STS-BASED MODELS FOR WEB SERVICE PERFORMANCE EVALUATION

(Presentation at SOA Security Course)

Romaric Sagbo
Ph.D Student

kouessi.sagbo@unimi.it
SESAR Lab – Dipartimento di Informatica
Università Degli Studi di Milano

May 30, 2013
Introduction

- Model-Driven Development (MDD) changes the focus of software development from the code to models [3, 4].
- Model-based approaches allow to better analyse the non-functional properties.
- Web service performance is a well-recognized problem in SOA management [5].
- Web service performance evaluation through simulation and test cases is important.
Motivations

- Accurate and rapid evaluation of web services performance is still a problem.
- Testing phases are time-consuming and costly.
- Difficult to assess the behavior of the web service before the end of development.
- Lack of framework to study the behavior of the web service before the development.
Our framework for performance evaluation

**Service Interface**

**Service Code**

**STS-based Model**

**Training Data**

**STS\(_s\)**

**STS-based Model + transition probabilities + delay distributions**

**Simulation Script**

**Output\(_s\)**

**STS\(_t\)**

**Simulation layer Testing layer**

**Compare the results**

**STS-based Model + idioms**

**Test Cases**

**Output\(_t\)**
Symbolic Transition System (STS)

Modeling a software/service as a transition system is a traditional approach used to **test functional properties** of systems.

- A Symbolic Transition System (STS) is a finite state automaton that describes the behavior and evolution of a software/service. It consists of states and transitions between states, labeled with actions, guards, and update mapping.
- Two types of actions:
  - input actions, denoted as `?function<parameters>;`
  - output actions, denoted as `!function<parameters>`.
An STS-based model is formally defined as follows.

**Definition (STS)**

A STS is a tuple $<S,s_1,V,I,A,\rightarrow>$ where:

- $S=<s_1,...,s_n>$ is a set of states, and $s_1$ the initial state;
- $V$ is a set of location (internal) variables;
- $I$ is a set of interaction variables representing operation inputs and outputs;
- $A$ is a set of actions (operations);
- $\rightarrow$ is a transition relation.
Reference scenario

Real world web service as reference scenario to validate our approach.

- IFX-based Reverse ATM web service.
- IFX (*Interactive Financial Exchange*) is an XML specification which defines the electronic exchange of financial data between financial institutions, business, and consumers through Internet.

The following operations are implemented:

- *Signon*, which authenticates the users by checking the validity of their credentials;
- *DebitAdd*, which allows authenticated user to withdraw funds;
- *CreditAdd*, which allows authenticated user to deposit funds.
Reference scenario: STS-based Model
An STS-based model for performance testing extends the standard STS-based model of a service with idioms.

- For monitoring the execution and service times.
- For logging.
- For security checks.

The idioms are expressed as annotations to STS. Ex. Idioms \texttt{startclock(t)} and \texttt{endclock(t)} are added to trigger the execution and service times.
An STS-based model extended for performance monitoring is formally defined from Definition 1 as follows.

Definition (STS<sub>t</sub>)

An STS-based model extended for testing STS<sub>t</sub> is a tuple 
\( <S, s_1, \mathcal{V}, \mathcal{I}, A, ID, \overset{id}{\rightarrow} > \) where:

- \( ID \) is the set of performance idioms;
- \( \overset{id}{\rightarrow} \), with \( id \in ID \), extends the transition relation in Definition 1 with idioms.
STS extended for performance monitoring (3)
STS extended for simulation

For simulation purposes, the standard STS model is extended with transition probabilities and delay distributions.

- The probability to move from one state to another.
- The distribution associated to the delay (waiting time) or the time needed to complete the task associated to the transition.
An STS-based model extended for simulation is formally defined from Definition 1 as follows.

**Definition (STS\textsubscript{s})**

An STS-based model for simulation STS\textsubscript{s} is a tuple $<S,s_1,V,I,A,\text{prob},\text{distr}>$ where:

- $\text{prob} \in [0,1]$ is a transition probability;
- $\text{distr}$ is a probability distribution of waiting times;
- $\text{prob},\text{distr}$ extends the transition relation in Definition 1 using probabilities and delay distributions.
STS extended for simulation (3)
XML encoding of the STS

