

# Enabling Automated Integration Testing of Cloud Application Services in Virtualized Environments

Tariq M. King<sup>1</sup>, Annaji S. Ganti<sup>1 2</sup>, and David Froslic<sup>2</sup>

<sup>1</sup> Department of Computer Science  
North Dakota State University  
Fargo, ND 58103

<sup>2</sup> Microsoft Corporation  
Fargo, ND 58104

## Abstract

Software development under the cloud computing model brings the advantage that new applications can be rapidly constructed by tailoring existing services. However, the use of Internet-based services as software components, leads to the development of applications in which some building blocks are hosted remotely, rather than locally in a controlled environment. This aspect of cloud-based development, when coupled with factors such as service autonomy, complexity, and high dependability criteria, make software testing of the cloud especially challenging.

In this paper we present a novel approach to support integration testing of applications that depend on remotely-hosted cloud services. Our approach seeks to reuse elements of the test automation, typically built to validate a cloud service prior to its deployment, for the realization of Test Support-as-a-Service (TSaaS). TSaaS provides developers with a set of functions that enable integration testing of cloud services using controlled virtual environments. To facilitate continued evaluation of our approach, we have implemented a prototype of TSaaS that is compatible with the Windows Azure platform.

---

Copyright © 2011 Tariq M. King, Annaji S. Ganti, and David Froslic. Permission to copy is hereby granted provided the original copyright notice is reproduced in copies made.

## 1 Introduction

Cloud computing provides ubiquitous, on-demand access to shared computing resources “as services” over the Internet [13]. This emerging paradigm describes web-based technologies for realizing efficient, scalable, interoperable computing systems. Many cloud computing solutions are defined through Service-Oriented Architecture (SOA) [20], and hence facilitate reuse of loosely-coupled, composable software services. As with any paradigm shift, cloud computing promises many benefits to the IT industry, and brings new challenges to researchers and practitioners alike.

Software development under the cloud computing model brings the advantage that new applications can be rapidly constructed by tailoring existing services. However, the use of Internet-based services as software components, leads to the development of applications in which some building blocks are hosted remotely, rather than locally in a controlled environment [8, 23]. This aspect of cloud-based development, when coupled with factors such as service autonomy, complexity, and high dependability criteria, make software testing of the cloud especially challenging [4, 10, 26].

On the positive side, characteristics of the cloud infrastructure, such as high computational power, storage, and virtualization, lend

themselves to testing activities [21, 22]. This has led to a movement towards delivering testing as an on-line service, and much research on software testing in the cloud (STITC) has been directed towards this area [22].

To the best of our knowledge, the current state-of-the-art in STITC does not consider the need for test collaboration when an application incorporates services that are controlled by other stakeholders. However, such a collaborative test model is necessary to adequately validate these types of applications during development, and prevent cascading failures as services are updated or replaced [8, 20].

In this paper we seek to address some of the challenges associated with testing cloud-based applications. More specifically, we present a novel approach to support integration testing of applications that depend on remotely-hosted cloud services. Our approach provides developers with a set of functions that enable automated integration testing of cloud services using controlled virtual environments. The major contributions of this paper are as follows:

1. Presents a novel approach to integration testing of applications that depend on remotely-hosted cloud services. This includes a high-level architectural model and control flow description.
2. Elaborates on the design and implementation of a prototype, which was developed to facilitate the investigation of practical issues surrounding the research problem.
3. Discusses the lessons learned from building the prototype and conducting the research activity, and describes a plan for evaluating the proposed approach.

The rest of this paper is organized as follows: Section 2 provides background material. Section 3 motivates the research. Section 4 defines the problem under investigation. Section 5 describes our approach. Section 6 presents the prototype. Section 7 discusses our practical experience and outlines the evaluation plan. Section 8 is the related work and in Section 9 we conclude the paper.

## 2 Background

This section provides background material necessary for understanding the problem under investigation, and our proposed solution. It describes fundamental aspects of cloud computing, virtualization, and software testing.

### 2.1 Cloud Computing

Cloud computing provides ubiquitous, on-demand access to shared computing resources as services over the Internet [13]. Service models are categorized into Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [11, 13].

SaaS describes applications that are deployed in a cloud computing environment. Like conventional applications, SaaS is designed to help consumers perform a specific task, e.g., word processing. However, under the cloud computing model, applications are accessed through a thin client interface such as a web-browser, and are primarily managed and controlled by a host provider, e.g., Google Docs [7]. The terms cloud applications and SaaS are used interchangeably in this paper.

PaaS delivers programming languages and other software development tools to support the construction of SaaS. The provider allows the consumer to deploy, control, and manage cloud applications, and configure the environment in which they are hosted, e.g., Microsoft Windows Azure [18]. IaaS is the provision of processing, storage, and networking capabilities to the consumer for the purpose of hosting operating environments and applications in the cloud, e.g., Amazon EC2 [1].

