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
Universita Degli Studi di Milano

May 30, 2013

OUTLINE

- Introduction
- Motivations
- Framework
- STS-based model and its extensions
- Implementation
- Conclusions
- Available thesis

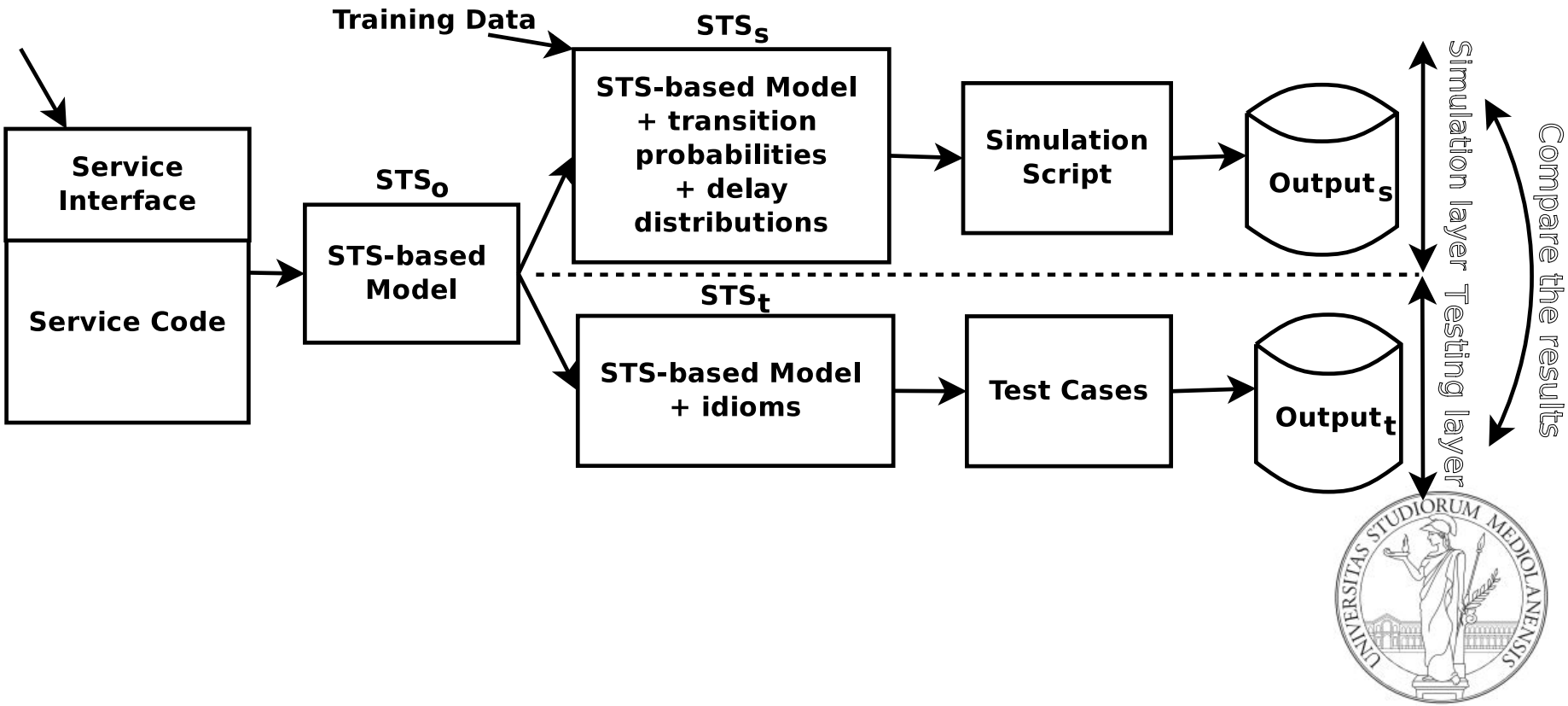
Introduction

- 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



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>*.

Symbolic Transition System (STS) (2)

An STS-based model is formally defined as follows.

