Building a protocol validator for Business to Business Communications

Rudi van Drunen, Competa IT B.V. (r.van.drunen@competa.com)
Rix Groenboom, Parasoft Netherlands (rix.groenboom@parasoft.nl)

Abstract

In this paper we describe the design and implementation of a system essential to enable the deregulation of the energy market in the Netherlands. The system is used to test and validate secure communications using XML messages through the AS2 standard between the business partners in the market. The tool is comprised of an Enterprise Service Bus component, a service virtualization component, a database with business logic and an user interface added. The version 1.0 of the system was built in less than one month.

1. Introduction

The energy market in the Netherlands has been deregulated. To facilitate this a communications structure is designed in which the different energy suppliers are able to communicate to each other. This communications structure and protocol is critical in the supply of energy (electricity, gas) from supplier to consumer. To validate these protocols a simulation and validation environment is required, and used for certification of the different market parties. This experience report discussed the implementation of such an certification environment for the gas transportation; more of these tools will be required for other market processes and will be constructed in a similar fashion.

The exchange of messages in the protocol is built up in different layers. The transport medium is the Internet, and every supplier has connected their production systems to the Internet, to reach all others. In the next layer the transport protocol is http, secured by ssl, so we're using standard https here. Then the data protocol is AS2, which provides a way to send (XML) data in an encrypted manner using S/Mime to an authenticated receiver. The actual payload is an XML message that should adhere to the Dutch energy interconnection standard.

The goal is to provide a test environment (in the project called Testtool) that can be used to certify over 100 market parties to see whether they adhere to a new XML definition that is being rollout during a migration process taking about one year. There are over fifty application and protocol level test-scenarios that need to be checked for each party before they are certified and can participate in the new communications infrastructure.

2. Architecture

To validate the complete communication stack the engine should be able to validate and test the communication on every level. Building this engine from scratch is a major undertaking, and will not be flexible. Therefore we choose a number of building blocks to technically implement the validation engine.

The https and AS2 communications are being handled by an Open Source Enterprise Service Bus (ESB) solution (UltraESB[1]), which then hands the XML payload to a product called Virtualize[2]. Virtualize is used as a service virtualization engine which tests the validity of the XML message, handles responses and stores the data in a database. The actual validation (as part of the certification process) is done using the data in the database and checking on the right contents and sequence of the XML payloads. The database is a MySQL database that stores all messages and metadata of the messages and the sender / receiver. All of this is driven by a number of Java classes that can be controlled by a Web GUI.

`Figure 1` shows the global architecture of the Testtool. The different messages (notification (NOT) and Message Delivery Notice (MDN) as Acknowledge) are asynchronously for external communications and synchronously for internal communications. The external party is simulated in the test environment as depicted using the Meldelson[3] tooling.

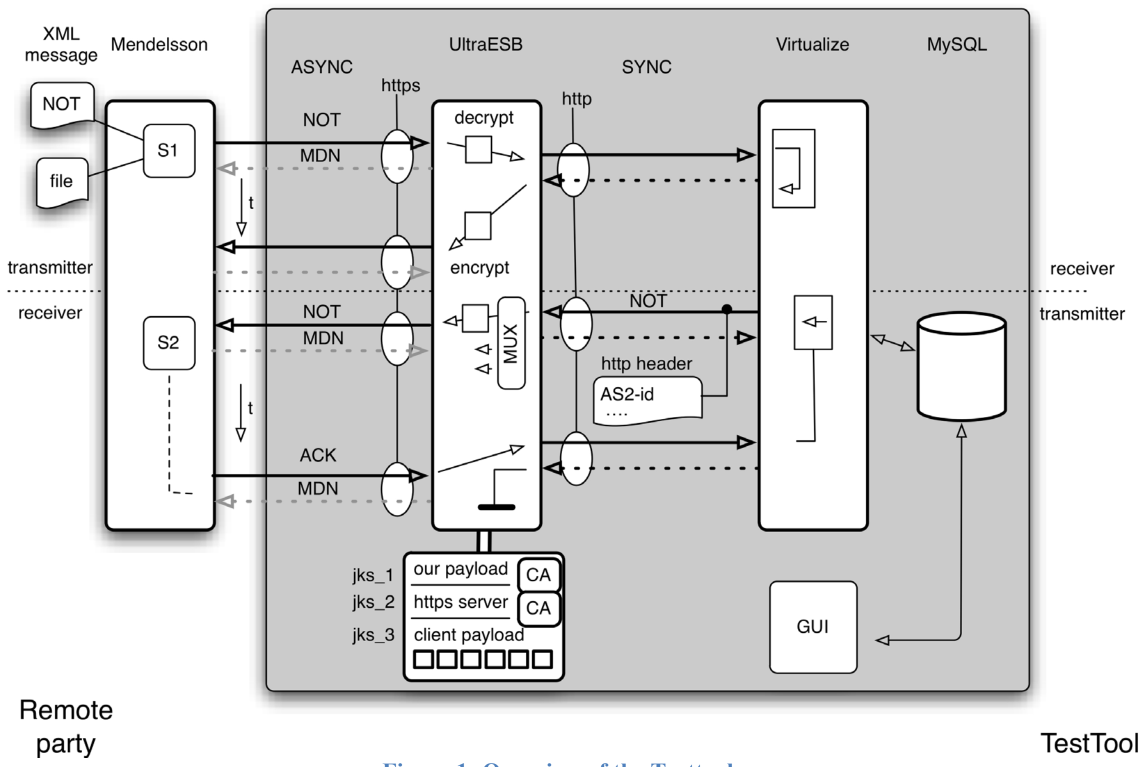


Figure 1: Overview of the Testtool

2.1 AS2

The protocol on AS2 level allows for multi level encryption of the data on the wire, as well as validation of the communication partners using certificates. Next to that communications can take place in a synchronous and asynchronous manner.

The AS2 message enters the system through a https connection. A standard server certificate, issued by a trusted certificate authority is used to validate the sender and decrypt the ssl transmission. The setup corresponds with the way an ssl-enabled webserver is set up. Important is that the Testtool, as well as the partner, can work both with one and two-way SSL.

As the ESB solution incorporates the webserver, the server certificate, as well as the (path to the) root CA needs to be present in the Java keystore it uses (a so-called `identity.jks`). This setup is shown in Figure 2.