In terms of deployment, cloud infrastructures may be categorized as [13]: Private – operated solely for a single organization; Community – shared by multiple organizations that have common interests or goals; Public – available to the general public or a large industrial group; and Hybrid – composed of two or more cloud deployment model types.

### 2.2 Virtualization

Virtualization is the creation of abstract computing resources from physical resources [2]. A

common example is server virtualization technology, in which one powerful physical machine is used to support multiple virtual machines (VMs) that can operate in parallel [2, 21]. In addition to hardware resources, it is also possible to manage virtual software resources such as operating systems and applications.

The component responsible for managing virtual resources in a virtualization system is called a *hypervisor* [2]. Hypervisor-based virtualization tends to have better performance and security than hosted virtualization, where the VM executes as a program inside the host operating system [21]. Due to the trend towards virtualization, modern operating systems, APIs, cloud platforms and infrastructures, are providing built-in support for managing virtual computing resources [1, 18, 19].

### 2.3 Software Testing

Software testing involves running software under specific conditions, recording the results, and making an evaluation of the system [9].

The approach to software testing can be based on the program specification, its implementation, or both [3, 27]. During specification-based testing, the requirements guide the process of test case selection, and provide a means of evaluating the adequacy of testing. Alternatively, implementation-based testing focuses on exercising the program under test, and hence adequacy is typically stated in terms of coverage of elements of the program structure [27], e.g., all statements, all branches.

Testing can be performed at three levels of granularity [3]: Unit Testing – validates the functionality of a component in isolation; Integration Testing – validates the interactions among multiple components; and System Testing – validates the entire piece of software to ensure functional and non-functional requirements are being met.

During the unit and integration testing, stubs and drivers are created to be used as scaffolding for executing tests. A test stub is a mock version of a component, which simulates the actual component for the purpose of testing [3, 5]. On the other hand, a test driver is a utility program that executes test cases on the system under test. The test drivers and all

other tools to support test execution are collectively referred to as the test harness [5].

The operations of an automated test fixture can be divided into the following categories [14]: (1) Setup/Initialize – establishes the pre-conditions for the test; (2) Input – stimulates the system under test with specific values; (3) Assertion – checks that whether actual results are in agreement with the expected results; and (4) Teardown/Cleanup – resets the state of the system under test and its environment after the test has been executed.

## 3 Motivation

Software development under cloud computing and service-oriented architectures has the advantage that new applications can be rapidly developed by tailoring existing services, while hiding the complexity of the underlying implementation. However, these and other characteristics of the cloud computing paradigm present significant challenges with respect to software testing [6, 8, 10, 23].

Internet-based access of software services as reusable components, means that some of the building blocks of cloud applications may be hosted remotely rather than locally in a controlled environment [8, 23]. Controllability and observability of remotely hosted services are limited to what has been exposed via their public interfaces, which may be highly restricted due to security and privacy concerns. Ownership of services by various providers implies that successful integration testing will require coordination of these stakeholders [8].

Cloud applications are hosted on intrinsically complex computing infrastructures, with multiple layers extending from the underlying network to the top-level client interface. Testing must check the behavior of each layer, the interactions among multiple layers, and various aspects of the fully integrated system in different deployment configurations [23].

Application service development must address high dependability criteria, which includes security, availability, robustness, fault tolerance, among others [20, 26]. Since information is communicated over the Internet, security testing is critical to ensure data confi-

dentality and integrity are maintained.

High availability requirements means that there is little downtime for testing modifications to services, which can lead to rippling effects throughout applications as services are updated or replaced by new ones [8]. In addition, dynamic software adaptation and integration in the cloud may require testing to be performed at runtime, possibly within the context of the production environment [8, 10].

In summary, cloud computing presents several new challenges with respect to software testing, and these challenges are the primary motivation for this work. On the positive side, characteristics of the cloud infrastructure, such as high computational power, storage, and virtualization, lend themselves to software testing activities. The potential benefit of using the cloud infrastructure itself to tackle the challenges of testing cloud applications, represents additional motivation for this research.

## 4 Research Problem

The research problem under investigation focuses on enabling automated integration testing in cloud computing environments.

As shown in Figure 1, we narrow the scope of the research area by analyzing a scenario in which one host (Provider B) is developing an application that will extend the functionalities of an existing service being offered on a remote host (Provider A). In addition, we assume that Service A has no inter-provider dependencies, and hence Provider A has full control over the components used to implement Service A.

As described in the problem motivation, this type of development model greatly reduces the testability of cloud applications like Service B, due to limited control over remotely-hosted services such as Service A. However, software testing theory and practice indicates that, although Service A may have been tested in isolation, it must still be tested in the context of Service B since faults can arise due to their interaction [25].

