Service Virtualization:
The Next Generation of Test Environment Management

Today's complex, interdependent systems wreak havoc on parallel development and functional/performance testing efforts—significantly impacting productivity, quality, and project timelines. As systems become more complex and interdependent, development and quality efforts are further complicated by constraints that limit developer and tester access to realistic test environments. These constraints often include:

- Missing/unstable components
- Evolving development environments
- Inaccessible 3rd party/partner systems and services
- Systems that are too complex for test labs (mainframes or large ERPs)
- Internal and external resources with multiple "owners"

Although hardware and OS virtualization technology has provided some relief in terms of reduced infrastructure costs and increased access, significant gaps still exist for software development and testing. It is not feasible to leverage hardware or OS virtualization for many large systems such as mainframes and ERPs. And more pointedly, configuring and maintaining the environment and data needed to support development and test efforts still requires considerable time and resources. As a result, keeping complex staged environment in synch with today's constantly-evolving Agile projects is a time-consuming, never-ending task.

This paper introduces Service Virtualization: a new way to provide developers and testers the freedom to exercise their applications in incomplete, constantly evolving, and/or difficult-to-access environments. Rather than virtualizing entire applications and/or databases, Service Virtualization (also known as "Application-Behavior Virtualization") focuses on virtualizing only the specific behavior that is exercised as developers and testers execute their core use cases. Although the term "Service Virtualization" was originally coined to reflect the initial focus on emulating web services, it currently extends across all aspects of composite applications—services, mainframes, web and mobile device UIs, ERPs, ESB/JMS, legacy systems, and more.

This new breed of virtualization—which is entirely complementary to traditional virtualization—radically reduces the configuration time, hardware overhead, and data management efforts involved in standing up and managing a realistic and sustainable dev/test environment.

The Complexity of Quality

In today's development environments, the scope of what needs to be tested is increasing exponentially. With multiple new interfaces and ways for people to access core technology, systems and architectures have grown broader, larger, and more distributed—with multiple endpoints and access points. For example, you might have a thick client, a web browser, a device, and a mobile application all accessing the same critical component. Not surprisingly, testing in this environment has become very difficult and time consuming.

Furthermore, the number and range of people involved with software quality is rising. Advancements in development methodologies such as Agile are drawing more and more people into quality matters throughout the SDLC. For instance, Business Analysts are increasingly involved with user acceptance testing, QA has become responsible for a broader and more iterative quality cycle, and the development team is playing a more prominent role in the process of software quality and validation. Moreover, today's large distributed teams also exhibit a similar increase in team members involved with quality.

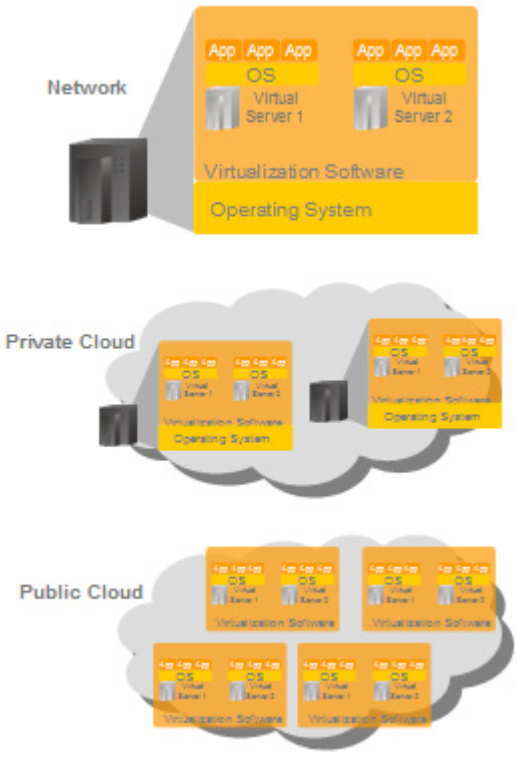
Also increasing are the permutations of moving parts—not only hardware and operating systems, but also client server system upgrades, patches, and dependent third-party application. As the service-oriented world broke apart many monolithic applications, service orientation also increased and distributed the number of connections and integration points involved in executing a business process.