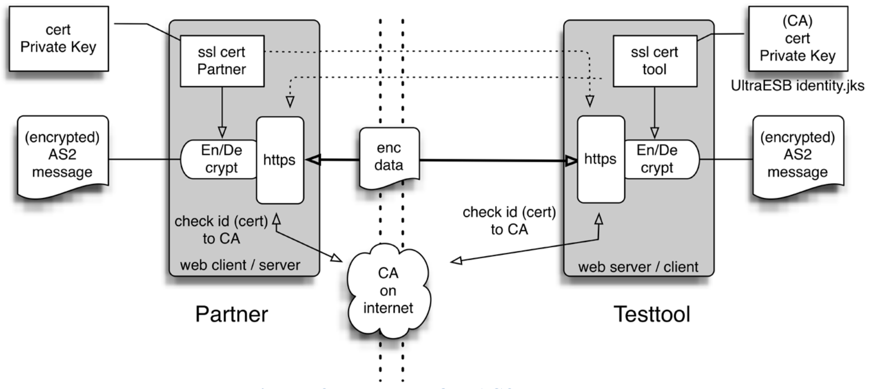


Figure 2: ssl setup for AS2 messages

The OpenSource ESB we employ in this project is UltraESB. This engine has the ability to handle the (AS2) communication with the partner in a synchronous and asynchronous manner, while it communicates in a synchronous fashion to the backend (in this case the Virtualize engine). Configuration and adding functionality to UltraESB turned out to be simple by changing the configuration files that are a mix of Java and XML.

As the protocol supports bi-directional communication we need to not only accept messages but also be able to send them. We needed to add extra functionality to the ESB to facilitate sending messages to partners, when the test environment is initiating the communication. We did this by adding functionality to select the correct certificate that belongs to the business partner with whom we need to set-up the AS2 communication. A custom http-header field: `dest-endpoint` is used to communicate this information from Virtualize to UltraESB.

Second, when we are receiving the, in this case asynchronous, acknowledgement message we should end the handshake directly and not send a formal reply from the backend to the business partner again. In that situation, the ESB would not forward the 'response' message, but direct it to "ground". We used this approach to have a clear separation of concern: the ESB deals only with the communication, only the Virtualize components are 'aware' of the type of message. If it is an incoming notification message that needs acknowledgement, Virtualize responds. If the incoming message is an acknowledgement we do not have to respond Virtualize adds the "ground" designator to the response message.

As there were no throughput requirements, as the system was to handle one communication at a time, with a payload maximum of about 1 Mbyte. We did not expect issues with buffering, and relied on internal buffers of the ESB. In practice this has proven to be sufficient.
The ESB allowed us to set the maximum payload size, which will be set to 10 Mbytes in the production version. Together with the appropriate settings of the java VM (max heap size) we were able to handle this maximum payload size without problems.

2.2 PKI

To authenticate and encrypt / decrypt the XML messages on AS2 level a Public Key Infrastructure is being used as shown in Figure 3. Every partner in the energy market in the Netherlands has a certificate that is signed by a root that is public for the Dutch energy market, and is maintained by the government. This PKI is used in the ESB software to maintain authentic and safe communications. The path to and the root CA, as it is not a standard trusted CA as well as the certificates of every business partner must be known to the ESB application. The local certificate must be made known to every partner as well. The ESB certificates are maintained in a different Java keystores and controlled by a GUI.

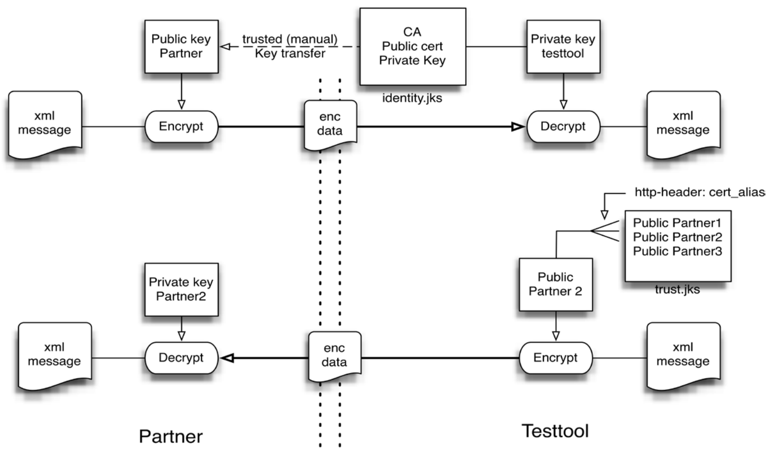


Figure 3: Message level encryption using PKI

If an AS2 message enters the ESB, it will be authenticated by checking the certificate against the CA, and then decrypted with the local (private key) certificate. If the ESB needs to send a message, it encrypts it with the public key of the partner and sends it on the wire. The Virtualize component used 3 http-headers to tell the ESB which certificates and endpoint to use.

For testing this process, we have used a separate AS2 client (Mendelson) that we can load with the certificates of the business partner so we can simulate and test the AS2 communication before bringing the application ‘live’. This helps in the troubleshooting of the correct loading of the certificates.

2.3 Virtualize

The simulation of the service that handles the communication between the partners is implemented using Parasoft Virtualize. The Virtualize toolkit was readily available and had all the functionality we needed for the backend. Next to that the customer had ample experience with other Parasoft tooling. In this setup Virtualize accepts XML messages on a Tomcat queue, checks the messages against the XML schema definitions and validates them. Based on the message content, Virtualize generates the response messages for a specific partner and sends it on to the ESB as http message. To let the ESB know which partners information to use in forwarding the message, Virtualize adds three custom http-headers:

`dest-endpoint`	AS2 endpoint where the ESB should deliver the message
`cert-alias`	The alias of the certificate in the keystore in the ESB that should be used for the encryption of the MIME payload
`as2-id`	The AS2 identifier of the business partner

Virtualize supplies the parameters to these headers based on the business partner identifier and the information stored in the backend database.

To be able to mimic asynchronous behavior, we need to be able to have a message that is generated in Virtualize not to be forwarded to the partner as described before we add the “ground” designator as value to `dest-endpoint`.