A possible solution to integration testing under the given scenario is for Provider B to develop a test stub of Service A. However, this also poses a significant challenge because

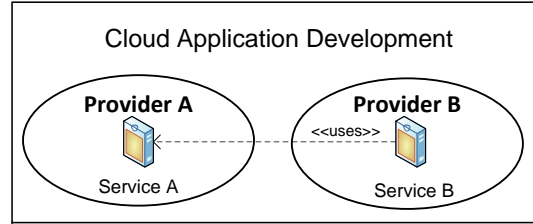


Figure 1: An Inter-Cloud Service Dependency

Provider B lacks knowledge of Service A’s implementation. Even with the required implementation knowledge, creating accurate test stubs of Service A may still prove to be difficult for Provider B if the service is characterized by a highly complex underlying infrastructure.

To guide the investigation, we have formulated the following research questions: (1) how can any pre-existing test artifacts and tooling, which may have been used to validate Service A prior to its deployment, be harnessed to provide the developers of Service B with automated integration test support; and (2) how can hardware and software virtualization techniques be employed by Provider A at runtime, to enhance the process of delivering automated integration test support to Provider B.

## 5 Approach

To address the research problem, we describe a novel approach that enables integration testing in cloud computing environments. Our approach is based on the notion of *Test Support-as-a-Service* (TSaaS), which was introduced by King et al. [10] in a high-level position on testing autonomic cloud applications.

In this section we provide the first formalized description of TSaaS, including a definition of its architectural components and a control flow diagram to support its implementation.

### 5.1 Overview

The idea behind TSaaS is that, prior to the deployment of Service A in its production environment, Provider A would have validated the service to check for accurate functionality, interoperability, performance and other quality attributes. Due to the inherent drawbacks of

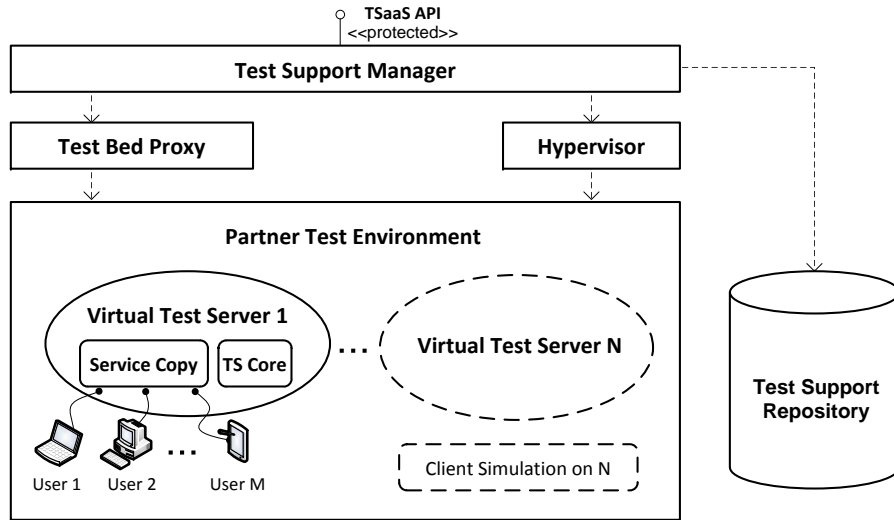


Figure 2: A Hypervisor-Based Architecture to Support Integration Testing of the Cloud

manual testing, Provider A is likely to have employed automated software testing techniques during validation. If so, a pre-configured local test environment, test scripts, and scaffolding to facilitate automatic test execution would be available on the infrastructure of Provider A.

Our approach seeks to reuse elements of Service A’s existing test automation to rapidly realize a set of test support services for Provider B, and other collaborating cloud providers. To avoid unnecessary security risks when applying the approach, we propose that TSaaS be deployed under a community cloud model [13]. In other words, TSaaS should be delivered via a protected interface that is only accessible to trusted collaborating providers, referred to as *cloud partners* in this paper.

TSaaS would enable integration testing by allowing partners to observe and control the private members of a copy of Service A. Privacy of the data accessible via TSaaS would not be a concern because only non-confidential test data would be made available. Furthermore, using a separate copy of the service for testing does not interrupt the regular operations of the actual service hosted in production. This ensures that the approach is applicable to services that have high availability requirements.

Ideally, the TSaaS implementation would need to be powerful enough to facilitate the

concurrent execution of test support requests from multiple partners, each having a separate handle to a copy of the service under test and its test environment. This would lead to increased production overhead due to the need to setup and manage these additional computing resources. Therefore, to reduce the overhead of the proposed approach, we incorporate virtualization techniques into TSaaS.

## 5.2 Architecture

Figure 2 defines an architectural model for TSaaS, showing its major components and their dependency relationships. As shown at the top of Figure 2, a test support API is exposed to cloud partners via a protected interface. The architectural components that realize the operations defined in the TSaaS API are described as follows:

- *Test Support Manager* – A controller component that coordinates underlying components to deliver the test support services defined in the TSaaS API. This component is also responsible for authenticating partners, and enforcing authorization policies.
- *Hypervisor* – A virtual machine monitor for creating, managing and destroying the virtual test servers where TSaaS requests will be realized.

