Collaborative Testing of SaaS
-- Testing research from system perspectives

Xiaoying Bai
Department of Computer Science and Technology,
Tsinghua University, Beijing, China

April 26, 2010
Outline

- SaaS and its testing needs
- The CV&V framework
  - Architecture
  - Testing Contracts
  - Testing Automation
  - Testing Intelligence
- Summary and look forward
Software as a Service

Services are autonomous, platform-independent computational elements that can be described, published, discovered, orchestrated and programmed using standard protocols to build networks of collaborating applications distributed within and across organizational boundaries.

The Software Paradigm Shift

Constituents
- Software Packages → Loosely Coupled Services
- Program Centric → Model-Driven
- In-house Development → Standard-Based Global reuse
- Product-Based Purchase → Service-Based Renting
- Installation & Patches → User Transparent Online Evolution
Four Maturity Levels of SaaS (wikipedia)

- **Level 1 - Ad-Hoc/Custom**
  - Each customer has its own customized version of the hosted application and runs its own instance of the application on the host's servers.

- **Level 2 – Configurable**
  - Greater program flexibility through configurable metadata, so that many customers can use separate instances of the same application code.

- **Level 3 - Configurable, Multi-Tenant-Efficient**
  - Adds multi-tenancy so that a single program instance serves all customers.

- **Level 4 - Scalable, Configurable, Multi-Tenant-Efficient**
  - Adds scalability through a multitier architecture supporting a load-balanced farm of identical application instances, running on a variable number of servers.
“Web services are not yet widely used because of security concerns. But there’s an even bigger roadblock waiting just down the road – it’s called trust. The big issue is ’Will the service work correctly every time when I need it?’ As yet few are thinking about the issues of testing and certification. We suggest that testing and certification of Web services is not business as usual and that new solutions are needed to provide assurance that services can really be trusted.”

Multi-Party Architecture

**The 3 party architecture**
- Dynamic service discovery
- Dynamic binding and composition
- Standard-based collaboration

**The 4 party architecture (SLA@SOI)**
- Dynamic resource allocation
- Continuous monitor and prediction
- Service Level Agreement
New Needs of SaaS Testing

- **Infrastructure Service**
  - Testing needs to be incorporated into the dynamic process of service discovery, binding, composition, and evolution.

- **Collaborative Testing**
  - Testing needs to be performed from different perspectives during service collaboration
    - Service provider: conformance to functional and performance commitments
    - Service user: usage in different composition context
    - Service broker: ranking and predicting based on cross usage statistics
  - The loose couple parties may share testing assets
New Needs of SaaS Testing

- Adaptive Testing
  - Testing needs to be adaptable in reaction to the dynamic changes in the target services, such as re-composition and re-configuration
    - Adaptive test selection, deployment, planning, ....

- Scalable Testing
  - Testing needs to address the scalable issues in the massive and fluctuant sharing of services in the open Internet environment
Outline

- SaaS and its testing needs
- The CV&V framework
  - Architecture
  - Testing Contracts
  - Testing Automation
  - Testing Intelligence
- Summary and look forward
The CV&V Framework
(Collaborative Verification and Validation)
CV&V Architecture:  
I. Trustworthy Service Broker

- The UDDI server is not accountable for the quality, including performance, dependability, and cost-effectiveness of the services.
- The traditional UDDI server is extended with testing capabilities so that services can be tested and evaluated to ensure satisfied quality.
Check-In Testing
-- To ensure that only qualified services will be registered and published
-- To rank the services based on broker statistics

Check-Out Testing
-- Regression testing on the candidate services to assess whether the services have changed and the change impacts.
CV&V Architecture:
II. Test Broker

- The test provider can be
  - Service provider
  - Service requester
  - Independent tester

- The tester can be
  - Service provider
  - Service requester
  - Service broker
  - Certification organization
  - Independent tester
Testing Contracts

- **TSC: Testing Service Contracts** – Collaboration between test and service under test
  - Test designers get service specification for test generation.
  - Test executors exercise the test on the service interface.
  - Test evaluators evaluate the service based on test results and defect reports.