Using the scripting facilities of Virtualize we created python methods that are fired off when specific messages arrive. These scripts log the (validated) message and metadata in a database to be inspected on the contents, and respond though the ESB in a correct AS2 message to the client.

2.4 Database

In order to check the contents of the XML and its validity, the message is logged in a database. Next to the message itself, meta information on the business partner and timestamps are stored. This information than can be used to actually check the sequence and detailed contents of the XML message and report to the partner.

Besides storage of the incoming messages, the database also contains the different messages the tool should be able to send for certain test-scenario. This requires a trade-off between storing complete messages or only the relevant payload information and let Virtualize compose the correct XML message with right information. The database offers enough flexibility to support both approaches.

2.5 Database application

The application to validate the message transfer is a Java application that supports the process of validation of a partner, in which a number of test cases need to be performed. Each test case is a scenario in which a number (>= 1) of messages containing specific information need to be sent to or received from the test tool. The application can evaluate the database contents and report on the correct sequence and contents of the messages, and then flag a test (case) as succeeded or not.

Next to this functionality the database application implements a portal for maintaining clients, and querying the database for certain test scenarios executed by partners, or provide a partner with the ability to have the test system initiate a test in the direction of the business partner.

3. Experiences / Usage

The described project has a high business value, as partners are obliged to certify themselves before being allowed to do business on the Dutch energy market. Certification ensures the peer-to-peer communication, and thus the national protocol will work. As market introduction depends on certification of the partners the deadlines are pretty tight. The development team managed to build a version 1.0 of the system within a month. This can be regarded as a good accomplishment, also given the fact that the messaging specification on a functional (which test-scenario's to support) and technical level (final versions of the XML schema definitions) was still evolving during the development.

Problems encountered on the way were partly due to the different ways the business partners have set up their production systems and the PKI. During the process of testing we encountered problems in validation of the certificates as different partners had certificates signed by an older version of the root certificate, and thus were not able to be authenticated. The error messages from UltraESB did not point us in the right direction right away. This problem was solved in the checklist and standard operating procedure.

Other issues that we have seen were due to the fact that the system was required to run on a Windows platform that was present as a VPS in the cloud. The acceptance system and the development systems were set up for a single developer use so making parallel development more complicated.

Technical issues were seen in the management of the Windows firewall through the Java application, as we wanted to close the system off for all other IP numbers than those of the partners that registered for the validation. Managing the Windows firewall rules was difficult, so we implemented access lists on the Apache webserver as part of the ESB. Other difficulties were encountered in the file format of certificates and the multiple different keystores. Also the inner workings of some of the products used were not as expected, and needed adaption in the user code.

The next (2.0) phase of the project, currently on its way features the building of several web interfaces to facilitate the management of partners and the certification / validation process, in such a way that partners can start up their own validation process and see the progress of their certification.

4. Lessons Learned

A number of lessons can be learned from this project.

First of all, separation of the three main functionalities of Testtool, has been key to the success of this development. Using an of-the-shelf ESB to handle the (AS2) communication, the Virtualize engine to handle the XML payloads including technical validation of the message and the database application to support the actual validation process. This clear separation in architecture has proven to be important throughout the development process, although fine-tuning the interface between these three components required frequent detail adjustments.

Second, the modular design of Testtool, turned out to be very helpful to handle the changed specifications. Virtualize was programmed using a draft set of XSDs that in a later stage where replaced by the final versions. The core functionality depended only key (stabled) elements in the header of the XML message. Similarly, incomplete sets of the certificates influenced only the testing of the AS2 connectively not the overall development of the Testtool.

Another important issue in this project is that a solid PKI should be one of the requisites for starting. Designing a PKI and setting it up during a project that technically uses the PKI components yields to issues in communication.

An observation during the project resulted in the best practice of having infrastructure development to be ahead of software development. This will result in having a solid base to plan tests. In this project the infrastructure was designed and built alongside of the software environment, which resulted in some planning issues, especially with the tight deadlines we were facing.

Finally, the complete application could have been built from scratch, but as this was not an option within the project, readily available and re-usable “off-the-shelf” components were found in UltaESB and Virtualize. Even the custom build Java GUI will be re-usable for other certification environments that are

required in the future. The combination gives a flexible setup to handle changes in transport protocol, message formats and test-scenario’s.
The use of the UltraESB and Virtualize combination turned out to be the right choice for building this tool.

5. Conclusion

Without a validation system and the system driving it, it is not possible to have a large number of partners using a complex protocol securely and reliably amongst each other, a requirement in the Dutch deregulation of the energy market. The validation service has moved to full production successfully. Whenever a new partner wants to join the energy market the validation and certification of their backend communication systems has become a simple job. This system, quickly built out of a number of off the shelf components, provides a quick way to validate communications, and can be reused for other protocols in the near future.

6. Acknowledgements

The authors want to thank all people involved in the project. Next to the customer and their very helpful staff, Richard Jaspers of Competa IT and Georg Tornqvist of OELAN are thanked for their work in implementing the software needed. Thanks go to Dirk Giesen of Parasoft Netherlands for his guidance and expert advice.

[1] Ultra ESB:	http://www.adroitlogic.org/
[2] Virtualize:	http://www.parasoft.com/
[3] Mendelsson:	http://as2.mendelson-e-c.com/

Parasoft Service Virtualization Kit

voke Market Analysis

Parasoft White Paper

Service Virtualization Maturity Model

voke Research

SOLUTION SNAPSHOT™ REPORT:

Parasoft Virtualize

By Theresa Lanowitz | November 1, 2011

Moving markets beyond the status quo!

SOLUTION SNAPSHOT™ REPORT:

Parasoft Virtualize

By Theresa Lanowitz | November 1, 2011

SUMMARY

Adopting a lifecycle virtualization strategy benefits organizations by rapidly lowering capital and operational expenditures. Lifecycle virtualization enables development, QA, and operations teams to be more efficient and strategic by providing access to the correct environments, components, and services when and where needed.

The Parasoft Virtualize solution brings virtualization to the core of the lifecycle and delivers much needed accessibility and speed to developers, testers, and partners. Parasoft Virtualize delivers accessibility by removing constraints associated with limited access to systems, capacity, and unavailable or incomplete systems and components.

