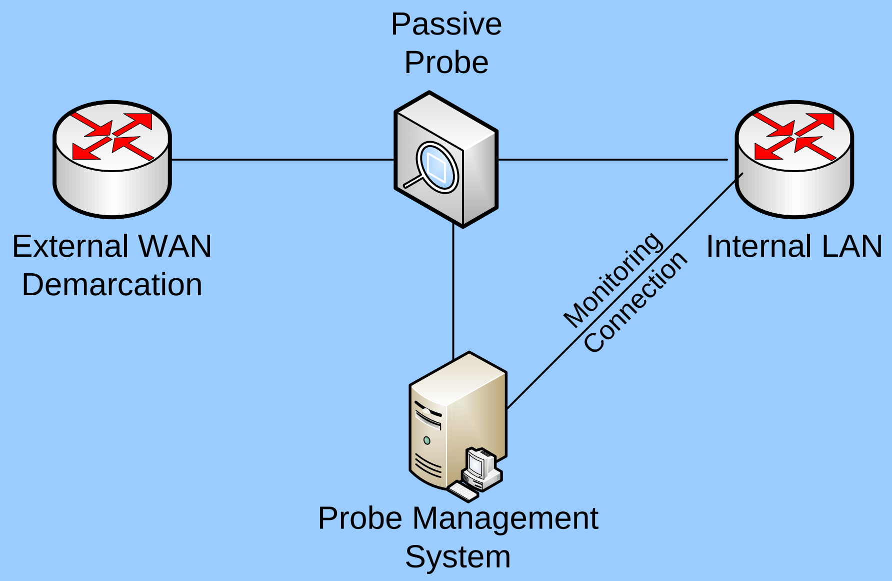
In this situation, payments for any services or items on the e-commerce Web site route through that external system. Yet this external system often lies outside the direct control of the local IT organization. Since the business derives all of its income through this interface, it is considered mission critical to the business. As such, it is a best practice to implement independent monitoring to verify its service quality.

This kind of monitoring can and likely should occur in-line with any external system that participates in the application. The resulting metrics can be used in independently identifying any violations to Service Level Agreements (SLAs) as well as in negotiating chargebacks when vendors don't meet their agreed obligations.

The Use of Probes in Service Provider Monitoring

The use of service providers for portions of an application's infrastructure is commonplace in business today. Not common are direct monitoring integrations into that service provider's network or server equipment. If, for example, you want to measure the bandwidth and latency across your Internet connection, your ISP is not likely to give you their internal passwords to gather data directly from their hardware.

In cases like this, the use of in-line probes is useful in gathering the right information. Probes were first introduced back in Chapter 4 as one mechanism for integrating APM monitoring into otherwise-unavailable infrastructure components. There, Figure 4.9 (provided below) showed an example of how a probe can be installed between a supplier extranet and an internal LAN to monitor traffic.



If you leverage external service providers for elements of your application's functionality and are unable to gather statistics directly from their equipment, consider the use of probes with your APM solution to gather this data.

The Role of Transactions in EUE

It should be obvious at this point that there are a number of areas where EUE provides benefit to the business and its applications. Yet this chapter hasn't yet discussed how EUE goes about gathering its data. If end users are scattered around the region or the planet, how can an EUE monitoring solution actually come to understand their behaviors? Simply put, *the metrics are right at the front door*.

EUE Feeds BSM

A much larger conversation on the role of end-user performance in meeting (or breeching) business goals will be discussed in Chapter 9. There, you will find a discussion on how APM links to the ideals of Business Service Management (BSM). Even more information about BSM's focus on applications can be found in the book *The Definitive Guide to Business Service Management*, downloadable from http://www.realtimepublishers.com.

Think for a moment about a typical Internet-based application such as the one being discussed in this chapter. Multiple systems combine to enable the various functions of that application. Yet there is one set of servers that interfaces directly with the users themselves: the External Web Cluster. Every interaction between the end user and the application must proxy in some way through that Web-based system. This centralization means that every interaction with users can also be measured from that single location.

