

The Power of the Business Transaction

The New Way to Manage Application Performance

Business White Paper

March, 2011

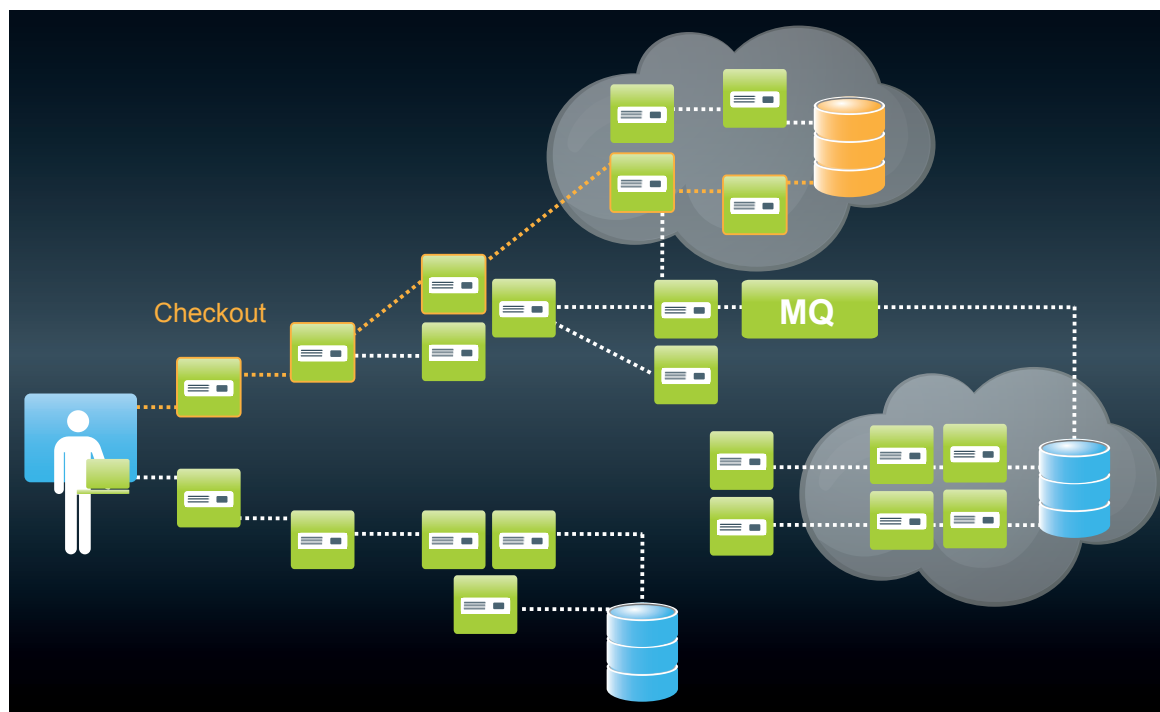
Introduction

For many companies, the application is the business. That means that uptime and availability are not just “nice to have’s”; they’re key to business survival.

But ensuring uptime and availability has never been harder than it is today. Application owners face a tremendous rate of change in regards to their production environments. Agile release cycles mean that more code drops are happening than ever before – and that’s just within the physical data center. When apps begin moving to virtual and cloud environments, supporting application health and performance becomes even more challenging.

What’s necessary is for app owners to confront and master the rate of change in their environments, ensuring that they can keep up with the needs of the business. They need to see and understand their entire application environment and resolve problems quickly when they occur – and do all of this without fear of what the next code drop may bring.

How can this be done? By *embracing business transactions*.



By becoming business transaction-centric – monitoring transactions and using them as the focus for an ongoing performance strategy – application owners can ensure uptime and availability even within a challenging application environment. Business transactions give them the insight that’s required in order to react to quickly changing conditions and respond accordingly.

THE WORLD PRIOR TO BUSINESS TRANSACTIONS

Two legacy approaches exist for managing application performance. The first pertains to managing the health of application infrastructure – figuring out CPU Utilization, Disk I/O, etc. But this offers a severely limited approach to application management. Evaluating infrastructure won’t tell you what’s actually happening inside the application itself, and this approach ends up being too high level without enabling application owners to quickly get to root cause.