TABLE OF CONTENTS

EXECUTIVE OVERVIEW

Virtualization technology caused a revolution in the data center and was the technology of choice for every member of the C-suite as a result of its amazing effect of lowering capital expenditures (CAPEX). The value proposition in the data center was very clear to understand, and results were rapid.

Virtualization technology now stands to change other parts of the IT organization, specifically the development, quality assurance (QA), and operations teams. Lifecycle virtualization[1] is delivering value by removing constraints in the lifecycle and between teams, speeding time-to-market, lowering costs, and freeing teams from tactical activities.

By adopting a lifecycle virtualization strategy, organizations can rapidly lower CAPEX and operational expenditures (OPEX). Lifecycle virtualization enables development, QA, and operations teams to be more efficient and strategic by providing access to the correct environments, components, and services when and where needed. Lifecycle virtualization solutions reduce the costly wait time frequently associated with testing. With a more efficient and strategic approach, the lifecycle becomes more nimble, and teams can focus on delivering a faster time-to-market, with greater quality, while keeping costs under control.

This report features analysis of and commentary about Parasoft Virtualize. For more detail and analysis of the lifecycle virtualization market, see *voke Category Snapshot™ Report: Lifecycle Virtualization* – November 7, 2011.

MARKET CONTEXT

The lifecycle lends itself extremely well to the benefits of virtualization. Those benefits have expanded significantly from the first generation of lifecycle virtualization solutions that focused on virtual lab management (VLM) only, to the second generation and expansion to the cloud, to today: the third generation of lifecycle virtualization.

The third and current generation of lifecycle virtualization includes solutions for:

- VLM — virtualizing and delivering an environment as close to production as possible
- Virtualized cloud platforms — a logical extension of VLM, enabling organizations to rapidly and inexpensively create production-like cloud environments. As the move to public, private, and hybrid clouds becomes more pervasive, development and testing teams need on-demand access to cloud platforms prior to moving applications or services to production.

1 Lifecycle virtualization is defined as the use of technologies such as virtual lab management, service virtualization, defect virtualization, device virtualization, virtualized cloud platforms, et cetera, to enhance the application or product lifecycle through the reduction of defects, lowering costs, speeding time-to-market, and increasing customer satisfaction.

- Service virtualization — giving development and testing teams access to either unavailable or limited services in a virtualized environment. Virtualized services may be a database, mainframe, architecture, et cetera. Service virtualization also leads the way for virtualized or shared infrastructure.
- Defect virtualization — using virtualization technology to record an application as it executes and then replay it to identify the point of failure, empowering teams to eradicate even the most elusive defects.
- Device virtualization — using virtualization and simulation to allow physical devices to be virtually deployed for testing

While lifecycle virtualization has not penetrated the market as extensively as a disruptive technology should, the demands placed on all aspects of the business are requiring more efficiency at every phase, including the lifecycle.

voke is predicting the following for lifecycle virtualization:

- Lifecycle virtualization will be the hub of the modern lifecycle. As the hub, lifecycle virtualization will shatter silos across development, QA, and operations.
- Lifecycle virtualization will be a major disruptor in breaking bottlenecks in the lifecycle.
- Widespread adoption of lifecycle virtualization will occur across all industry sectors.
- Compliance issues will bring to the forefront the challenge of licensing in virtual environments. This will force software vendors to rethink licensing models to accommodate the exploding use and benefits of virtualized environments.

Lifecycle virtualization is an important and emerging category that will enable organizations of all types and sizes to reduce manual activities that are expensive and error-prone. Initial adoption of lifecycle virtualization will appeal to the logical need to lower CAPEX. However, as lifecycle virtualization becomes part of the lifecycle, organizations will experience tremendous strategic value through adoption and ongoing consistent use of the technology.

LIFECYCLE COMPLEXITY

Software and applications continue to grow in complexity. And while complexity increases at a rapid rate, development and QA teams do not grow in proportion.

Development and testing environments are frequently constrained due to budgets, scheduling, or lack of availability. Development and QA teams are too often delayed in completing critical activities because what the teams need is not available due to incompleteness or scheduling conflicts.

Because of delays associated with access to critical environments, services, systems, or other key assets, testing is frequently limited. This limited testing means environments and configurations are often incompletely tested or untested. Test coverage is impacted by these delays.

Service and System Availability

Testing of and access to environments such as mainframes, large ERP systems, and third-party systems is typically limited. Access is limited because of the unavailability of systems or components due to incompleteness or scheduling conflicts. Limited access results in testing delays or incomplete testing that add to schedule and cost. This is prevalent when business transactions involve third parties or partners operating mainframes or ERP systems. This usually creates scheduling conflicts in granting testers access to the required systems—as well as excessive access fees.

Performance

Performance testing is often incomplete, inadequate, or ignored due to the excessive cost of creating a production-equivalent environment. Limited capacity in development and testing environments hampers realistic performance testing. The effect on performance testing is further exacerbated by inaccessible or incomplete dependent systems.

Test Data

Proper testing requires production-equivalent test data. However, privacy concerns and compliance restrict the use of live data in test environments. Proper testing of data means QA teams need access to production-scale data and a production-equivalent environment for testing. Delivering the proper data structure and environment is a significant cost burden without the use of automated test tools and solutions.

Development and Test Labs

The need to build out costly test environments prohibits organizations of all sizes from delivering the best and most thoroughly tested software possible.

Development and QA teams both wait for the configuration and provisioning of environments from operations groups. Proper environments go well beyond just hardware and include operating systems as well as supporting software applications and databases.

Defect Reproduction

Developers and testers are frequently locked in conflict over the inability to reproduce a defect identified in testing or verify a fix in the proper environment. In many cases, defects are left unresolved and released to production or the field. Errors found in production or the field are far more costly. This inability to resolve a defect adds to cost and quality issues.

Unresolved defects remain in source code indefinitely and cause problems in subsequent releases of the software or application.

All of these complexity challenges are compounded when dealing with third parties such as systems integrators, professional service providers, or partners. In global and multi-dimensional organizations, development and QA teams need to be as productive as possible and meet schedule, cost, and quality goals.