Hardware and OS Virtualization Lowers Cost & Increases Access—Yet Significant Gaps Remain

In an attempt to provide all of the necessary team members ubiquitous access to realistic dev/test environments in light of these complexities, many organizations have turned to hardware and OS virtualization. Virtualizing the core test foundations—specific operating systems, configurations, platforms, etc.— has been a tremendous step forward for dev/test environment management. This virtualization provides considerable freedom from the live system, simultaneously reducing infrastructure costs and increasing access to certain types of systems. Moreover, leveraging the cloud in concert with virtualization provides a nearly unlimited bandwidth for scaling dependent systems.

Nevertheless, in terms of development or test environments, some significant gaps remain. First of all, some assets cannot be easily virtualized. For example, it is often unfeasible to leverage hardware or OS virtualization technology for large mainframe applications, third-party applications, or large ERPs.

Moreover, even when virtualization can be completed, you still need to configure and manage each one of those applications on top of the virtualized stack. Managing and maintaining the appropriate configuration and data integrity for all the dependent systems remains an ominous and time-consuming task. It is also a task that you will need some outside help with—you will inevitably be relying on other groups, such as operations or DevOps, to assist with at least certain aspects of the environment configuration and management.



Service Virtualization reduces this configuration and data management overhead by enabling the developer or tester to rapidly isolate and virtualize just the behavior of the specific dependent components that they need to exercise in order to complete their end-to-end transactions. Rather than virtualizing entire systems, you virtualize only specific slices of dependent behavior critical to the execution of development and testing tasks.

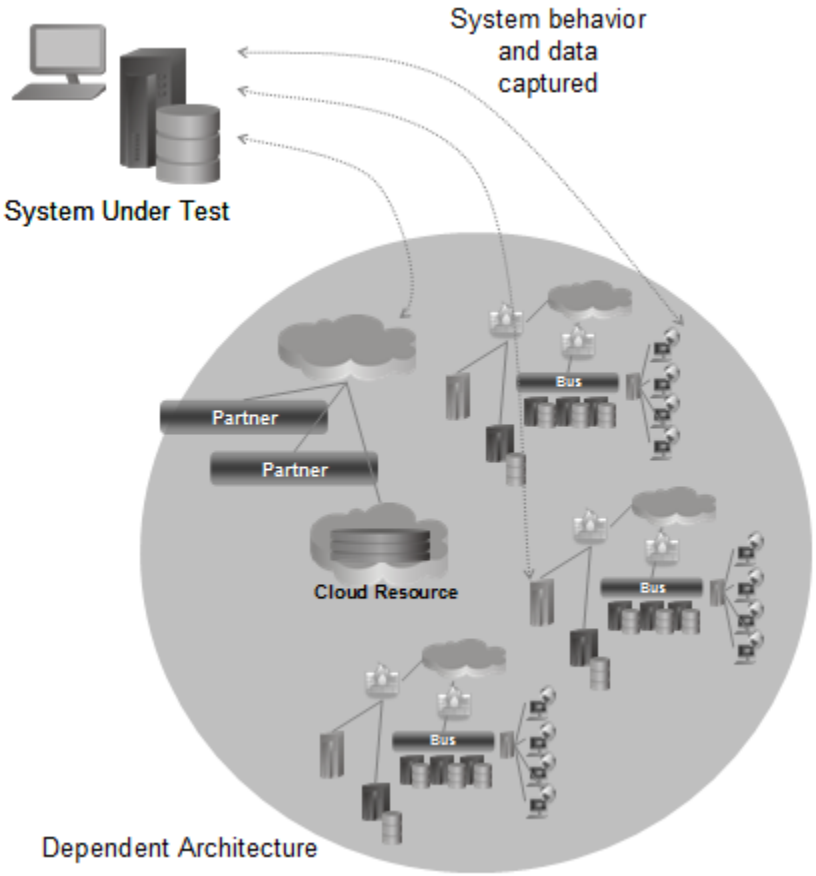
It is completely feasible to use the cloud for scalability with Service Virtualization. Nevertheless, since you're virtualizing only the specific behavior involved in dev/test transactions (not entire systems), the scope of what's being virtualized is diminished… and so is the need for significant incremental scalability.