- Enable automatic generation of the performance interceptors, executed by test drivers, and simulation script.
- Extension of the STS standard XML definition with the following three XML tags:
  - `<ns1:idiom idiom1; idiom2;` that allows to annotate the model with idioms;
  - `<ns1:probability value>` that allows to define the probability associated with state transitions;
  - `<ns1:distribution value>` that allows to define the delay distribution associated with state transitions.
XML encoding of the STS extended with probabilities and delay distributions.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ns1:STS>
    <ns1:location>1</ns1:location>
    ......<br />
    <ns1:location>7</ns1:location>
    <ns1:location>7a</ns1:location>
    <ns1:location>7b</ns1:location>
    ...
    <ns1:initialLocation>1</ns1:initialLocation>
    <ns1:interactionVars>
        <ns1:interactionVar>
            <ns1:name>token</ns1:name>
            <ns1:type>String</ns1:type>
        </ns1:interactionVar>
        <ns1:interactionVar>
            <ns1:name>amount</ns1:name>
            <ns1:type>Double</ns1:type>
        </ns1:interactionVar>
    </ns1:interactionVars>
    ...
    <ns1:messages>
        <ns1:message>
            <ns1:name>DebitAdd</ns1:name>
            <ns1:kind>input</ns1:kind>
        </ns1:message>
    </ns1:messages>
    <ns1:switch>
        <ns1:from>7</ns1:from>
        <ns1:to>7a</ns1:to>
        <ns1:message>DebitAdd</ns1:message>
        <ns1:kind>input</ns1:kind>
        <ns1:restriction>amount>0 &amp;&amp; token!=null</ns1:restriction>
        <ns1:update />
        <ns1:distribution>delay in [0ms,4ms]</ns1:distribution>
        <ns1:probability>1</ns1:probability>
    </ns1:switch>
    ...
    <ns1:switch>
        <ns1:from>7a</ns1:from>
        <ns1:to>7b</ns1:to>
        <ns1:message>Check_Balance</ns1:message>
        <ns1:kind>input</ns1:kind>
        <ns1:restriction />
        <ns1:update />
        <ns1:distribution>delay in [1ms,4ms]</ns1:distribution>
        <ns1:probability>1</ns1:probability>
    </ns1:switch>
</ns1:switches>
</ns1:STS>
```
Implementation

Performance Interceptors

- Performance monitoring code automatically integrated within the service code using the STS-based model extended with idioms.
- Monitoring the performance by observing the service operation calls and by measuring their response and service times.
- Implementation using the Enterprise Java Bean (EJB) interceptors.

Simulation Scripts

- Automatic script generation based on the STS-based model annotated with transition probabilities and delay distributions.
- Use of the script generated to estimate some performance indicators.
EJB performance interceptors

```java
@Interceptors (ExecutionTimeMeasure.class)
public String DebitAdd(Double amount, String token) {
    // Your code here
}

public class ExecutionTimeMeasure {
    @AroundInvoke
    public Object ServiceTime(InvocationContext ctx) throws Exception {
        long startClock = System.currentTimeMillis();
        Object[] parameters = ctx.getParameters();
        try {
            return ctx.proceed();
        } catch (Exception e) {
            logger.warning("Error calling ctx.proceed method");
            return null;
        } finally {
            long stopClock = System.currentTimeMillis() - startClock;
        }
    }
}
Algorithm for simulation script generation

INPUT: \( STS_s \)
OUTPUT: Simulation script

MAIN
Let \( e=(s_1, s_2) \) be an edge between two states \( s_1 \) and \( s_2 \) and \( p_t := 0.01 \) the probability threshold

\[
STS_s = \langle S, s_0, V, I, A, \overset{\text{prob, delay}}{\rightarrow} \rangle = \text{loop_unroll}(STS_s, p_t)
\]

foreach \( s_i \in S \) do
flag \( (s_i) := "\text{Unexplored}" \)
return ProcessState \( (s_0) \)

PROCESS_STATE \( (s) \)
if \( |\text{children}(s)| == 0 \)
flag \( s := "\text{Visited}" \)
else {
if \( s \) has flag "Unexplored"
return add_delay \( (s) \)
foreach \( s_i \in \text{children}(s) \) do
process_state \( (s_i) \)
}

ADD_DELAY \( (s) \)
if \( |\text{children}(s)| == 1 \) {
e = (s, \text{children}(s))
if \( e.\text{delay} != \text{null} \)
return generate_delay \( (e.\text{delay}) \)
flag \( (s) := "\text{Visited}" \)
}
else {
foreach edge \( e_i = (s, s_i), (s_i \in \text{children}(s)) \) do {
if \( e_i.\text{delay} != \text{null} \)
return generate_proba_delay \( (e_i.\text{delay}, e_i.\text{prob}) \)
flag \( (s) := "\text{Visited}" \)
}
Simulation script generated from our algorithm

```java
public long EvaluateServiceTime() {
    long beginT = System.currentTimeMillis();
    Distribution event = new GenerateRandomEvent();

    // transition (7,7a)
    Delay(Uniform(0,4));
    // transition (7a,7b)
    Delay(Uniform(1,4));
    Double pevent = event.nextRandom();
    switch (pevent) {
        // transition (7b,7c) and (7c,7)
        case pevent <= 0.1:
            Delay(Uniform(1,1));
            Delay(Uniform(1,1));
            // transition (7b,7d) and (7d,7)
            case pevent > 0.1:
                Delay(Uniform(4,7));
                Delay(Uniform(2,9));
    }

    return System.currentTimeMillis() - beginT;
}

// Delay method that performs the waiting feature
public void Delay(Uniform(int start, int end)) {
    int time = Uniform(start, end);
    try {
        Thread.sleep(time);
    }
    catch (InterruptedException ex) {
        Thread.currentThread().interrupt();
    }
}
```
This framework allows to generate automatically the performance monitoring code and the simulation scripts from the extended STS-based models.