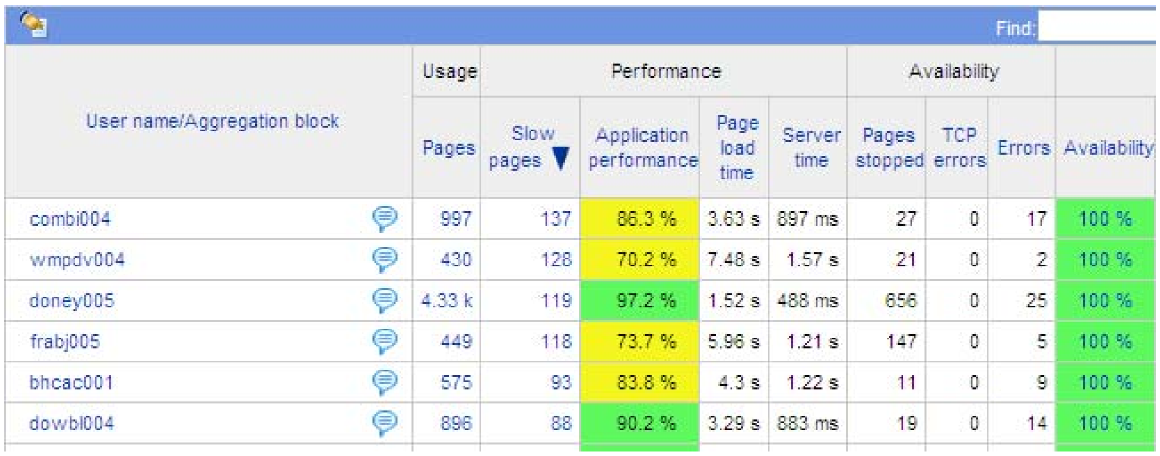
EUE leverages transaction monitoring between users and Web servers as a primary mechanism for defining the users' experience. Every time a user clicks on a Web page, the time required to complete that transaction can be measured. The more clicks, the more timing measurements. As users click through pages, an overall sense of that user's experience can be gathered by the system and compared with known baselines. These timing measurements create a quantitative representation of the user's overall experience with the Web page, and can be used to validate the quality of service provided by the application as a whole.

It is perhaps easiest to explain this through the use of an example. Consider the typical series of steps that a user might undergo to browse an e-commerce Web site, identify an item of interest, add that item to their basket, and then complete the transaction through a check out and purchase. Each of these tasks can be quantified into a series of actions. Each action starts with the Web server, but each action also requires the participation of other services in the stack for its completion:

- **Browse an e-commerce Web site.** The External Web Cluster requests potential items from the Java-based Inventory Processing System, which gathers those items from the Inventory Mainframe. Resulting items are presented back to the External Web Cluster, where they are rendered via a Web page or other interface.
- **Identify an item of interest.** This step requires the user to look through a series of items, potentially clicking through them for more information. Here, the same thread of communication between External Web Cluster, Inventory Processing System, and Inventory Mainframe are leveraged during each click. Further assistance from the ERP system can be used in identifying additional or alternative items of interest to the user based on the user's shopping habits.
- **Add that item to the basket.** Creating a basket often requires an active account by the user, handled by the ERP system with its security handled by the Kerberos Authentication System. The actual process of moving a desired item to a basket can also require temporarily adjusting its status on the Inventory Mainframe to ensure that item remains available for the user while the user continues shopping. Information about the successful addition of the item must be rendered back to the user by the External Web Cluster.
- **Complete the transaction through a check out and purchase.** This final phase leverages each of the aforementioned systems but adds the support of the Credit Card Proxy System and Order Management System.

In all these conversations, the External Web Cluster remains the central locus for transferring information back to the user. Every action is initiated through some click by the user, and every transaction completes once the resulting information is rendered for the user in the user's browser. Thus, a monitor at the level of the External Web Cluster can gather experiential data about user interactions *as they occur*. Further, as the monitor sits in parallel with the user, any delay in receiving information from down-level systems is recognized and logged.

A resulting visualization of this data might look similar to Figure 5.6. In this figure, a top-level EUE monitor identifies the users who are currently connected into the system. Information about the click patterns of each user is also represented at a high level by showing the number of pages rendered, the number of slow pages, the time associated with each page load, and the numbers of errors seen in producing those pages for the user.