⊙ MODERN SOLUTIONS

Lifecycle virtualization solutions such as Parasoft Virtualize are emerging in the market to solve classic problems associated with application/software development, testing, and delivery.

Access to appropriate environments, components, and systems is a classic development and test problem in need of a modern solution. Access constraints are removed through the use of lifecycle virtualization solutions, specifically solutions with service virtualization capability. Providing development and QA organizations access to what is needed at the required time and location keeps the teams productive and active.

Lifecycle virtualization is the hub of the modern lifecycle. Lifecycle virtualization will enable more fluid collaboration and communication between development and QA teams as they work to manage and tame increasingly complex software and applications. Removing unnecessary constraints and delays through lifecycle virtualization will facilitate a faster time-to-market, on budget, and with a high degree of quality.

Adopting a solution such as Parasoft Virtualize will help lifecycle stakeholders remove unnecessary constraints, improve access, and reduce the overhead and cost of managing and maintaining development and testing environments.

ABOUT PARASOFT VIRTUALIZE

Parasoft, founded in 1987, is an independent software vendor specializing in software development lifecycle automation. The company offers solutions in the categories of software development management, software quality lifecycle management, and development/test environment management.

Parasoft Virtualize, the company's lifecycle virtualization product, provides access to development and test environments by eliminating constraints inherent in complex, heterogeneous, and component-based applications.

Parasoft Virtualize is one of nine products offered by the company. Parasoft Virtualize is sold separately and integrates with popular industry testing and application lifecycle management (ALM) solutions including Parasoft Concerto and Parasoft Test.

⊙ PLATFORM SUPPORT

Parasoft Virtualize runs locally on the Windows and Linux platforms and supports the following technologies and protocols:

- HTTP/HTTPS
- ISO 8583
- JDBC
- JMS (WebShpere, webMethods, Sonic, TIBCO and others)
- JSON
- MQ
- MTOM (XOP)/MIME/DIME attachments
- .NET
- PoX
- REST
- SAML
- SOAP
- WSDL
- WS-*
- WS-Security
- XML
- XML Schema
- Fixed-length messages
- Custom

⊙ SOLUTION OVERVIEW

Parasoft Virtualize is a lifecycle virtualization solution that allows easy and fast access to any environment needed to develop, test, or validate software or an application—on-demand—at any point in the lifecycle.

Parasoft Virtualize's flexible architecture enables five core operations for lifecycle virtualization:

1. Define
2. Capture/model
3. Instruct
4. Provision
5. Utilize/consume

Developers or testers using Parasoft Virtualize define the components of the system under test that need to be virtualized for the purpose of development or testing.

When the developer or tester has access to a live instance of the component, the real behavior of the virtualized asset is captured by Parasoft Virtualize. If the system is unavailable or incomplete, the behavior of the intended virtualized asset is modeled. At this point in the process, a Parasoft Virtualized Asset (PVA) is created.

The PVA's behavior can be fine-tuned. For example, it is possible to adjust performance, data source usage, and conditional response criteria.

The PVA can then be provisioned or deployed within the desired infrastructure for testing purposes. The infrastructure may be a server, public cloud, private cloud, or hybrid cloud for simplified uniform access.

Once the PVA is provisioned or deployed to the desired infrastructure, developers, testers, or partners may consume and utilize the PVA to conduct testing on an as-needed basis. The PVA may be used in conjunction with all popular commercially available testing solutions. Constraints of system access are removed.

At the conclusion of the test cycle with PVAs, the environment may be provisioned with real assets so teams or partners may conduct a complete end-to-end test for final testing.

There are a variety of different use case scenarios for how Parasoft Virtualize may be used within the lifecycle. Regardless of use case, the flow of using Parasoft Virtualize remains the same. Developers, testers, or partners get access to the required environment for testing on-demand.

Common lifecycle use cases for Parasoft Virtualize include:

- Accessibility — providing testers with access to the services, environments, or systems needed to deliver thorough and complete testing
- Parallel development enablement — eliminating system dependencies and providing access to incomplete, unfinished, or evolving components for testing

- Test data management — simplifying the test data management process to simulate realistic production scenarios in testing
- Realistic performance tests — removing capacity constraints from testing environments to prevent corruption in testing
- Training — assisting training labs to provide clean data limits CAPEX, and reduces the risk of data corruption

SOLUTION ANALYSIS

Parasoft Virtualize represents a new breed of lifecycle virtualization solution that uses service virtualization to reduce the time required to configure applications for test, lower costs associated with hardware or system access, increase quality, and increase team productivity. Service virtualization used by Parasoft Virtualize removes constraints and allows developers and testers access to environments, components, or systems when and where needed.

The Parasoft Virtualize solution brings virtualization to the core of the lifecycle and delivers much needed accessibility and speed to developers, testers, and partners. Parasoft Virtualize delivers accessibility by removing constraints associated with limited access to systems, capacity, and unavailable or incomplete systems and components. Developers, testers, and partners work without delays and are able to meet time-to-market pressures.

Parasoft Virtualize uses the same mechanics in creating a PVA, regardless of the specific use case scenario that is exercised. This common workflow helps organizations with varying degrees of technical skills. Delivering a consistent workflow with an easy-to-use user interface removes some of the hesitation associated with the adoption of a new tool or solution.

Parasoft Virtualize integrates with popular testing solutions such as HP Quality Center, IBM Rational Quality Manager, Oracle ATS, and Parasoft Test. Integration with popular testing solutions allows testers to use the solution that is most familiar while conducting efficient and timely end-to-end functional and performance tests.

Parasoft is a vendor forging the adoption of lifecycle virtualization solutions. The market is in the emerging state and early phases of adoption. Parasoft must continue to deliver new and differentiated value in Parasoft Virtualize to remain competitive in what will most certainly become a very innovative and disruptive market.

Potential customers should be aware of the key strengths and weaknesses of Parasoft Virtualize during the evaluation phase.

Strengths:

- Ease of use
- Consistent workflow
- Integration with existing testing solutions
- Test data management component
- Performance testing in virtual environments
- Flexible and open architecture
- High customer satisfaction
- Price

Weaknesses:

- Lack of broad industry partnerships
- Third-party tools leveraged for highly complex test data scenarios

Because the lifecycle virtualization market is in the early stages, expect a variety of start-up vendors to emerge as the demand for lifecycle virtualization expands. Innovation in this market will accelerate as early adopters identify benefits associated with lifecycle virtualization solutions. Because of its ease of use and consistent workflow, we expect Parasoft Virtualize to be a strong contender in the lifecycle virtualization market.