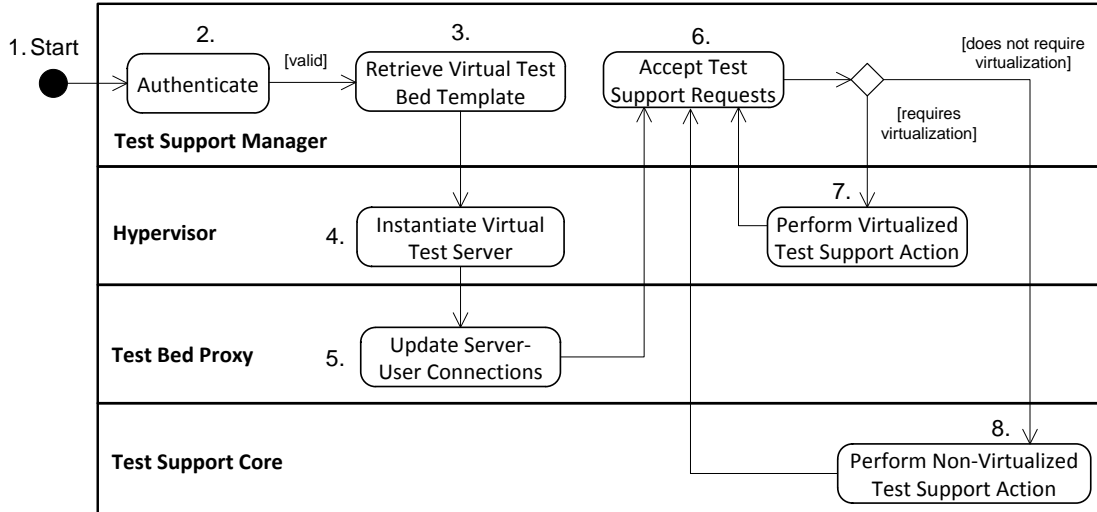


Figure 3: Activity Diagram showing the Control Flow of TSaaS Components

- *Test Bed Proxy* – A component which interfaces between the Test Support Manager and the virtual test servers when servicing user requests. This component allows the virtual test servers to remain anonymous for security purposes, and facilitates logging usage statistics.
- *Partner Test Environment* – A hardware and software platform for hosting the virtual test servers. Each virtual server instantiated on this platform will consist of: (1) a hosted copy of the service under test, and any required dependencies; and (2) the Test Support Core component.
- *Test Support Core* – A service implementation of automated test fixture operations for the service under test. This requires that all test-related dependencies be pre-installed on the virtual test servers.
- *Test Support Repository* – A data store containing artifacts that are used and generated by TSaaS, including virtual hard disk images, snapshots, test datasets, test results, server logs, configuration files, etc.

### 5.3 Control Flow

Under TSaaS, partner requests for test support can range from simple actions such as invoking

the service under test using specified inputs, to more complex actions like simulating clients and user loads via virtualization.

Figure 3 presents an activity diagram that describes how the architectural components of TSaaS collaborate to realize test support requests. Horizontal swimlanes have been used to denote which components perform each action shown in the diagram. The control flow description starts when a partner connects to TSaaS via its protected interface. The steps in the control flow of TSaaS are as follows:

1. A partner provider connects to TSaaS, and logs in using predefined credentials.
2. The Test Support Manager (TSM) authenticates the partner by checking the credentials entered against the known set of valid users.
3. If the user is valid, the TSM retrieves an image of the partner’s virtual test bed from the Test Support Repository (TSR).
4. The TSM passes the virtual test bed image to the hypervisor, which instantiates a virtual test server for the partner.
5. A handle to the address of the virtual test server is returned to the Test Bed Proxy

(TBP), which updates its list of server-user connections.

6. The TBP notifies the TSM that the new virtual test server is ready for use, and the TSM starts accepting test support requests from the user.
7. If a test support request requires virtualization (e.g., loading a snapshot, resetting the server, simulating loads), the TSM delegates the request to the hypervisor.
8. Otherwise, the TSM forwards the request to the Test Support Core, which performs the desired actions on the service copy and/or its operating environment.

## 6 Prototype

To investigate the feasibility of the proposed approach, we developed a prototype of TSaaS. In this section we describe the application service used in the prototype, and provide details on our setup environment and implementation.

### 6.1 Application Description

The application service used in the prototype is based on consumer credit reporting functionalities. Our service, referred to as the Credit Reporting Service (CRS), is summarized by the following features:

- *Payment History* – determines if there are any negative items associated with a consumer’s payment history.
- *Debt* – computes the total amount and type of debt owed by a consumer.
- *Age* – calculates the age of a consumer’s oldest credit account, and the average age of all accounts in his/her credit report.
- *Diversity* – establishes whether or not the consumer has multiple account types.
- *Inquiries* – checks for recent inquiries on a consumer’s credit report.