User name/Aggregation block	Usage	Performance				Availability			
	Pages	Slow pages ▼	Application performance	Page load time	Server time	Pages stopped	TCP errors	Errors	Availability
combi004	997	137	86.3 %	3.63 s	897 ms	27	0	17	100 %
wmpdv004	430	128	70.2 %	7.48 s	1.57 s	21	0	2	100 %
doney005	4.33 k	119	97.2 %	1.52 s	488 ms	656	0	25	100 %
frabj005	449	118	73.7 %	5.96 s	1.21 s	147	0	5	100 %
bhcac001	575	93	83.8 %	4.3 s	1.22 s	11	0	9	100 %
dowbl004	896	88	90.2 %	3.29 s	883 ms	19	0	14	100 %

Figure 5.6: User statistics help to identify when an entire application fails to meet established thresholds for user performance.

Adding in a bit of preprogrammed threshold math into the equation, each user is then given a metric associated with their overall application experience. In Figure 5.6, you can see how some users are experiencing a yellow condition. This means that their effective performance is below the threshold for quality service. Although this information exists at a very high level, and as such doesn't identify *why* performance is lower than expectations, it does alert administrators that degraded service is being experienced by some users.

An effective APM solution should enable administrators to drill down through high level information like what is seen in Figure 5.6 towards more detailed statistics. Those statistics may illuminate more information about why certain users are experiencing delays while others are not. Perhaps one server in a cluster of servers further down in the application's stack is experiencing a problem. Maybe the items being requested by some users are not being located quickly enough by inventory systems. Troubleshooting administrators can drill through EUE information to server and network statistics, network analytics, or even individual transaction measurements to find the root cause of the problem.

Browsers Aren't the Only Client Application

This chapter has talked at length about how Internet-facing e-commerce systems are a good target for EUE monitoring. These types of applications tend to use a standard Internet browser as their client interface. However, EUE needn't necessarily be limited to browser-based client interactions. EUE integrations can be incorporated into remote application infrastructures such as Windows Terminal Services or Citrix XenApp, or other client-based solutions in much the same way. Such solutions leverage different mechanisms for gathering data, but the result is the same: The user's experience is measured and quantified.

The C-N-S Spread

Transactions provide the basis by which experience is measured in applications; however, the process of analyzing the transactions themselves can be a challenging activity. The prototypical systems administrator often doesn't have the development background or the experience with an application's codebase to convert individual transaction information into something that is actionable. Further, some problems don't necessarily exist at the transaction level. If a loss of application performance has more to do with a server failure, analyzing individual transactions is too close a perspective to be useful.

It is for these reasons that an effective APM solution will create numerous visualizations out of collections of transaction information. These visualizations roll up the communication behaviors between user and server or server and server into an easy-to-use graphical form. One particularly useful visualization that is commonly used in troubleshooting is the C-N-S Spread (see Figure 5.7).

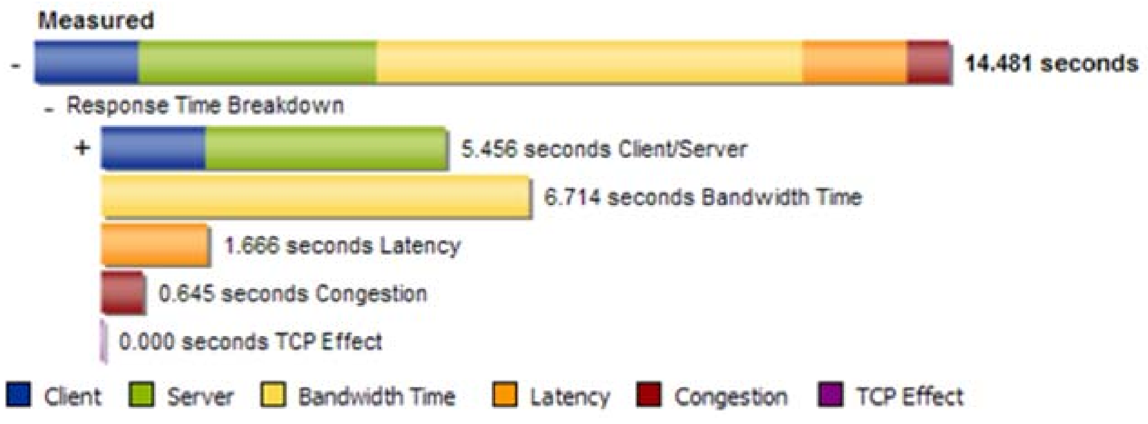


Figure 5.7: The C-N-S Spread.