- **TCC: Test Collaboration Contracts** – Collaboration among test participants
  - Protocols for collaborative test design, execution, and evaluation.
The Test Ontology Model
The Test Ontology Model

- **Test Design**
  - Test Data
    - data pool, data partitions, data selection strategy, and data value
  - Test Behavior
    - test procedures, expected results

- **Test Execution**
  - Test Plan
  - Test Suite
  - Test Schedule
  - Test Configuration
  - Test Results
Contract-Based Automatic Testing

- Test generation
  - A model-based approach to automatic transform from services specifications to test artifacts

- Test execution
  - An agent-based approach to automatic schedule test plan, allocate tasks, deploy to host computers and connect SUT for executing test cases
Contract-Based Automatic Testing

Service Spec
- WSDL
- BPEL
- OWL-S

Spec Analysis → Modeling
- Model Checking

Test Agent
- Driver
- Monitor

Controller → Test Generation
- Adaptive Strategy Selection
- Result Analysis
Intelligent Testing

- Semantic-Aided Testing
  - Ontology-based test generation
  - Ontology-based test collaboration contracts

- Intelligent Test Agents
  - The autonomous model for agent test knowledge carrying and self decision making
  - The social model for test collaboration
Ontology-Based Test Data Generation

1. Analyze the parameters of the service specified with OWL-S
2. Create the data pools for the parameters for different test purposes
3. Derive the class hierarchy for the data partitions for each pool of each parameter
4. Derive and define the class restrictions and relationships of the data partitions
5. Derive the data values in each partition
Hierarchical Analysis of Data Partitions

1. Direct mapping from S to T

```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accommodation</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>BedAndBreakfast</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>BudgetAccommodation</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Campground</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>LuxuryHotel</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>clz_Accommodation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>clz_Hotel</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>clz_BedAndBreakfast</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>clz_BudgetAccommodation</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>clz_LuxuryHotel</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>clz_Camground</td>
</tr>
</tbody>
</table>
```
Hierarchical Analysis of Data Partitions

2. Sub-partitions identified by class property analysis
Hierarchical Analysis of Data Partitions

3. Remove redundant class by class relationship and property restriction analysis

1. **BudgetAccommodation** are those with one or two star ratings, i.e.,
   \[ \text{BudgetAccommodation} = \{ \text{Accommodation} \mid \text{hasRating}(\text{OneStar} \cup \text{TwoStar}) \} \]
2. **Campground** are those with one star ratings, i.e.,
   \[ \text{Campground} \subseteq \{ \text{Accommodation} \mid \text{hasRating}(\text{OneStar}) \} \]
3. **LuxuryHotel** are those with three star ratings, i.e.,
   \[ \text{LuxuryHotel} \subseteq \{ \text{Hotel} \mid \text{hasRating}(\text{ThreeStar}) \} \]
Data Partition Relationship Analysis

- The relationships defined in the service ontology can be mapped to the test data ontology
  - Subclass
  - Equivalent
  - Disjoint
  - Intersection
  - Union
  - Complementary

\[
\begin{align*}
& \text{(Hotel} \cap \text{BedAndBreakfast=} \varnothing \text{)} \cap \\
& \text{(Clz\_Hotel=} \text{Hotel}) \cap \\
& \text{(Clz\_BedAndBreakfast=} \text{BedAndBreakfast}) \cap \\
& \text{(Clz\_TwoStarHotel} \sqsubseteq \text{Clz\_Hotel}) \cap \\
& \text{(Clz\_TwoStarBedAndBreakfast} \sqsubseteq \text{Clz\_BedAndBreakfast}) \Rightarrow \\
& \text{(Clz\_Hotel} \cap \text{Clz\_BedAndBreakfast=} \varnothing \text{)} \cap \\
& \text{(Clz\_TwoStartHotel} \cap \text{Clz\_TwoStartBedAndBreakfast=} \varnothing \text{)}
\end{align*}
\]
Data Partition Property Restriction

- **Cardinality constraints**
  - The minimum and maximum number of values of a class property
  - The *Accomodation* can have at maximum 1 value for *hasRating* property
  - Can be mapped directly from the service data definition to the test data definition.