The central role of services such as CRS when making financial decisions can result in lending

agencies building their own applications around them. For example, a bank may use CRS in an initial step of the business process for an on-line loan service. Therefore, in relation to the research problem, CRS represents our baseline service (Service A) that is being extended to create another derivative application such as the banking loan service (Service B).

### 6.2 Setup Environment

A survey performed as part of our preliminary investigation led us to select Microsoft Windows Server 2008 R2 as the operating environment for the prototype [10, 19].

The primary reason for choosing this platform is that it uses the same tools and technologies upon which the Windows Azure cloud computing platform is built [18]. These include hypervisor-based server virtualization technology through the Hyper-V server role, and support for runtime virtualization via calls to the Hyper-V Windows Management Instrumentation (WMI) APIs [17].

Using these tools and technologies allowed TSaaS development to begin immediately within a local environment, with the goal of building an implementation that could be easily migrated to the Windows Azure cloud computing platform.

### 6.3 Implementation

Our description of the prototype implementation elaborates on the service under test, automated test fixture, test support core, virtual test servers, and test support manager.

#### 6.3.1 Service Under Test

Using Visual Studio 2010, we developed CRS as a Windows Communication Foundation (WCF) service so that it could be deployed in the Windows Azure web role [15]. The following operations, corresponding to a subset of the features listed in Subsection 6.1, were implemented for CRS: *CalcHistory*, *CalcDebt*, *CalcAvgCreditAge*, and *CalcInquiries*.

Access to a representative set of consumer credit report information was achieved through the use of a SQL Server 2008 R2 database. CRS interacts with the database using the

.NET 4.0 entity frameworks. In addition, WCF diagnostic tracing capabilities were enabled on the service to facilitate logging runtime calls, warnings, exceptions, and other events.

### 6.3.2 Automated Test Fixture

Automated tests for CRS were developed using the unit testing framework which is integrated into Visual Studio. To allow the automated tests to be customizable and extensible for delivering TSaaS, we applied an XML-based test case specification technique to parameterize data values which appeared in the fixture.

Figure 4 provides a snippet of the test case specification XML (.tsx) of CRS. The specification is divided into the three tagged sections `ClassInitialize`, `TestCase`, and `ClassCleanup`, representing the test setup, procedure, and teardown aspects of the automated test fixture respectively. For example, each `TestCase` element consists of tagged subsections defining the parameterized inputs and expected output values of the test.

Using the Linq-to-XML framework, the specified test structure is loaded into a `TestCase` object as part of the test initialization process. A helper method then retrieves the specific test input and expected output values, and creates the instance of the test case. This process is repeated until a complete set of tests for CRS has been built, and then packaged into a dynamic-link library called `CRSTests.dll`.

The automated tests stored in the `CRSTests.dll` assembly can be run programmatically using the command line tool `MSTest`. Test execution produces a test results XML (.trx) log file indicating whether each test passed or failed, along with any error messages associated with test case failures.

The CRS service was instrumented for code coverage through the use of a configuration file named `Local.testsettings`. This configuration file and `CRSTests.dll` can be passed as arguments to the `MSTest` tool at runtime to execute the tests with code coverage enabled. Such a call would produce a similar log of test results to the previously mentioned call, but would also facilitate extraction of the code coverage results of test runs.

```
<?xml version="1.0" ?>
<Tests>
  <ClassInitialize>
    <DB>TestDBConnection</DB>
  </ClassInitialize>
  <TestCase Name="CalculateDebtTest">
    <Inputs>
      <SSN>123-45-678</SSN>
    </Inputs>
    <ExpectedOutputs>
      <Value>1024.50</Value>
    </ExpectedOutputs>
  </TestCase>
  <ClassCleanup />
</Tests>
```

Figure 4: Snippet of Test Specification XML

### 6.3.3 Test Support Core

The parameterized test fixture implementation, consisting of the `CRSTests.dll`, test specification XML, and `MSTest` tool, was reused to develop a set of fundamental test support operations for CRS. These test support operations were themselves encapsulated and deployed as a WCF service. Table 1 lists the operations that were implemented within this core test support service, and describes their purpose. The operations are divided into the categories: (1) Test Specification and Execution, (2) Test Configuration, and (3) Test Reporting.

### 6.3.4 Virtual Test Servers

Hypervisor-based server virtualization was achieved via the Hyper-V role of Windows Server 2008 R2. We prepared a virtual hard disk (VHD) image of the CRS test environment, which allowed virtual test beds for partner providers to be instantiated on-demand. The VHD specified a uniform hardware configuration for each virtual test server, and was loaded with all of the software components necessary for replicating a controlled test environment for CRS.