SOLUTION HIGHLIGHTS

Parasoft Virtualize benefits developers and QA teams by enabling them to easily create an environment that realistically represents the constrained components that they need to access to complete their development or testing tasks. In addition to creating virtualized assets that emulate specific constrained components, it enables developers or QA teams to create an "instance" that represents a specific permutation or combination of real and virtualized assets. The notion of an "instance" and the management of complex transactions through both real and virtualized assets is critical for iterative testing.

Historically, development and QA teams are under pressure to rapidly complete their assigned tasks despite delayed or limited access to dependent components. Parasoft Virtualize enables teams to advance their development and testing efforts without having to wait for access to the dependent components. The reduction in wait time enables faster software delivery as well as more complete and thorough testing.

Parasoft Virtualize assists development teams that are trying to evolve interconnected components in parallel. By virtualizing the dependent components' expected behavior, each team can move forward without having to wait for the others to complete their work.

The solution aids QA teams in their quest to thoroughly test the designated application even when dependent system components are not yet ready or available for testing. Furthermore, test organizations spend excessive time waiting for test-ready code from development. Parasoft Virtualize reduces the wait time and risk associated with incomplete applications by allowing QA teams to model expected behavior. The reduction in wait time enables more complete and thorough testing of code and environments. QA teams are able to complete end-to-end testing without compromising schedule or costs.

As the adoption of Parasoft Virtualize grows, organizations with dedicated automation or performance centers of excellence (CoE) will find the addition of lifecycle virtualization essential. Adding lifecycle virtualization subject matter experts to an established CoE will help institutionalize the skill set across the organization.

⊙ EASE OF USE

Many organizations are using internally developed tools in an attempt to deliver lifecycle virtualization functionality. Internally developed tools are typically not scalable, lack in support, and become too expensive and difficult to maintain. Eventually, these teams will reach a point where an internal solution is no longer a viable or cost-effective solution.

Ease of use of a commercially available solution is often a requirement. Because development and QA teams frequently conduct evaluations of new technology on top of having to complete their regular work, evaluation teams are eager to try a solution with an easy-to-use user interface and a consistent workflow. This criterion is especially important for solutions that promise a quick ROI.

Parasoft Virtualize is differentiated in the market because of its ease of use and attractiveness to teams with varying degrees of technical prowess.

⊙ INTEGRATION

Parasoft Virtualize allows users to work without time, location, or infrastructure constraints. Part of the removal of constraints means flexibility and integration with other lifecycle virtualization solutions, cloud provisioning tools, and traditional application lifecycle management (ALM) solutions.

Virtualized assets created in Parasoft Virtualize may be called for in unit, functional, and performance tests and leveraged by popular test environments such as HP Quality Center, IBM Rational Quality Manager, Oracle ATS, and Parasoft Test.

Through this integration with existing testing solutions, Parasoft Virtualize enables an organization to preserve existing investments in both tooling and skills while delivering more value by speeding up the testing process and removing constraints. Parasoft Virtualize adds value to any development and test organization by complementing and integrating with existing automated or manual solutions, or working as a stand-alone solution to facilitate more complete and thorough testing.

⊙ DEFECT VIRTUALIZATION

Defect reproduction is an often elusive issue that developers, QA teams, and ultimately end-users must deal with. When test teams identify an issue, historically, reproduction by development has not always been guaranteed. Additionally, if reproduction is achieved by development, there is no guarantee that the defect remediation has not caused a cascading effect and produced another defect elsewhere in the source code.

Parasoft Virtualize creates a test environment as an "instance". The "instance" is attached to the defect. Everything travels and moves with the "instance" to ensure that the developer has everything associated with the defect. This complete environment means that when a defect is fixed, it is really fixed and does not inadvertently create another defect that might potentially go undetected. This also enables QA teams to verify the fix in the actual environment where the defect was found.

⊙ TEST DATA MANAGEMENT

One of the most frequently overlooked aspects of testing is test data management. Test data management is pivotal in reducing the time spent on test data preparation as well as reducing the risk of a compliance or security breach through inappropriate use of test or production data. Best practices in test data management call for test data to retain the proper data structure for testing while scrambling, encrypting, or masking the data for security purposes.

Parasoft Virtualize enables test teams to automate test data preparation and masking. Test teams have the ability to independently generate, scramble, mask, and refresh test data. This is a compelling and attractive feature of the product and the strategic value should not be overlooked.

⊙ PERFORMANCE TESTING

Performance testing is one of the most critical components in the lifecycle. Yet, obtaining accurate results is complicated by a number of obstacles. Performance testing is traditionally performed within the organization's own boundaries, making end-to-end performance testing involving third-party calls extremely challenging. When these third-party components can be accessed, such access typically incurs not only scheduling restrictions and access fees, but also bandwidth charges which can escalate rapidly for large-scale, high-throughput load or stress tests. Moreover, if the test environment does not match the production environment (e.g., in one case, the database is running on a virtual machine; in another, it is on a dedicated server), performance test results will inevitably be inaccurate.

Parasoft Virtualize enables performance testing through the use of service virtualization. Dependent application components can be virtualized and exercised in place of the actual components during performance testing. Deployed in this manner, the virtual assets can be accessed without scheduling issues and without incurring any fees.

The team can then analyze the virtualized assets' impact on the performance and behavior of the application under test. For example, if a business application relies on a database backend as a dependency, Parasoft Virtualize can emulate the database connection and the query execution times to assess their impact on the user experience at the application interface level. Performance is initially set to mimic the components' historical performance, and can be adjusted using graphical user interface (GUI) controls to simulate various "what if" scenarios.

Due to the rapid ascendance of the cloud, performance along with security are rapidly becoming the two most important pillars of testing. Parasoft Virtualize is innovating in lifecycle virtualization and performance testing.

⊙ SOLUTION SUMMARY

Parasoft Virtualize has the potential to play a significant role in the lifecycle virtualization market. With its ease of use, testing solution integration, defect virtualization with environment replication, test data management ability, and ability to remove performance testing roadblocks, Parasoft Virtualize is a solution that can be used by organizations of all types and sizes.