The **STS2JAVA** framework implements two modules:
- Testing module;
- Simulation module.
Framework STS2JAVA (3)

import org.rsagbo;
import org.w3c.dom.*;

public class STS2Java extends JFrame {
    ...
    public void actionPerformed(ActionEvent event) {
        int value;
        if (event.getSource() == addButton) {
            if (STS2Java.stsModelPath != null) {
                ...
            }
        }
    }

    <ns1:STS>
    <ns1:location>1</ns1:location>
    <ns1:location>2</ns1:location>
    <ns1:initialLocation>1</ns1:initialLocation>
    <ns1:locationVars><ns1:locationVar>
    <ns1:name>saved_s</ns1:name>
    <ns1:type>enum</ns1:type>
    ...
    </ns1:STS>
Plugin STS2JAVA

- Make available our framework STS2JAVA as a plugin for the main Java IDE, Eclipse and Netbeans.
- Enable the Java IDE to offer the annotation of the web service code with the performance monitoring code.
- Enable the Java IDE to generate also the performance interceptors directly from the appropriate STS-based model.
- Generate the simulation scripts within the IDE by choosing the STS-based model annotated for this goal.
- Generate a code template for performance interceptors.
Implementation

Plugin STS2JAVA (2)

STS2JAVA PLUGIN

- Generate a simulation script
- Generate an inspector code
- Annotate a method with interceptor
- Generate an interceptor template

Steps

1. Choose your action
2. Choose your appropriate STS Model
3. Generate your Simulation Script or Performance Code
4. Step #4

Romaric Sagbo Ph.D Student

STS-BASED MODELS FOR WEB SERVICE PERFORMANCE EVALUATION

May 30, 2013 26/35
Experimental results

- CreditAdd operation service time with performance idioms
- CreditAdd operation service time with simulation script
Experimental results: Comparison

- Comparison of simulation and testing results for CreditAdd operation

![Graph showing comparison of simulation and testing service times for CreditAdd operation](image-url)
Experimental results: Statistical analysis

- Chi-Square test

To better evaluate the quality of our simulation results, we computed the statistical distance between the two data (test and simulation).

<table>
<thead>
<tr>
<th>Test Cases</th>
<th>Distance $\chi^2$ [18, 19]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creditadd</td>
<td>$2.597 \rightarrow (P&gt;0.96)$</td>
</tr>
<tr>
<td>Debitadd</td>
<td>$2.583 \rightarrow (P&gt;0.96)$</td>
</tr>
</tbody>
</table>
Summary

Our experimental results show that:

- Simulation scripts can represent a suitable solution for an early assessment of service performance.
- The performance interceptors provide a good approach to measure the performance of existing service.
- Both performance interceptors and simulation scripts can be used to negotiate and evaluate the performance SLAs of the web service.
Conclusions

- Our work proves that model-based representation of web services can be used to effectively assess the services behavior, as part of the development lifecycle in a partial and full-knowledge scenarios.

- Our future work will consider evaluation of service compositions, and a zero-knowledge scenario where service code and results of real service executions are not yet available.
1 Framework STS2JAVA: This thesis proposes to continue the development of our framework for automatic generation of the performance monitoring code and the simulation scripts from the STS-based models.

2 Plugin STS2JAVA: This thesis is the extension of the previous one. It should propose a plugin of our framework STS2JAVA within the most popular IDEs Netbeans and Eclipse.

3 Instrumented web service for performance analysis: This thesis proposes first to review the existing literature on tools for web performance analysis. Moreover, after analysis, some tools will be selected and used to measure the performance of a sample set of services. The results of the different tools will be compared.

4 Web services crowd-sourcing: This thesis proposes to build a web service dataset composed by the WSDL files of the services available on the web and generate an instrumented client service to test their performance.
References I


Questions?

THANK YOU
MERCI
GRAZIE

...