WHITE PAPER | NOVEMBER 2014

Eliminate Software Development and Testing Constraints with Service Virtualization

Table of Contents

Executive Summary

Challenge

The growing complexity of applications, along with globally-dispersed organizations, means teams face a barrage of bottlenecks as they strive to deliver. Examples of bottlenecks—or constraints—include access to mainframe or ERP test environments, lack of test data, access to third-party systems, and budget limitations. Often, constraints are created parallel development teams seeking access to the same environments. Often, teams try to either copy a complete environment in a test lab, or "mock up" responsive systems by coding their own versions for test purposes. This can become a costly and time-consuming endeavor.

Opportunity

In essence, Service Virtualization (SV) is the productization of the practice of "mocking and stubbing" dev and test environments, with enough realism and context to push development forward faster, while **shifting testing left** in the lifecycle so integration and release processes can happen faster, with higher quality and less risk.

Benefits

What follows are a number of capabilities your organization should look for in a Service Virtualization solution, enabling your extended teams to get high quality applications to market faster, with lower cost and risk. These include:

Provides development with a more live-like environment	Higher quality development and more effective regression/system testing
Enables parallel development and testing	Reduce cycle times, earlier defect discovery, efficient use of resources
Virtualizes test data for out-of-scope systems	Faster setup/teardown, more stability for test automation
Enables high-performance environments	Better realism and quantity of performance testing at dramatically lower cost

Since no application development and testing environment is an island unto itself, it is also critical that Service Virtualization solutions provide a "vendor neutral" substrate for the tools of choice that teams may have in place. Service Virtualization should provide target environments that work alongside existing application lifecycle solutions such as Test Management (TM), defect management/issue tracking and leading hardware and test lab virtualization products that exist in the environment.

Section 1: Creating a "Live-like" Environment

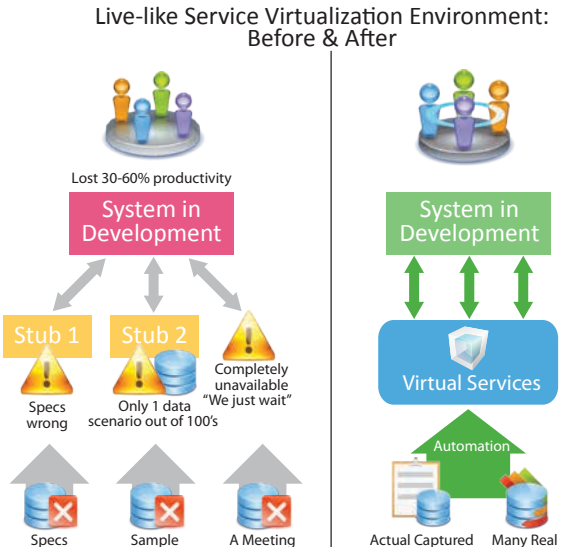
Simulating Dependent Systems in Dev and Test Environments

As application development trends toward more composite, service-oriented architecture approaches, a much wider variety of upstream and downstream systems must be simulated by teams in their dev and test environment. Service Virtualization should be applied at or between any layer where dependencies exist, to provide the most realistic or "live-like" possible environment.

In the conventional approach, teams try to move forward with their own component development by "stubbing" the next downstream system only.

For instance, if I am developing a web UI, I build a stub for a couple expected responses from the next layer down, (i.e. the web service). Then the web service developers might stub out their underlying ESB layers, or try to mock up some user requests from the web UI. Unfortunately, this is a manual process that is never sufficient to encapsulate the many types of connections and data that exist within enterprise software architectures, and may be completely unavailable if there's not yet a UI coded, as seen below.

Figure A.



Alternatively, when teams are working with real data scenarios and real behaviors captured as Virtual Services, their productivity level is higher, as the resulting environment is far more realistic and current than sets of stubs that must be manually coded and maintained.

Therefore, the critical technique that enables "live-like" environments is automation of virtual service creation and data maintenance. This enables development teams to be far more productive due to realistic virtual lab environments, even if the user interface is incomplete, while spending less time working to create or modify outdated stubs.