⊙ BENEFITS

Parasoft Virtualize helps organizations achieve the ultimate goal of staying on schedule and budget while producing a high degree of quality. Parasoft Virtualize is a solution created to benefit development and QA organizations.

Parasoft Virtualize:

- Accelerates time-to-market by reducing testing time.
- Reduces access fees to third-party environments or systems.
- Eliminates lab configuration time.
- Eliminates cascading defects as a result of incomplete environments for defect replication and remediation.
- Reduces delays experienced by testing teams working in parallel development environments.

- Removes constraints to environments, components, and services.
- Removes bottlenecks associated with environment set-up, maintenance, and management.
- Delivers predictability and consistency.
- Enables more thorough and complete testing.
- Enables faster time-to-market.

Developers and testers alike will benefit from a consistent and easy-to-use workflow to speed access to constrained systems, components, and services.

ASSESS YOUR ORGANIZATION

Ultimately, every organization will reach the conclusion that lifecycle virtualization is a technology that must be used to complement existing lifecycle solutions to help manage growing complexity, meet time-to-market pressures, and deliver a greater level of quality. Parasoft Virtualize is a solution that offers lifecycle virtualization benefits today.

Assess the readiness of your organization today to adopt a lifecycle virtualization solution.

- Does your organization struggle with the time and cost involved in setting up development or test environments?
- Does your organization experience delays because of unavailable or incomplete services, systems, or components during testing?
- Does your organization struggle with access to realistic performance testing environments?
- Does your organization need to reduce the access fees and third-party charges involved in functional and performance testing?
- Does your organization struggle with building and testing applications that interact with frequently changing components?
- Does your organization need to simplify its test data management strategy?

If you answered "yes" to any of these questions, your organization is ready at some level to embrace lifecycle virtualization. Once lifecycle virtualization is brought in to an organization, benefits will quickly be demonstrated and the desire to adopt it will spread rapidly.

JUSTIFY THE PURCHASE

Individual organizations may calculate projected ROI from a lifecycle virtualization solution such as Parasoft Virtualize by tracking any of the following suggested measurements.

Hardware Savings

- Number of physical servers replaced
- Number of physical machines replaced
- Storage and memory cost reduction
- Heating and cooling cost reduction

Lab and Environment Efficiency

- Time/headcount required to provision and deploy environments
- Time/headcount required to establish development and test labs

Productivity

- Time to deliver applications/software to production
- Number of platforms/environments tested
- Number of defects found/remediated prior to production
- Time associated with delays of system, component, service availability
- Parallel development time savings
- Improved productivity enabled by more strategic activities for developers and testers

Access Fee Savings

- Reduced time spent on shared infrastructure environments such as mainframes
- Reduced time for testing

Customer Satisfaction

- Fewer service calls associated with new releases
- Improved customer reaction to applications/products

Assess your organization's need for overall quality improvement by both development and QA and make a case for the adoption of lifecycle virtualization solutions to complement existing application lifecycle activities.

NET/NET

Parasoft Virtualize should be considered as either a complement to existing lifecycle solutions from a variety of vendors or as a stand-alone solution to act as the hub of all lifecycle activity.

Conduct a proof of concept or pilot with Parasoft Virtualize to determine how the solution fits within your organization. Once a part of lifecycle activities, Parasoft Virtualize may become part of an established center of excellence for either test automation or performance.

Contact Information

Corporate Headquarters

voke, inc.
2248 Meridian Boulevard
Suite H
Minden, NV 89423
USA

Phone: +1-866-895-9045
Web: www.vokeinc.com
Blog: www.voke.blogspot.com

ABOUT VOKE

voke, founded in 2006, is a modern independent technology analyst firm focused on the edge of innovation. voke's primary coverage area is the application lifecycle and its global transformation, including virtualization, cloud computing, embedded systems, mobile and device software.

voke provides data and analysis for the economy of innovation. Companies of all sizes, financial firms, and venture capital organizations turn to voke to harness strategic advice, independent and impartial market observations and analysis to move markets beyond the status quo. Please visit www.vokeinc.com to subscribe to voke research.

voke®
Moving markets beyond the status quo!

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.

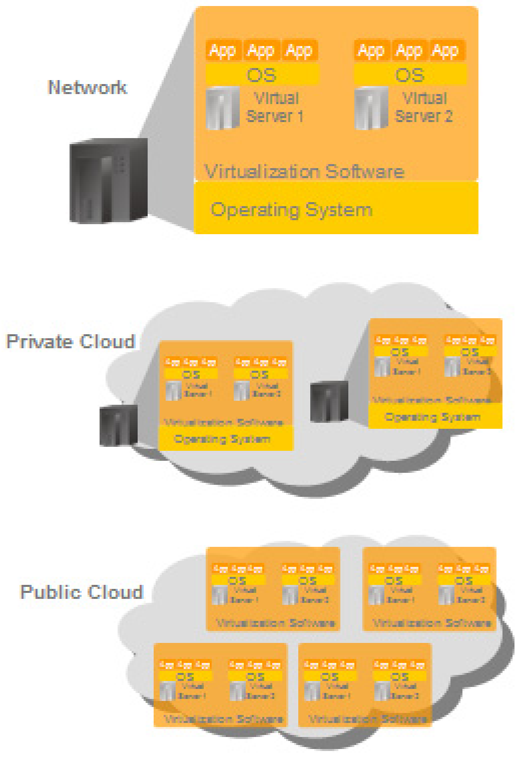
Furthermore, the number and range of people involved with software quality is rising. Advancements in development methodologies such as Agile are drawing more and more people into quality matters throughout the SDLC. For instance, Business Analysts are increasingly involved with user acceptance testing, 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.