Figure 5 provides an overview of the software configuration of the VHD image used for

Test Specification and Execution	
Operation	Description
<i>UpdateTestSpec</i>	Facilitates editing and overwriting the set of test cases contained in the test specification XML.
<i>RunTests(in TestName)</i>	Executes a specific test case, whose name is denoted by the input parameter <i>TestName</i> , in the test specification.
<i>RunAllTests</i>	Executes all test cases defined in the test specification.
<i>RunAllTestsWithCoverage</i>	Executes all test cases defined in the test specification with code coverage enabled.
Test Configuration	
Operation	Description
<i>UpdateTestConfig</i>	Facilitates editing and overwriting a server-side settings file that is used to configure the test harness ( <i>Local.testsettings</i> ).
<i>UpdateServiceConfig</i>	Facilitates editing and overwriting a server-side settings file that is used to configure the service under test ( <i>Web.config</i> ).
<i>ResetWebServer</i>	Performs a reset action on the Web server to use updated configuration settings, or to reinitialize the server after failures.
Test Reporting	
Operation	Description
<i>GetLatestTestResults</i>	Facilitates retrieving the test results XML produced from the most recent test run performed on the service under test.

Table 1: Summary of Operations in the Test Support Core of the TSaaS Prototype

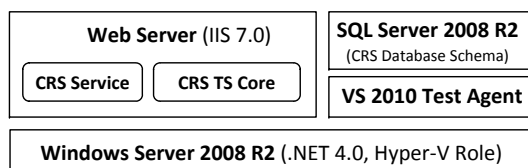


Figure 5: Software Configuration of Virtual Hard Disk Image used for Virtual Test Servers

the virtual test servers. As shown in Figure 5, the image consisted of the following software components: (1) Windows Server 2008 R2 with the .NET 4.0 runtime installed; (2) SQL Server 2008 R2 with the CRS data schema applied; (3) Internet Information Services 7.0 Web Server with CRS and the Test Support Core services hosted; and (4) Visual Studio 2010 Test Agent, which includes the MSTest tool.

### 6.3.5 Test Support Manager

Our implementation of the test support manager (TSM) invoked the Hyper-V Windows Management Instrumentation (WMI) APIs to create, configure, and destroy the virtual test servers at runtime. The broker design pattern was used to structure the TSaaS components, and allow the TSM to call the methods exposed by the test support core on the respective virtual test servers. Components to support user-load simulation have been left for future work.

## 7 Discussion

The primary reason for developing the prototype was to investigate the feasibility of the proposed approach, and discover practical issues surrounding its implementation and usage. In this section we elaborate on our practical experiences when developing and applying

TSaaS, including the major successes and challenges identified during the project. An evaluation plan for further investigation into the research problem is also discussed.

## 7.1 Successes

The CRS TSaaS prototype provided us with a framework within which we could build and adequately test extensions and derivatives of the remotely hosted service.

As proof-of-concept we developed and performed integration testing on a banking loan service that incorporates CRS in its business process, and a customized version of CRS that overrides part of its functionality. The prototype facilitated quick and easy access to a cross section of automated testing operations (Table 1), preexisting test data, and a controlled test environment for CRS during integration. In addition, having access to server-side logging and trace information through TSaaS was extremely useful for testing and debugging.

Successful implementation of the prototype suggests that the approach and research ideas presented in this paper are feasible. In particular, our prototype implementation addresses the specific research problem described in Section 4, and validates the TSaaS architectural model. A major factor that contributed to the success of the project was the use of the XML-based test case specification language, which allowed us to decouple the test case definitions from the test fixture implementation for direct reuse in the TSaaS prototype.

Another contributing factor was having built-in virtualization support via the Hyper-V role of Windows Server 2008 R2, and the Hyper-V WMI APIs. This made it possible for us to expose remote procedures for instantiating and destroying virtual test servers, and configuring their virtual hardware resources. Furthermore, these technologies lay the foundation for experimentation on the Windows Azure cloud computing platform as is described in our evaluation plan.

## 7.2 Challenges

Some technical challenges were experienced during prototype development, most of which

surrounded the need to integrate multiple tools and technologies. Implementing TSaaS required a good combination of programming experience, test automation skills, virtualization expertise, and access to proper frameworks and tools to support the approach and development process. However, at this time of writing cloud computing is an emerging technological model, with limited tool support.

Configuring the baseline VHD required attention to technical details in order to ensure the test environment was secure and stable. This included writing auto-login scripts that provide the user credentials to the VMs at startup, and applying all security patches and other updates to the virtual machine. We also ran into the issue where it was possible for a VM to restart if automatic updates were enabled, and therefore had to adjust this setting to prevent test support sessions from being interrupted unexpectedly.

Our investigation also revealed the need for practical ways to keep the VHDs up-to-date with the latest versions of the service, its environmental configuration, and test support tools. To avoid inconsistencies due to human error, such maintenance should be automated through VM tools and update scripts. The relatively large size of VHDs also means that manipulating these images, and re-deploying them to the cloud environment, may also present a significant challenge during maintenance.