The second approach pertains to collecting metrics related to method-level performance. Doing so can tell you what’s happening at a code level – as long as you are able to survive the flood of data and actually find the “smoking gun” hidden inside all that information. Collecting metrics provides no context as to the nature of the performance problem, and it is extraordinarily time consuming.

It’s easy to see how these two legacy approaches fall short. Evaluating infrastructure health fails to give you insight into the heart of the application; collecting method-level data is inefficient and frustrating. Neither approach works if your application has millions of lines of code and operates inside a dynamic, agile environment.

Application owners require a new approach. They need to monitor their applications through the lens of business transactions.

WHAT THE HECK IS A BUSINESS TRANSACTION, ANYWAY?

Consider a business transaction to be a *user-generated action* within your system. The best practice for determining the performance of your application isn’t to measure CPU usage, but to track the flow of a transaction that your customer, the end user, has requested.

A business transaction can take different forms based on your particular industry.

Here are some examples:	
eCommerce Bookstore	“Add to Cart” transaction
Insurance Portal	“File a Claim” transaction
Financial Services	“Pay My Bill” transaction
B2B IT Company	“Run a Credit Check” Transaction

One way to think about the business transaction is to consider that it ties business operations and the application together, representing how user requests are being serviced.

And why should you care? Because shifting your focus to business transactions completely changes the game in terms of your ability to support application performance.

HOW BUSINESS TRANSACTIONS SUPPORT APPLICATION MANAGEMENT

Business Transactions equip application owners with three important advantages.

Knowledge of User Experience

If a business transaction is a “user-generated action,” then it’s pretty clear how monitoring business transactions can have a tremendous effect on your ability to understand the experience of your end user.

If your end user adds a book to a shopping cart, is the transaction performing as expected or is it taking 3 seconds longer? (And what kind of impact will that have on end users? Will they decide to surf away and buy books somewhere else, thus depriving your business of not just the immediate purchase but the potential loss of lifetime customer revenue?)

Monitoring business transactions gives you a powerful insight into the experience of your end user.

Service Assurance – the ability to track baseline performance metrics

AppDynamics hears from our clients all the time that it’s difficult to know what “normal” actually is. This is particularly true in an ever-changing application environment. If you try to determine normal performance by correlating code-level metrics – while at the same time reacting to frequent code drops – you will never get there.

Business transactions offer a Service Assurance constant that you can use for ongoing monitoring. The size of your environment may change and the number of nodes may come and go, but by focusing on business transactions as your ongoing metric, you can begin to create baseline performance for your application. Understanding this baseline performance is exactly what you need in order to understand whether your application is running as expected and desired, or whether it's completely gone off the rails.

For example, you may have a sense of how your application is supposed to perform. But do you really know how it performs every Sunday at 6 p.m.? Or the last week of December? And if you don't, how will you know when the application is deviating from acceptable performance? It's figuring out "normal" in terms of days, weeks, and even seasons that you need to truly understand your application's baseline performance.

Triage & Diagnosis - always knowing where to look to solve problems

Finally, when problems occur, business transactions prevent you from hunting through logs and swimming through lines of code. The transaction's poor performance immediately shines a spotlight on the problem – and your ability to get to root cause quickly is dramatically improved.

If you're tracking code-level metrics in a large environment instead of monitoring business transactions, the chances are that the fire you're troubleshooting is going to roar out of hand before you're able to douse it. Troubleshooting via business transactions gives you the ability to:

- Immediately flag when something has gone wrong, due to the transaction violating its historical baseline performance
- Triage where the problem is occurring – what node, for example, is being hampered by poor performance
- Diagnose the problem (a call to the SQL database, perhaps?)
- Troubleshoot down to the very line of code

Dramatically reducing Mean-Time-to-Resolution cycles is how business transactions support application owners. Multiple code drops are nothing to fear if resolving performance problems ends up being a matter of minutes, rather than torturous hours of late-night firefighting.