The C-N-S Spread measures the amount of time required to complete a transaction between two elements. In generalities, it breaks down that quantity of time between the amount consumed by their Client, Network, and Server components. You can see in Figure 5.7 that these three components are broken down even further to include the network overhead components of Latency, Congestion, and the TCP Effect. This spread of information illuminates a number of interesting behaviors associated with the communication:

- **Client.** The quantity of time spent at the client. This can include the amount of time required for the client to process and render incoming data for the user.
- **Server.** This relates to the amount of time a server is processing an inbound request. This can include locating records in a database, processing the business logic surrounding those records, or completing essentially any activity associated with the request.
- **Bandwidth time.** This represents the Network link speed component of the Spread. Here, the amount of time required to clock data onto the network is measured; the faster the link speed, the faster the clocking rate.
- **Latency.** This represents the Network distance component of the Spread—the amount of time required for requests and replies to traverse the network. The greater the distance, the higher the latency.
- **Congestion.** Similar to Latency, congestion measures the delay associated with too much data attempting to pass across the network. When congestion is high, data is delayed or even discarded if the network is oversubscribed.
- **TCP Effect.** Any network communication also has a certain level of flow-control overhead associated with reliably getting packets from one location to another. This TCP Effect can be broken out separately as well to identify when TCP-based errors or other issues are having an impact on communication.

The C-N-S Spread Illuminates Environment Behaviors

Graphs such as the C-N-S Spread bring high-level detail to what would otherwise be individual packets of information crossing the network. They are particularly interesting because the creation of such visualizations is usually not possible with traditional monitoring solutions alone. At the same time, their creation enables a look at application processing that is more holistic than with traditional monitoring point solutions.

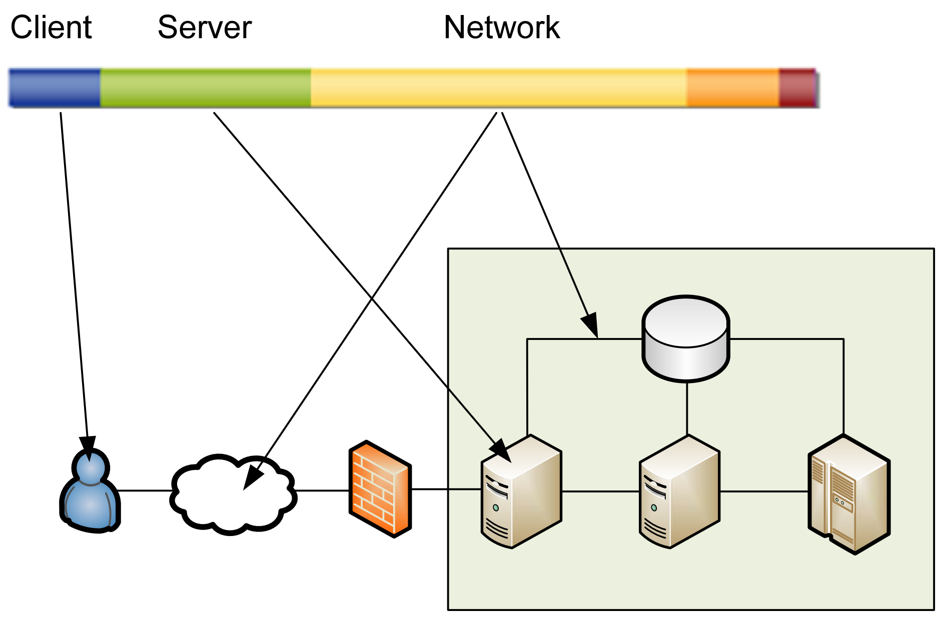


Figure 5.8: Graphs like the C-N-S Spread leverage integrations across the suite of an application's components.