A grand challenge to be addressed is the provision of standardized testing interfaces and guidelines for building testing and test support implementations for cloud infrastructures. This type of research direction is necessary for the successful adoption of the approach, and can provide benefits to the general movement towards software testing as an online service. Standardization could help to overcome many of the inherent challenges of the approach such as scalability, security, among others.

## 7.3 Evaluation Plan

The current prototype and investigation has been purposely limited in scope to manage the high complexity of the research problem. As such, much work still need to be done to evaluate the proposed approach to integration test-

ing of cloud-based applications. In particular, conducting experiments on a cloud computing platform is necessary to reveal the essential difficulties associated with the research problem. With this objective in mind we have carefully designed the prototype in a way that streamlines the process of moving it to the cloud.

The Windows Server 2008 R2 image developed for the prototype is directly supported by Windows Azure through its VM role [15, 16]. The VM role provides a high degree of control over the server instances, and facilitates many of the goals required for our approach to be successful in practice (i.e., scalability, in-place upgrades, integration with other service components, and load-balanced traffic [16]).

We have identified that the following steps are necessary for migration to the Windows Azure cloud platform: (1) install the Windows Azure SDK and its prerequisites on the development environment; (2) use the Windows Azure SDK to configure the baseline VHD and then upload it to a Windows Azure storage drive in the cloud, and (3) move the SQL Server 2008 R2 database to SQL Azure.

With the prototype hosted and deployed in Windows Azure, we plan to conduct a series of controlled experiments to determine the effectiveness of the proposed approach. The experiments will seek to: (1) provide evidence that TSaaS can be used to overcome challenges associated with cloud application testability, and (2) demonstrate the benefits of harnessing virtualization technologies in the cloud for testing cloud application services. Lastly, we plan to investigate the industrial applicability of TSaaS, and compare it to approaches such as test-isolation and testing in production.

## 8 Related Work

The rapid adoption of cloud computing technologies and popularity of SOA has generated much research interest. As such, researchers have begun to address some of the challenges of testing these types of applications. In this section we provide a literature review of works that are related to our research.

Chan et al. [4] use graph theory to represent the computations of cloud applications,

and define model-based criteria to support testing them. Part of their model describes interactions among cloud computations, and defines the notion of inconsistency detection between nodes. In addition, they demonstrate how their cloud graph model can be used for dynamic composition of adaptive clouds. Their definitions and testing criteria help to formalize problems associated with cloud integration, and is highly complementary to our work.

Zech et al. [26] propose a model-driven approach to address security testing of cloud applications. Their approach derives a set of misuse cases through risk analysis, and then uses them to generate tests that attack the application environment. Although our work on TSaaS does not address security testing, this is a critical aspect of testing cloud applications and should be considered at an early stage in the development process.

Greiler et al. [8] describe industry challenges for runtime integration and testing for SOA. The challenges identified have motivated our work, and include factors such as stakeholder separation, service integration, dynamic service updating, binding and reconfiguration. The work also discusses some directions for addressing the challenges from both methodical and technical perspectives. Our approach specifically addresses cloud computing, which provides enabling technologies for solutions that may be defined through SOA. Furthermore, we use those same technologies to tackle some of the challenges of testing cloud applications.

Several works on testing and testability of SOA applications can also be found in the literature [12, 20, 24]. Tsai et al. [24] propose several evaluation criteria for SOA software, and apply those criteria to a case study where the application domain is stock-trading. O'Brien et al. [20] describe quality attributes for SOA, and discuss its testability concerns. Mei et al. [12] describe an approach to data flow testing of SOA applications, specifically applied to WS-BPEL software products.

## 9 Conclusion

This paper presented a collaborative approach to integration testing of cloud-based applica-

tions using virtualized environments, referred to as Test Support as-a-Service (TSaaS). To demonstrate the feasibility of our research ideas, we implemented a prototype of TSaaS using technologies compatible with the Windows Azure cloud computing platform.

Although our work was limited in scope, it provided much insight into the research problem and laid the foundation for the establishment of an evaluation plan. A major milestone defined for the success of this research is the application of TSaaS to an industrial case study. Future work also calls for the incorporation of performance-based testing techniques, and user load simulation features into TSaaS.

## Acknowledgements

The authors would like to thank the participants of the 2010 Workshop on Software Testing in the Cloud, and the 2011 Microsoft Fargo Engineering Day Excellence Expo for their valuable feedback on this work. Annaji would also like to express his gratitude to the Microsoft tuition reimbursement program that supports his graduate studies.

This work was supported in part by the National Institutes of Health (NIH) under grant 2R44RR024779-02A1. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of the NIH or the Microsoft Corporation.

## About the Authors

Tariq M. King is an Assistant Professor in the Department of Computer Science Department at North Dakota State University. His areas of interest are software testing, autonomic and cloud computing, model-driven development, and computer science education. His e-mail address is tariq.king@ndsu.edu.