What is Service Virtualization?

Service Virtualization is a more focused and efficient strategy for eliminating the system and environment constraints that impede the team's ability to test their heterogeneous component-based applications. Instead of trying to virtualize the complete dependent component—the entire database, the entire third-party application, and so forth—you virtualize only the specific behavior that developers and testers actually need to exercise as they work on their particular applications, components, or scenarios.

For instance, instead of virtualizing an entire database (and performing all associated test data management as well as setting up the database for each test session), you monitor how the application interacts with the database, then you virtualize the related database behavior (the SQL queries that are passed to the database, the corresponding result sets that are returned, and so forth). This can then be accessed and adjusted as needed for different development and test scenarios.

To start, you designate which components you want to virtualize, then—as the application is exercised—the behavior of the associated transactions, messages, services, etc. is captured in what we call a "virtual asset." You can then configure this virtual asset by parameterizing its conditional behavior, performance criteria, and test data. This virtual asset can then emulate the actual behavior of the dependent system from that point forward—even if the live system is no longer accessible for development and testing.



Test data can be associated with these virtual assets, reducing the need for a dependent database and the need to configure and manage the dependent database that, if shared, usually gets corrupted.

By applying Service Virtualization in this manner, you can remove the dependency on the actual live system/architecture while maintaining access to the dependent behavior. This ultra-focused approach significantly reduces the time and cost involved in managing multiple environments—as well as complex test data management.

What Does Service Virtualization Involve?

Service Virtualization is achieved via the following phases:

- Capture or model the real behavior of dependent systems
- Configure the virtualized asset to meet demands of the test scenarios
- Provision the virtualized asset for the appropriate team members or partners to access and test on their schedule

Phase 1: Capture

Real system behavior is captured—using monitors to record live transaction details on the system under test; by analyzing transaction logs; or by modeling behavior from a simple interface.

The intent here is to capture the behavior and performance of the dependent application for the system under test and leverage that behavior for development and testing efforts. This capturing can be done in three ways:

- If you have access to the live system, you can capture behavior by monitoring live system traffic. With a proxy monitoring traffic on the dependent system, the related messages are monitored, then the observed behavior is represented in a virtualized asset. This capturing can cover simple or composite behavior (e.g., a call to transfer funds in one endpoint can trigger an account balance update on another).
- If you want to emulate the behavior represented in transaction logs, virtual assets can be created by analyzing those logs. This is a more passive (and less politically volatile) approach to capturing the system behavior.
- If you're working in an environment that is evolving to include new functionality, you might want to model the behavior of the "not yet implemented" functionality within the Service Virtualization interface. Leveraging the broad scope of protocol support available to facilitate modeling, you can rapidly build a virtual asset that emulates practically any anticipated behavior. For instance, you can visually model various message formats such as XML, JSON, and various legacy, financial, healthcare, and other domain-specific formats.

Phase 2: Configure

The virtualized asset's behavior can be fine-tuned, including performance, data source usage, and conditional response criteria.

After you use any of the three above methods to create a virtual asset, you can then instruct that asset to fine-tune or extend the behavior that it emulates. For instance, you can apply Quality of Service metrics so you can alter how you would like the asset to behave from the performance (timing, latency, and delay) perspective. You can also apply and modify test data for each particular asset to reproduce specific conditions critical for completing dev/test tasks. For example, you can configure various error and failure conditions that are difficult to reproduce or replicate with real systems. By adding data sources and providing conditional response criteria, you can tune the virtualized asset to perform as expected—or as unexpected (for negative testing).