Consider the areas in which monitoring must be in place to create such a visualization. Clients must be monitored to understand their behaviors. The network must be monitored to watch for transaction traversal. The processing of requests on servers themselves must additionally be watched. Even more complex is the logic involved with tracking this information across the various components of an environment, and ultimately converting it to a useable form. Only a mature APM solution with its integrations across the suite of application components has the reach necessary to create such a visualization.

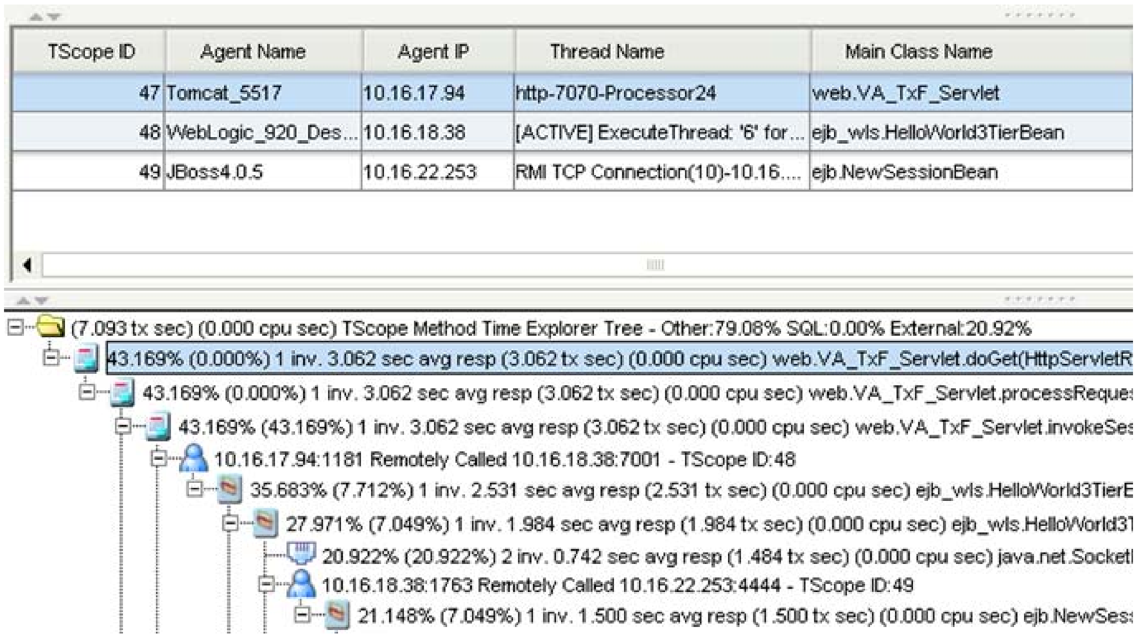
A Use Case of the C-N-S Spread as Troubleshooting Tool

Once these components are in place, the resulting information provides a starting point for tracking down problems with an application. For example, assume that a problem has occurred in our chapter's application. That problem lies deep within the application's processing, making it difficult to "see" with the naked eye or through any one component's monitoring integrations. Perhaps this problem has to do with a recently–updated piece of code in the Inventory Processing System. This update changed a series of methods within the system's home-grown Java codebase.

Coded into one updated method was a change that removed optimizations in inventory processing. Removing this optimization forced the server to slow its processing of inventory requests. Such a problem can be commonplace, especially with home-grown code, and can be quite difficult to track down once implemented.

In this case, however, the application's administrators were quickly able to determine that a problem existed with the updated code. Looking at a visualization similar to Figure 5.6, the application's administrators immediately noticed that the metrics associated with user experience were dropping from the green state to the yellow, and occasionally the red state. This high-level monitor immediately indicated that users were "experiencing" the problem. Although it is likely that no one had called in to complain—perhaps users considered it a momentary hiccup rather than an endemic problem—administrators were immediately aware that something was wrong.