Annaji S. Ganti is a PhD Student in the Department of Computer Science at North Dakota State University. He also works at Microsoft as a Software Design Engineer in Test. His areas of interest are software testing and cloud computing. His e-mail address is annaji.ganti@ndsu.edu.

David Froslic is a Software Test Architect working on the Dynamics Enterprise Resource Planning product line for Microsoft. One of his primary areas of interest is test infrastructure for automated software testing. His e-mail address is dfroslic@microsoft.com.

## References

- [1] Amazon.com. Amazon Elastic Computing Cloud (Amazon EC2), June 2011. <http://aws.amazon.com/ec2/>.
- [2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the 19th ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.
- [3] Boris Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, New York, second edition, 1990.
- [4] W.K. Chan, Lijun Mei, and Zhenyu Zhang. Modeling and testing of cloud applications. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 111–118, Dec. 2009.
- [5] Jean-Francois Collard and Ilene Burnstein. *Practical Software Testing*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.
- [6] Colleen Frye. Cloud Computing Creates Software Testing Challenges, Nov. 2008. <http://searchcloudcomputing.techtarget.com/news/1355198/Cloud-computing-creates-software-testing-challenges> (June 2011).
- [7] Google Inc. Google Docs: Create and Share Your Work Online, Feb. 2007. <http://docs.google.com> (June 2011).
- [8] M. Greiler, H.-G. Gross, and K.A. Nasr. Runtime Integration and Testing for Highly Dynamic Service Oriented ICT Solutions – An Industry Challenges Report. In *Testing: Academic and Industrial*

- Conference - Practice and Research Techniques, 2009.*, pages 51–55, Sept. 2009.
- [9] IEEE Computer Society. Std 610.12-1990(r2002): Glossary of software engineering terms. Technical report, 2002.
- [10] Tariq M. King and Annaji S. Ganti. Migrating autonomic self-testing to the cloud. In *ICSTW '10: Proceedings of 3rd IEEE International Conference on Software Testing, Verification and Validation Workshops*, pages 438–443, April 2010.
- [11] The RAD Lab. Above the clouds: A Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.
- [12] Lijun Mei, W.K. Chan, and T.H. Tse. Data flow testing of service-oriented workflow applications. In *Proceedings of the 30th international conference on Software engineering, ICSE '08*, pages 371–380, New York, NY, USA, 2008. ACM.
- [13] Peter Mell and Tim Grance. Draft: The nist definition of cloud computing. Technical Report SP 800-145, National Institute of Standards and Technology, Jan 2011.
- [14] Gerard Meszaros. *XUnit Test Patterns: Refactoring Test Code*. Addison-Wesley, 2007.
- [15] Microsoft Corporation. Overview of Creating a Hosted Service for Windows Azure, April 2011. <http://msdn.microsoft.com/en-us/library/gg432976.aspx>.
- [16] Microsoft Corporation. Overview of the Windows Azure VM Role, March 2011. <http://msdn.microsoft.com/en-us/library/gg433107.aspx>.
- [17] Microsoft Corporation. The Hyper-V WMI Provider, May 2011. [http://msdn.microsoft.com/en-us/library/cc136766\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/cc136766(v=VS.85).aspx) (June 2011).
- [18] Microsoft Corporation. Windows Azure, June 2011. <http://www.microsoft.com/windowsazure/>.
- [19] Microsoft Corporation. Windows Server 2008 R2, June 2011. <http://www.microsoft.com/windowsserver2008>.
- [20] Liam O'Brien, Paulo Merson, and Len Bass. Quality attributes for service-oriented architectures. In *Proceedings of the International Workshop on Systems Development in SOA Environments, SD-SOA '07*, pages 3–, Washington, DC, USA, 2007. IEEE Computer Society.
- [21] Alan Page and Ken Johnston. *How We Test Software at Microsoft*. Microsoft Press, 2008.
- [22] L.M. Riungu, O. Taipale, and K. Smolander. Software testing as an online service: Observations from practice. In *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, pages 418–423, april 2010.
- [23] Shilpa Venkateshwaran. Cloud Testing: Trends and Challenges, Nov. 2008. <http://solutions.wolterskluwer.com/blog/2010/12/cloud-testing-trends-challenges> (June 2011).
- [24] W.T. Tsai, J. Gao, Xiao Wei, and Yinnong Chen. Testability of software in service-oriented architecture. In *Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International*, volume 2, pages 163–170, Sept. 2006.
- [25] E J Weyuker. Axiomatizing software test data adequacy. *IEEE Trans. Softw. Eng.*, 12(12):1128–1138, 1986.
- [26] Philipp Zech. Risk-based security testing in cloud computing environments. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pages 411–414, March 2011.
- [27] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit testing coverage and adequacy. *ACM Computing Surveys*, 29(4):366–427, December 1997.