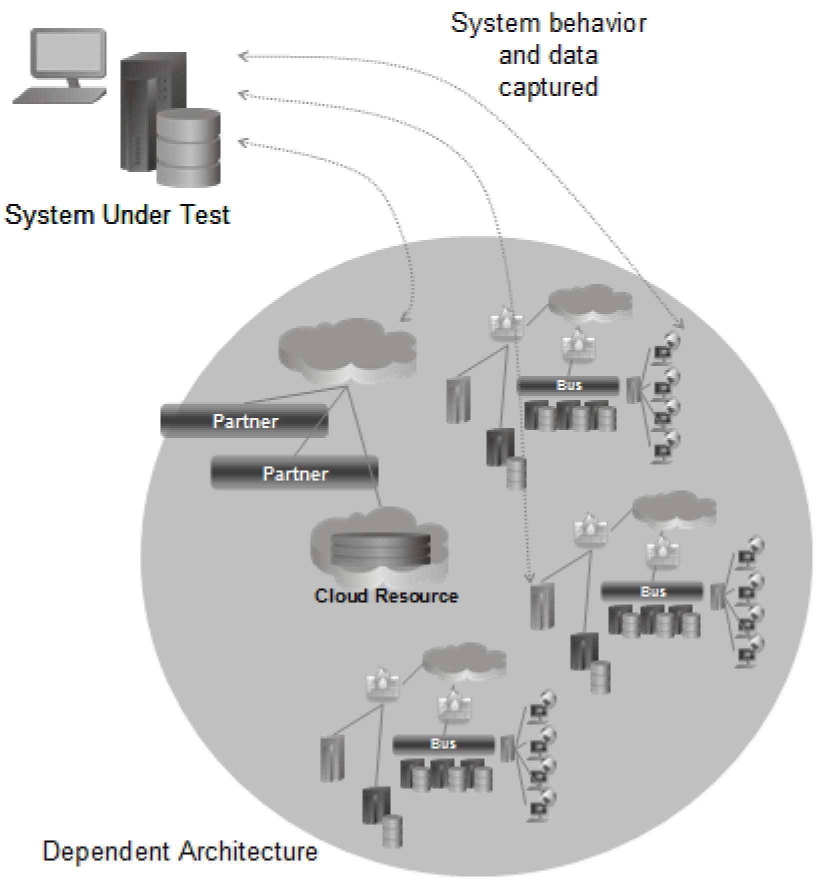
It is completely feasible to use the cloud for scalability with Service Virtualization. Nevertheless, since you're virtualizing only the specific behavior involved in dev/test transactions (not entire systems), the scope of what's being virtualized is diminished… and so is the need for significant incremental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

Service Virtualization Maturity Model

	Ad-Hoc	Reactive	Proactive	Managed	Optimized
Focus	Individual: One-off attempts to bridge gaps obstructing an individual's ability to complete a specific development or test task.	Project: Service Virtualization (SV) emulates dependent system components and allows the project's development or testing tasks to "shift left."	Environment: SV provides consistent access to dev/test environments that involve difficult-to-access, inconsistent, or unreliable system dependencies.	Scenario: Environments are coordinated to rapidly exercise different scenarios (performance, security, error conditions, etc.) in order to achieve better testing outcomes.	Enterprise: Provides optimized and secured environment access across and beyond the enterprise—including portals for business partners.
Characteristics	Dev/test scenarios need to execute across complex, dependency-rich environments, but access to a staged test environment is constrained. Developers react by creating stubs to pry the test or scenario out of the constrained environment. This is an ‚inside-out' approach. QA/performance test engineers react by waiting for access to a complex staged test environment (if available) or using stubs to bypass critical dependent systems.	A single group/project drives the creation and management of virtual assets that mimic behavior of incomplete or unavailable dependent components. Virtual assets are created for specific use cases and are augmented when needed for alternative cases. The extension of data sets or performance profiles is reactive based on specific testing needs.	A more holistic approach; accommodates a broader enterprise audience. SV is leveraged to provide continuous access to realistic dev/test environments (rather than simply alleviate project-specific access pains). Virtual assets are created, accessed, and managed in the context of environments. Policies, procedures, and standards exist around the application of SV. Consistent, continuous environment access enables more extensive and accurate testing to occur with or without access to a staged test environment.	Environments are governed by business rules that not only dictate what components are available, but also specify what permutations are valid under various contexts. Since business rules automate environment access and control, users can rapidly "self-provision" test environments. Configurations are accessed as 'disposable software' with zero risk. Lays the foundation for goal-oriented business-driven scenarios.	Provides the appropriate level of environment access to each constituency. A Center of Excellence is established to optimize and manage policies, procedures, and standards. Optimized environment for goal-oriented, business-driven scenarios significantly reduces application risk.

Service Virtualization Maturity Model

	Ad-Hoc	Reactive	Proactive	Managed	Optimized
Process Fit	Any pockets of maturity are based on the experience and initiative of individuals. No centralization of assets; every man for himself.	Enables earlier, easier testing, but does not necessarily diminish the need for staged test environments. A net new test environment is available by the use of SV; this is an initial step for facilitating Agile/parallel development.	Creates more sophisticated and flexible dev/test environments. Promotes a level of interconnectedness between SV and virtual test lab management systems.	Facilitates more mature coordination between SV and virtual test lab management systems.	Seamless integration and orchestration of SV with virtual test lab management systems. The unified solution establishes a single entity that allows for regression test suites to automatically call complex environments.
Environment Management	Assets are typically created as one-off solutions and stored on a local machine, inaccessible to anyone but the creator. The 'stub' is created without consideration of the environment and serves only the individual test.	Virtual assets might be evolved if needed to bridge project-specific gaps, but no overarching change management policies or processes exist.	Change is managed from the environment perspective. Users are notified of new virtual asset versions upon accessing the environment; change-impacts are highlighted, and users have the option of accessing the required version.	Robust change management and scenarios. Automated business rules drive the evolution of changing environment components.	The SV environment is governed by differentiated states associated with how various entities are accessing SV assets and environments.
Maturity Drivers	The application is not adequately tested or integrated due to limited access to environment conditions. The time and complexity of managing stubs outweighs the value they provide. Constrained access to staged test environments results in unacceptable test coverage or time-to-market delays.	Additional groups request access to the virtual assets with varied configuration options. Increasingly comprehensive scenarios vs. multiple dependent systems need to be tested. Project experience exposes the need for common, proactive processes for reuse and management.	Increased need to access multiple on-demand, "disposable" test environment configurations tailored for specific application or project demands.	Controlled access to sophisticated test environments is needed internally across the enterprise and externally with strategic business partners.	