Phase 3: Provision and Test

The environment is then provisioned for secure access across teams & business partners. The virtualized asset can then be leveraged for testing.

Once a virtualized asset is created, it can be provisioned for simplified uniform access across teams & business partners—either locally or globally (on a globally-accessible server, or in the cloud). They can then be used in unit, functional, and performance tests. Since virtual assets leverage a wide array of native protocols, they can be accessed for manual testing or automated testing by any test suite or any test framework, including Parasoft Test, HP Quality Center suite, IBM Rational Quality Management suite, Oracle ATS, and more. It is also easy to scale virtualized assets to support large-scale, high-throughput load and performance tests.

Even after the initial provisioning, these virtual assets are still easily modifiable and reusable to assist you in various dev/test scenarios. For instance, one of your test scenarios might access a particular virtual asset that applies a certain set of conditional responses. You can instantly construct an additional virtual asset that inherits those original conditions, then you can adjust them as needed to meet the needs of a similar test scenario.

How Service Virtualization Speeds Testing & Cuts Costs: 3 Common Use Cases

To conclude, let's look at how organizations have successfully applied Service Virtualization to address dev/test environment management challenges in three common contexts:

- Performance/capacity-constrained environment
- Complex, difficult-to-access systems (mainframes, large ERPs, 3rd party systems)
- Parallel development (Agile or other iterative processes)

Performance/Capacity-Constrained Environments

Staged environments frequently lack the infrastructure bandwidth required to deliver realistic performance. Placing multiple virtualized applications on a single piece of hardware can increase access to a constrained resource, but the cost of this increased access is often degraded performance. Although the increased access could technically enable the execution of

performance and load tests, the results typically would not reflect real-world behavior, significantly undermining the value of such testing efforts.

Service Virtualization allows you to replicate realistic performance data independent of the live system. Once you create a virtual asset that captures the current performance, you can adjust the parameters to simulate more realistic performance. Performance tests can then run against the virtual asset (with realistic performance per the Quality of Service agreement) rather then the staged asset (with degraded performance).

Controlling the virtual asset's performance criteria is simply a matter of adjusting controls for timing, latency, and delay. In addition to simulating realistic behavior, this can also be used to instantly reproduce performance conditions that would otherwise be difficult to setup and control. For instance, you can simulate various levels of slow performance in a dependent component, then zero in on how your application component responds to such bottlenecks.

Even when it is possible to test against systems that are performing realistically, it is often not feasible to hit various components with the volume typical of effective load/stress tests. For example, you might need to validate how your application responds to extreme traffic volumes simulating peak conditions—but how do you proceed if your end-to-end transactions pass through a third-party service that charges per-transaction access fees?

If your performance tests pass through a component that you cannot (or do not want to) access under extreme load testing conditions, Service Virtualization enables you to capture its behavior under a low-volume test (e.g., a single user transaction), adjust the captured performance criteria as desired, then perform all subsequent load testing against that virtualized component instead of the actual asset. In the event that the constrained component is not available for capture, you can create a virtual asset from scratch—using Service Virtualization visual modeling interfaces to define its expected behavior and performance.

Complex, Difficult-to-Access Systems (Mainframes, Large ERPs, 3rd Party Systems)

With large complex systems (mainframes, large ERPs, third party systems), multiple development and test teams are commonly vying for limited system access for testing. Most of these systems are too complex for a test lab or a staged environment. To exercise end-to-end transactions involving these components, teams usually need to schedule (and pay for) access to a shared resource. This approach commonly causes test efforts to be delayed and/or prevents the team from performing the level and breadth of testing that they would like. For iterative development processes (e.g., Agile), the demand for frequent and immediate testing increases the severity of these delays and fees exponentially.