Expected benefits:

- Ability to start development despite interface system unavailability
- Reduced cycle time for test execution
- Ability to improve the test coverage because of reduced data dependency on other applications and available testing time
- Improved unit testing with lesser effort
- Improved code quality due to increased test coverage and regression testing
- Ability to build a simulator quickly with low maintenance effort

---

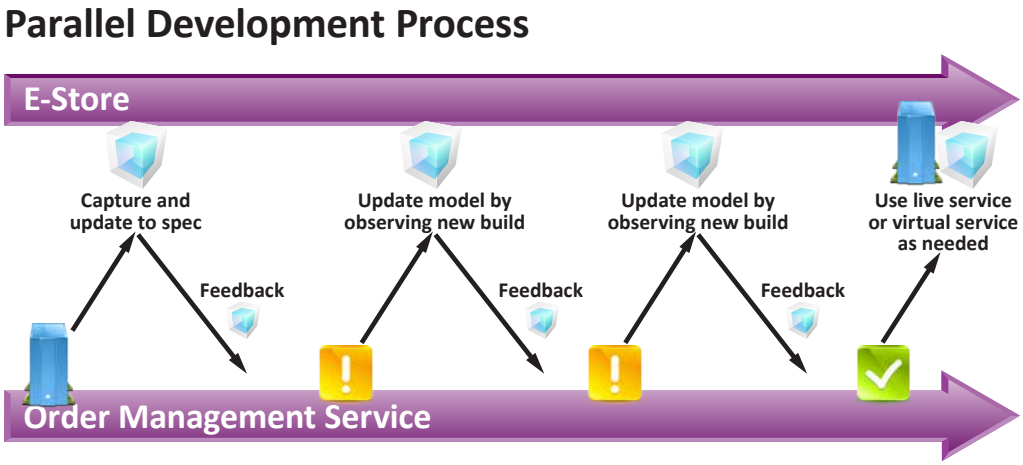
Section 2: Enabling Parallel Development and Testing

Dev and Test Teams Working Simultaneously

The second critical capability of a Service Virtualization solution is enabling parallel development and testing. When dev and test teams work simultaneously, the overall software lifecycle reaches a whole new level of efficacy and efficiency. New solutions can be delivered at great value to the organization.

In parallel dev and test, Virtual Services act as the "go between" assets between the system under development, and the system under test in a symbiotic fashion. In the example below, a team is developing an order management service (OMS) while the team at the top is developing and testing an "e-store" app.

Figure B.



A Virtual Service is captured from the existing OMS system as an initial back-end for the e-store's testing activity. Then as the testing continues, the e-store team can communicate back any unexpected or new response requirements as "feedback" virtual service requests that essentially become a next set of requirements for development. Each parallel dev and test cycle continues to accelerate as each iteration of Virtual Service model updates happens with each new build, and feedback happens faster and faster.

The perfect parallel development solution allows teams to execute against the live services where they are available, functionally robust, and data synchronized. And in those cases where teams do not yet have services that support the component correctly, they can switch immediately back to virtual services. This ability to flip a switch and go between a purely virtual downstream system through virtual services, or to the live system, knowing that you always can go back if a new build breaks, or if a new data scenario is required, is a very powerful asset to creating a robust parallel development capability.

Is your practice of generating and updating software lab environments keeping in touch with reality? In essence, the parallelism of Service Virtualization enables the agility we expected of Agile for complex software environments—by more closely aligning dev and test cycles with business release goals in tight iterative releases.

Expected benefits:

- Ever-increasing speed of test and development cycles
- Enables true responsiveness of Agile iterations, with continuous integration and builds aligned around test results and business requirements
- Reduced burden of version control, works with existing ALM and development management tools to make them more effective
- Increased rate of issue acceptance and resolution prior to production
- Deliver function points up to 60% faster, with higher quality and accuracy to specification

---

Section 3: Handling Test Data for Out-of-Scope Dependencies

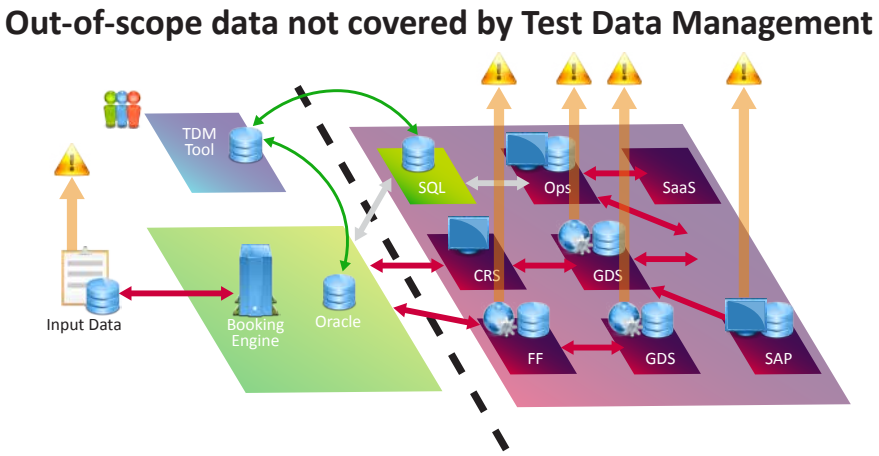
Missing Downstream Data is No Longer a Problem

Every team needing to provision a lab for their activities has some systems and associated data that are considered in-scope, and others that are out-of-scope. An in-scope system is one in which they are performing a development change or test directly on that system. An out-of-scope system is one that is required in support of that in-scope system, but is in fact not the subject of the development or testing activity. It is considered a dependency. It is necessary, but it is not the subject of the development or test activity.