With this information in hand, administrators could quickly pull up another visualization similar to Figure 5.9 to trace the problem at a slightly lower-level perspective. There, they identified that the time consumed by the Server component was far larger than its established baseline.



TScope ID	Agent Name	Agent IP	Thread Name	Main Class Name
47	Tomcat_5517	10.16.17.94	http-7070-Processor24	web.VA_TxF_Servlet
48	WebLogic_920_Des...	10.16.18.38	[ACTIVE] ExecuteThread: '6' for...	ejb_wls.HelloWorld3TierBean
49	JBoss4.0.5	10.16.22.253	RMI TCP Connection(10)-10.16....	ejb.NewSessionBean

```
(7.093 tx sec) (0.000 cpu sec) TScope Method Time Explorer Tree - Other:79.08% SQL:0.00% External:20.92%
  43.169% (0.000%) 1 inv. 3.062 sec avg resp (3.062 tx sec) (0.000 cpu sec) web.VA_TxF_Servlet.doGet(HttpServletR
    43.169% (0.000%) 1 inv. 3.062 sec avg resp (3.062 tx sec) (0.000 cpu sec) web.VA_TxF_Servlet.processReque
      43.169% (43.169%) 1 inv. 3.062 sec avg resp (3.062 tx sec) (0.000 cpu sec) web.VA_TxF_Servlet.invokeSes
        10.16.17.94:1181 Remotely Called 10.16.18.38:7001 - TScope ID:48
          35.683% (7.712%) 1 inv. 2.531 sec avg resp (2.531 tx sec) (0.000 cpu sec) ejb_wls.HelloWorld3TierE
            27.971% (7.049%) 1 inv. 1.984 sec avg resp (1.984 tx sec) (0.000 cpu sec) ejb_wls.HelloWorld3T
              20.922% (20.922%) 2 inv. 0.742 sec avg resp (1.484 tx sec) (0.000 cpu sec) java.net.Socketl
              10.16.18.38:1763 Remotely Called 10.16.22.253:4444 - TScope ID:49
                21.148% (7.049%) 1 inv. 1.500 sec avg resp (1.500 tx sec) (0.000 cpu sec) ejb.NewSess
```

Figure 5.9: Drilling into the server's Java codebase enables developers to locate un-optimized code.

Clicking further into the details, administrators began peering into the individual servers to find areas of delay, eventually focusing their attention to the update on the Inventory Processing System. Recognizing that the problem is likely related to the update, administrators enlisted the support of the development team. That team dug deeper to find the visualization shown in Figure 5.9. There, you can see how a particular Thread and Main Class is highlighted along with its rate of delay. This timing information related to specific threads—and ultimately the methods being processed by those threads—enabled the development team to quickly find and fix the offending code.

The rate at which a problem like this can be resolved is attributable directly to the depth of monitoring provided by an APM solution. Without its “everything and everywhere” approach to watching for environment behaviors, such a rapid resolution would not be possible.

The Impact of Users Themselves

Yet another area where an application’s performance can be impacted relates to the sheer number of users on the system. Even in the best of architectures, there occasionally comes the time when a product announcement or a large-scale run on services turns customers to your services all at once.

This can be a particular problem when services are available for a short period of time or on a limited basis. The TicketsRus.com story provides another metaphor for this situation in relation to its selling of concert and sporting event tickets. These types of items are generally available starting at a particular date and time, with a finite number of tickets available. While for many events this is not a problem as the level of ticket supply meets the level of customer demand, there occasionally comes the time when demand far exceeds supply: sporting event finals, major concerts, and so on.

The way in which these situations manifest into customer-facing systems is through a widespread slowdown in application performance. Figure 5.10 shows an example set of graphs that can explain such a situation. Here, an application’s Application Performance Index (Apdex) is related to the rate of unique users attempting to use the application. You can see here that the performance index falls dramatically with the inbound spike of users.