Even if organizations manage to use virtualization for these complex systems, proper configuration for the team's distinct testing needs would require a tremendous amount of work. And once that obstacle is overcome, another is right on its heels: developing and managing the necessary set of test data can also be overwhelming.

When teams use Service Virtualization in such contexts, they only need to access the dependent resources long enough to capture the specific functionality related to the components and transactions they are working on. With this behavior captured in virtual assets, developers and testers can then access it continuously, allowing them to exercise end-to-end transactions at whatever time they want (without scheduling) and as frequently as they want (without incurring exorbitant transaction/access fees).

Parallel Development (Agile or other Iterative Processes)

Even for simple applications, providing continued access to a realistic test environment can be challenging for teams engaged in parallel development (Agile or other iterative processes). A wide range of team members—including developers, testers, sometimes business analysts—all need easy access to a dev/test environment that is evolving in synch with their application. If the team decided to take the traditional virtualization route here, they would not only face all the initial setup overhead, but also be mired in constant work to ensure that the virtualized systems remain in step with the changes introduced in the latest iteration. When the team ends up waiting for access to dependent functionality, agility is stifled

Service Virtualization reduces these constraints and associated delays by giving developers and testers the ability to rapidly emulate the needed behavior rather than having to wait for others to upgrade, configure, and manage the dependent systems. Even if anticipated functionality or components are not yet implemented, their behavior can be modeled rapidly then deployed so team members can execute the necessary end-to-end transactions without delay, And if the dependent functionality recently changed, previously-captured behavior can be easily modified—either by re-capturing key transactions or by adjusting behavior settings in a graphical interface (without scripting or coding).

For example, many organizations are developing mobile applications, and this development is typically performed by a separate mobile development team. Since mobile applications commonly depend on core application components developed and maintained by other teams, the mobile team is often delayed as they wait for the other teams to complete work on the core components that their own mobile apps need to interact with. Service Virtualization can eliminate these delays by allowing the mobile development team to emulate the behavior of the dependent components—even if the actual components are incomplete, evolving, or otherwise difficult-to-access during the parallel development process.

Key Takeaways

Leveraging Service Virtualization, teams reduce the complexity and the costs of managing multiple environments while providing ubiquitous access for development and test. Service Virtualization helps you:

- Reduce infrastructure costs
- Improve provisioning/maintenance of test environments
- Increase test coverage
- Reduce defects
- Improve predictability/control of software cycle times
- Increase development productivity
- Reduce 3rd party access fees

To learn more about how Parasoft implements Service Virtualization in Parasoft Virtualize, visit the Parasoft Virtualize center.

About Parasoft

For 25 years, Parasoft has researched and developed software solutions that help organizations deliver defect-free software efficiently. By integrating end-to-end testing, dev/test environment management, and software development management, we reduce the time, effort, and cost of delivering secure, reliable, and compliant software. Parasoft's enterprise and embedded development solutions are the industry's most comprehensive—including static analysis, functional testing with requirements traceability, service virtualization, and more. The majority of the Fortune 500 companies rely on Parasoft in order to produce top-quality software consistently and efficiently. For more information, visit the Parasoft web site and ALM Best Practices blog.

Author Information

This paper was written by:

- Wayne Ariola (wayne.ariola@parasoft.com), VP of Strategy at Parasoft
- Cynthia Dunlop (cynthia.dunlop@parasoft.com), Lead Technical Writer at Parasoft

Contacting Parasoft

USA
101 E. Huntington Drive, 2nd Floor
Monrovia, CA 91016
Toll Free: (888) 305-0041
Tel: (626) 305-0041
Fax: (626) 305-3036
Email: info@parasoft.com
URL: www.parasoft.com

Europe
France: Tel: +33 (1) 64 89 26 00
UK: Tel: + 44 (0)208 263 6005
Germany: Tel: +49 731 880309-0
Email: info-europe@parasoft.com

Asia
Tel: +886 2 6636-8090
Email: info-psa@parasoft.com

Other Locations
See http://www.parasoft.com/contacts