In this scenario, the team developing and testing a booking engine is attempting to aggregate the needed downstream and user input data scenarios, but they don't have keys to the whole kingdom.

The most obvious solution is the usual practice of Test Data Management, importing data directly from the in-scope systems, and "stubbing" or mocking those out-of-scope systems by attempting to write code and import a couple lines of data to represent the expected responses of those out-of-scope dependencies. These stubs are inherently brittle, and developers take the most expedient route in simulating the basic functionality or response scenarios they expect. However, as the complexity of today's distributed software increases, the effort of manually coding and maintaining useful stubs has become far too costly.

Figure C.

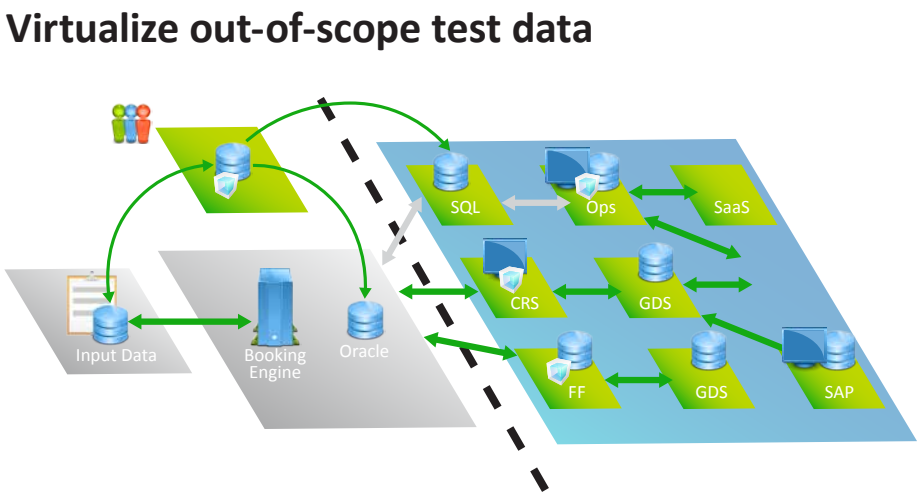


Is it real, or virtual data?

What we need is a way to simulate the behavior of those out-of-scope systems with enough intelligence so that the in-scope system believes it is talking to the live system, but is in fact not.

Service Virtualization takes the approach of making all that missing data behind the in-scope system no longer a problem, by automating the capture of relevant downstream scenarios that are out of scope.

Figure D.



The end-state example of virtualized Test Data Management shown here looks almost too simple, but this is actually how your testing process should interact with external or out-of-scope dependencies. Virtual models allow all your teams to always have on-demand access to relevant datasets for systems under test, and that data will cover almost infinite valid data scenarios to support high-volume performance and regression testing needs.

Service Virtualization capabilities should include mechanisms to bring all of the systems needed into the development and test lab environment, including provisioning the data of out-of-scope systems, and maintaining "stateful" transaction data as it passes between systems in a workflow over time. For instance, simulating the changing of values such as dates or cumulative amounts in a long-running transaction so these kinds of complex workflows can be validated at a necessary level of detail.

Expected benefits:

- Eliminates delays due to lack of access or current data from out-of-scope systems
- Availability of valid test scenarios 24/7 for multiple test and development teams
- No conflicts over test data or invalidation of other teams' activities by overwriting or changing data in systems
- Little or no impact on critical live systems
- Support for "stateful" transactions to support complex processes that cross multiple technology layers and need to maintain specific values such as dates, customer IDs, etc.
- Reduced time spent on data setups and resets by up to 90%, cutting overall test lifecycle times by 40-60%