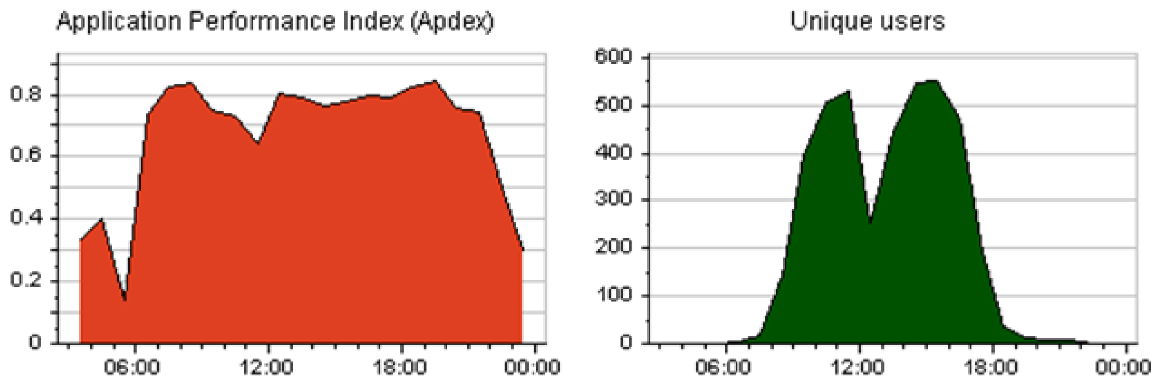


Figure 5.10: Relating an applications performance index to the number of unique users.

In the case of a massive user influx, metrics such as these help identify when performance problems have little or nothing to do with the application's architecture itself. Consider the types and rate of alerts that could be triggered by such an inrush of users. Processor utilization on servers goes above thresholds. Network bandwidth becomes saturated, causing latency to spike dramatically. Application analytics on down-level services begin notifying that they cannot keep up with the load. Web pages are unable to refresh, causing errors at the client level.

Lacking a holistic perspective on the environment, such a situation could cause an alert storm to the pagers of unsuspecting administrators. Administrators might find themselves struggling to bring meaning to such a situation, tracking down symptoms of a much greater problem.

Applications that have the right kinds of APM monitoring in place might be able to encapsulate overall application performance into an index such as the Apdex metrics noted in Figure 5.10. Relating this to the rapid rise in incoming users provides a focus for understanding why the alert storm is occurring. It also provides ways to recognize where system bottlenecks can be later eliminated to reduce the effect of massive popularity the next time demand overwhelms supply.

Lastly, is the ability to notify the end users themselves when their activities cause a reduction in overall performance. Although such performance problems aren't often the result of bad decisions made by the business, the business itself is usually the one that is blamed. One very useful way to eliminate finger-pointing and remind users of the site's overwhelming short-term popularity is to notify the users of the problem. In this case, users can be automatically alerted by the system that a high volume of requests is currently being received and that their requests may take longer than expected. In the end, an educated customer population is less likely to blame your business when the problem is related to their use.

Leveraging EUE for Improved Application Quality

This chapter has attempted to show how an EUE monitoring solution adds great value to the management of a large-scale application. EUE is, in fact, one of the pillars of an effective APM platform. It integrates monitoring across numerous components to bring a quantitative perspective to the otherwise-subjective impression of user behaviors. Because user behaviors can be quantified into specific actions that require specific amounts of time to complete, administrators can very discretely understand how successfully their application meets user needs.

EUE further improves this recognition of success by enabling a greater vision into the environment. That vision speeds root cause analysis, enabling visualizations that very quickly drill down to problems. Because user behaviors are specifically tracked, even the most challenging of code-oriented problems can be isolated for a quick fix.

Finally, EUE improves a business' capability to refine their application infrastructure over time. Its data shows where bottlenecks require hardware expansion or software update while providing a real-world justification for basing short-term and long-term planning activities.

To this point, however, EUE is still but one piece of the larger model of service created by an APM solution. That service model is the central whiteboard by which an application's components are laid upon and connected. Through the process of creating and refining an application's service model, the linkages between components become well-defined. This creates a web of dependencies upon which alerting and status information can be based. Chapter 6 discusses how this model is created, and how the concepts surrounding the service-centric monitoring approach enable a complete representation of an application's entire set of resources.