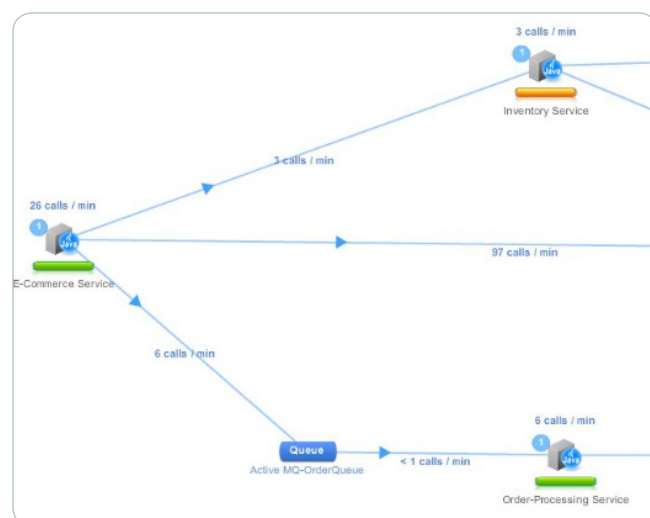
USE CASES: MANAGING PERFORMANCE WITH BUSINESS TRANSACTIONS

Once you're monitoring business transactions, you gain the ability to determine your application's performance baseline, identify SLAs, and significantly reduce the time it takes to drill-down and troubleshoot problems – even in a distributed environment. Let's look at a few examples of how business transactions support the application owner's ability to manage performance.

Use Case #1: You can't manage because you can't see

The most basic use case for business transactions is to get a true picture of your application environment. As strange as it sounds, AppDynamics hears this from our customers all the time: after they see a complete application map, they're surprised by what the topology reveals. They'll say, "Why is that call going to THAT database? And why is it taking so long?"

If your e-commerce service is making multiple calls to an inventory service, it could be the point of the bottleneck – but unless you understand how transactions are flowing, it may take you too long to reach that conclusion. And that's just in regards to a single hop. When it comes to multiple hops over a distributed system, it becomes even harder to understand how the architecture of the application may be affecting performance.



Use Case #2: You don't know where to start when firefighting

Business transactions can help you with one of the most important components of firefighting: prioritization.

If everything in your application is slow, how do you know where to look? Should you treat every node equally, and look through log files until you find a problem?

If you're focusing on business transactions, you have a better way. You can zero in on the business transactions that appear to be problematic – such as the one performing 8 calls a minute when the standard SLA is much faster than that.

Even in a distributed environment, you don't need to hunt for a needle in a haystack. You can zero in on the transactions that are the most likely culprits in regards to poor performance.

Name	Calls / min	Time (ms)	Health
View Cart	320	2	Good
Product Lookup	321	15	Good
Present Items	321	4	Good
Logout	320	0	Good
Login	320	16	Good
Checkout	320	137	Good
Add to Cart	320	96	Warning
/appdynamicspilot/WEB-INF/presentation/...	320	2	Good
/appdynamicspilot/WEB-INF/presentation/...	320	0	Good
/appdynamicspilot/	320	2	Good

Case #3: You need to get surgical

Once you've determined where the performance problem is, you need to do a lot better than say "Well, it's the Request a Quote transaction." In order to ensure rapid collaboration with your colleagues on the development side, you need to get to the line of code that's causing the issue.

Business transactions allow you to do this because they've helped you triage the situation and locate where the performance problem is actually occurring. As a result, it's relatively simple to get to code-level quickly, turning root cause analysis into a matter of minutes rather than hours.

In this case, it's a SpringBeans call to an external database that's taking up too much time, and is likely the cause of the situation:

Name	Time (ms)	External Calls
Servlet - POJOServlet:service	11 ms (self) 2.2 %	
Servlet - POJOServlet:doGet:32	0 ms (self) 0 %	
com.appdynamics.testappserver.spring.SpringBean1:businessMethod:17	0 ms (self) 0 %	
com.appdynamics.testappserver.spring.SpringBean2:businessMethod:15	0 ms (self) 0 %	
com.appdynamics.testappserver.spring.SpringBean3:businessMethod:16	0 ms (self) 0 %	
com.appdynamics.testappserver.spring.SpringBean4:businessMethod:21	0 ms (self) 0 %	
com.appdynamics.testappserver.spring.SpringBean5:businessMethod:17	0 ms (self) 0 %	
com.appdynamics.testappserver.spring.SpringBean6:businessMethod:16	0 ms (self) 0 %	
com.appdynamics.testappserver.spring.SpringBean7:businessMethod:16	0 ms (self) 0 %	
com.appdynamics.testappserver.spring.SpringBean8:businessMethod:16	0 ms (self) 0 %	
com.appdynamics.testappserver.spring.SpringBean10:businessMethod:17	128 ms (total) 25.2 %	JDBC
com.appdynamics.testappserver.runtime.database.DBExecutor.executeQueryBundle:27	41 ms (total) 8.1 %	JDBC
com.appdynamics.testappserver.runtime.database.DBExecutor.executeQueryBundle:27	0 ms (self) 0 %	
com.appdynamics.testappserver.runtime.Sleep:sleep:17	6 ms (total) 1.2 %	
com.appdynamics.testappserver.runtime.database.DBExecutor.executeStatements:63	10 ms (total) 2 %	JDBC

When Application owners are able to identify the line of code causing the problem, and give that code to their Development team, they contribute to rapid iterations that help ensure strong application performance.

More About Business Transactions (for Techies Only)

To get a bit more technical: business transactions represent the entry and exit points into applications – based on popular technologies like servlets & struts.

The first step for finding transactions within your application is to set up rules for identifying servlets by URI, struts actions by action names, and web service end points by service names. In the case of POJOs, you will want to use attributes such as class names, interfaces implemented, or super class names. These rules would be based on “entry points” – which represent the first significant landing points in the application. Such entry points identify the context for monitoring in the system; they also help you understand whether the user experience represented by the transaction was successful, or whether it contained errors.

You should identify a business transaction at the point where you can uniquely identify a particular category of user requests. For example, in a shopping cart application, the point where you can decide if the request belongs to a “Checkout” or an “Add to Cart” transaction becomes the point of identification. In a Struts application, this will typically belong to the Struts component layer – for example, the Checkout Struts Action. In a web service application it will belong to the web service such as the Checkout web service. In a pure servlet application, it will belong to the servlet layer. If it were a POJO based custom container, it can be com.foo.CheckoutProcessor or something similar that uniquely identifies the user request as being “Checkout.”

The right application monitoring solution should be able to do most of this work for you, identifying entry/exit points as well as bucketing business transactions automatically. Then you would simply perform a little more configuration to identify additional transactions you care about. The result? Snippets of code are transformed into business-friendly transactions that you can monitor and troubleshoot.

SUMMARY: THE POWER OF BUSINESS TRANSACTIONS

Application owners are under extraordinary pressure to incorporate frequent code changes while still being held responsible for 100% application uptime and performance. In a distributed and rapidly changing environment, meeting these high expectations becomes tremendously challenging.

A strong focus on business transactions becomes absolutely essential for maintaining application performance. Transaction-centric monitoring provides the basis for a stable performance assurance metric, it delivers powerful insights into user experience, and it ensures the ability to know where to hunt during troubleshooting.

The right Application Performance Management (APM) solution can automate much of this work. It can help application owners identify and bucket their business transactions, as well as assist with triage, troubleshooting, and root cause diagnosis when transactions violate their performance baselines. In this way, business transactions are essential to ensuring the success of Developers, Operations, and Architects – anyone with a stake in application performance.

ABOUT APPDYNAMICS

AppDynamics is the leading provider of application management for modern application architectures in both the cloud and the data center, delivering solutions for highly demanding distributed, dynamic, and agile environments. We offer simplicity, ease-of-use, and built-in intelligence via our patent-pending Dynamic Flow Mapping™ and Deep-on-Demand Diagnostics™, which continuously discover and auto-instrument production applications. Companies such as Williams-Sonoma, Netflix, Nationwide Insurance, and Deutsche Bank use AppDynamics to monitor, troubleshoot, diagnose, and scale their production applications – gaining 10x their current level of visibility and getting to root cause 90% faster. Try our free java performance solution at www.appdynamics.com/free or find out more at www.appdynamics.com.



AppDynamics
303 Second Street, Suite 450 | San Francisco, CA 94107
www.appdynamics.com