Section 4: Supporting Heterogeneous Technology and Platforms

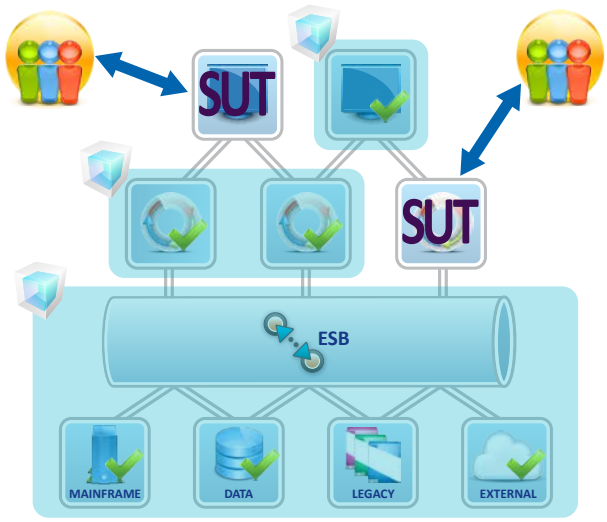
Creating a Truly Complete Environment

Recently I visited the architecture team of one of the world's largest banks, when I heard a stunning statistic. This gentleman said their hardware asset management system claims there are more servers deployed in the bank than there are employees in the bank. Every project team would justify the expense of its own development, testing, pre-production, and production hardware expenditures. Most every application currently in use at the bank has at least these four environments behind it, even though the maintenance or changes on those applications in many cases go months or years between releases.

Ironically, the greatest challenge most existing pre-prod environments have is that they are never the complete system. A project team can purchase any number of servers, and use them to replicate some hardware or components on VMs they have access to. But even though every team allocated their own hardware budget, they still spend countless months of their development cycle waiting on access, and inefficiently accessing shared system resources.

Heterogeneous systems are the norm in enterprise IT environments. Therefore, Service Virtualization should be applied to virtualize any and all dependencies that can impact the system under test (SUT). This includes web traffic (HTTP), web services (SOAP/XML), integration layers & ESB (JMS, etc.), as well as simulating transactions and connections with underlying mainframes (CICS, CORBA, etc.), databases (JDBC, etc.) and third-party services.

Figure E.



Since every connection point in your software architecture represents a potential point of change, and presents a potential risk of failure, it becomes critical that Service Virtualization provides a better way for teams to "virtualize everything else" and thereby isolate themselves from dependencies on these heterogeneous components.

Using Service Virtualization, hundreds of pre-production labs are folding into one vastly simpler to manage infrastructure, with software based provisioning on an on-demand basis for any of the required environments. Projects not currently under change will no longer consume power, generate heat, or consume floor space. Dependencies are captured and eliminated. Instead of needing live system access for the mainframe partition, the team will provision a virtual service of the customer information management system on-demand.

Expected benefits:

- Faster delivery through anytime 24/7 access to isolated lab environments for all dev and test teams
- Reduced cost of conventional pre-production infrastructure—potentially saving $20M+ per year for larger enterprises
- Eliminates service costs and fees of accessing remote systems
- Allows teams to "shift testing left" and build in quality at a component development and system integration level far before conventional user acceptance testing activities. Reduced time spent on data setups and resets by up to 90%, cutting overall test lifecycle times by 40-60%.

Also note that heterogeneity applies to process-level tools which teams use to collaborate, and not just the assets you virtualize. It is critical that Service Virtualization solutions provide a "vendor neutral" substrate for the tools of choice that teams may have in place. Service Virtualization should provide target environments that work alongside existing application lifecycle solutions such as Test Management (TM), Defect Management/Issue Tracking, Test Data Management and leading Hardware and test lab virtualization products that exist in the environment.

Section 5: Conclusion

Taking the Right First Steps

How many times have you approached your development or testing manager with the question: "Why didn't you deliver when you said you would?" Only to hear every reason for the delay was because of other teams and their presumed broken promises or challenges. Examples that have been related to us, and that I have experienced myself include:

- "Yeah, but we were supposed to get the build for the new order management system last month but we just got it last week."
- Or, "Yeah, but every new build of the order management system introduced some new things we needed but broke other things we already had working."
- Or, "Yeah, but the stub that other team gave me for order management only has one customer profile in it, so I can't build or test any of the other scenarios that I really need in order to get this thing done."

Ultimately, we can't finish our job until the other guys are finished with their "Yeah, but…" job we are depending on. Each team associated with a composite application must be free to construct its own lab from the infrastructure, on-demand. There will inevitably be intra-team dependencies. This is why Service Virtualization capabilities are so critical. Each team is free to take the specifications from its downstream dependencies and build the expected, To-Be state of its downstream dependencies, before they even need to see the first build of those downstream dependencies.

Service Virtualization provides an essential platform for enterprise software development and testing. There are many other details to consider when planning a strategy for SV enablement, especially on the organizational side. Who contributes to the solution, and who consumes it? Who owns and manages the virtual services within your extended organization of teams and partners? A well-rounded approach to Service Virtualization will take into account not just the technical details but the operational ones that can best guarantee widespread adoption and success.

Connect with CA Technologies at ca.com

CA Technologies (NASDAQ: CA) creates software that fuels transformation for companies and enables them to seize the opportunities of the application economy. Software is at the heart of every business, in every industry. From planning to development to management and security, CA is working with companies worldwide to change the way we live, transact and communicate – across mobile, private and public cloud, distributed and mainframe environments. Learn more at **ca.com**.

CS200-94676_1114