- **Value constraints**
  - The value scope and range of a property
  - The *hasRation* property of *Accomodation* has a *owl:hasValue* constraint as an enumeration of the accepted values “{*OneStar, TwoStar, ThreeStar}*”
  - Test data are derived by property decomposition. Hence, the value constraint for the service data cannot be mapped directly to the test data.
Data Value Generation

- Automated Data Value Generation
  - Enumerated Class Instances
  - Generated based on the constraint analysis of the property
  - Pre-defined data or accumulated historical data
  - Defined rules for deriving data values

```python
def has_Companion = Tourist(?x) ∧ Tourist(?y)
  ∧ Guide(?g) ∧ hasGuide(?x, ?g) ∧ hasGuide(?y, ?g)
  ∧ differentFrom(?x, ?y)
  → has_Companion(?x, ?y)
```
Ontology-Based C&C Checking

- C&C Checking is to ensure that all the testing assets and service ontology are properly covered that no inconsistent or conflicting definitions of the assets can occur.

- C&C rules are defined in four categories
  - Classes, relationships, properties, and restrictions
  - Internal C&C: testing assets within TOM
    - In the TOM, each data pool has at least one data partition and each data partition has at least one data value.
  - External C&C: testing assets against the OWL-S
    - For each parameter in the OWL-S specified service
      1. There is at least one data pool defined
      2. There is at least one corresponding equivalent class defined in the data pool
      3. Each property of the parameter has at least one corresponding data partition
BDI Agent Design

[Diagram of BDI Agent Design showing the relationship between Desire, Intension, and Believe, with specific steps like Test Case Selection, Agent Selection, Task Allocation, Runner Status, Task Status, and Test Results.]
The Prototype System

GUI Editor

Test
- Test Generator
- Test Scheduler
- Test Agents
- Test Analyzer
- Reliability Analyzer

Model Analyzer (WSDL/OWL/OWL-S/TOM/SWRL)
Model API (WSDL/OWL/OWL-S/TOM/SWRL)

Management

Models
- OWL-S
- TOM
- OWL
- WSDL

Execution
- SOAP Engine
- WSDL Engine
- OWL-S Engine
Service Spec. → Modeling & Analysis → Test Generation → Deploy & Execution
Outline

- SaaS and its testing needs
- The CV&V framework
  - Architecture
  - Testing Contracts
  - Testing Automation
  - Testing Intelligence
- Summary and look forward
Summary

- SaaS presents new needs of testing from system perspectives
  - Infrastructure service
  - Collaborative testing
  - Adaptive testing
  - Scalable testing
- Collaborative testing aims to address the unique issues in SOA
  - Loosely coupled, autonomous services
  - Standard based collaboration
  - Dynamic architecture
Summary

- The major considerations in the CV&V framework
  - How to incorporate testing mechanism into the multi-tier service architecture?
  - How to define the communication protocols at both syntactic and semantic levels?
  - How to automate the testing activities and coordinate the process to enable online continues testing?
  - How to enhance the effectiveness and efficiency of testing using intelligent computing techniques?
Future Work

- Testability
  - Extension and annotation of current service specifications to enhance testability

- Semantic
  - Incorporate knowledge to guide test generation and execution

- Autonomous
  - Towards self-* service-based systems

- Dynamic
  - React to online changes of the targets
References

THANK YOU!

Xiaoying Bai
Ph.D, Associate Professor
Department of Computer Science and Technology,
Tsinghua University
Beijing, China, 100084
Phone: 86-10-62794935
Email: baixy@tsinghua.edu.cn