Definition (STS_o)

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

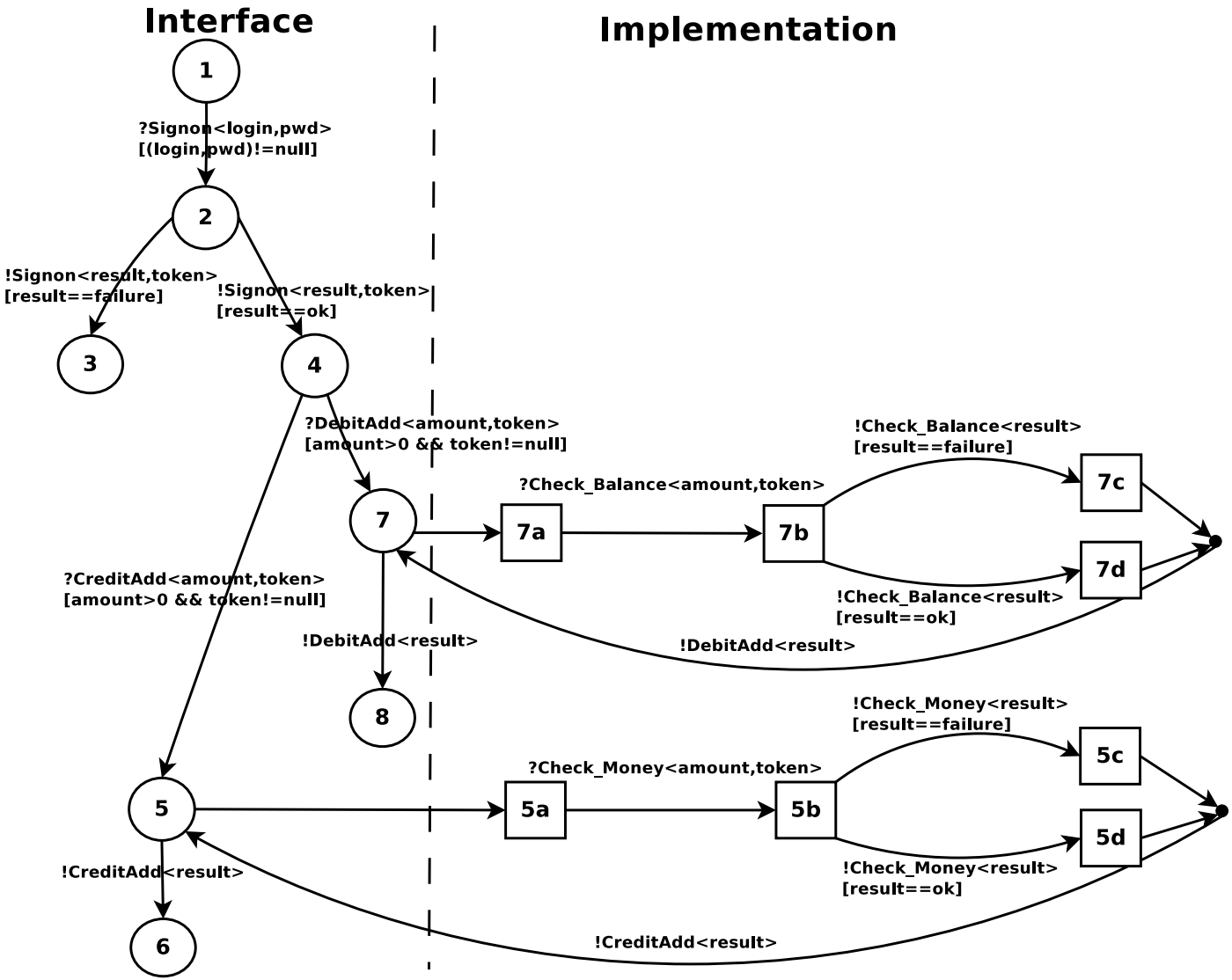
- $\mathcal{S}=<s_1,...,s_n>$ is a set of states, and s_1 the initial state;
- $\mathcal{V}$ is a set of location (internal) variables;
- $\mathcal{I}$ is a set of interaction variables representing operation inputs and outputs;
- $\mathcal{A}$ is a set of actions (operations);
- $\rightarrow$ is a transition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



STS extended for performance monitoring

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 **startclock(t)** and **endclock(t)** are added to trigger the execution and service times.

STS extended for performance monitoring (2)

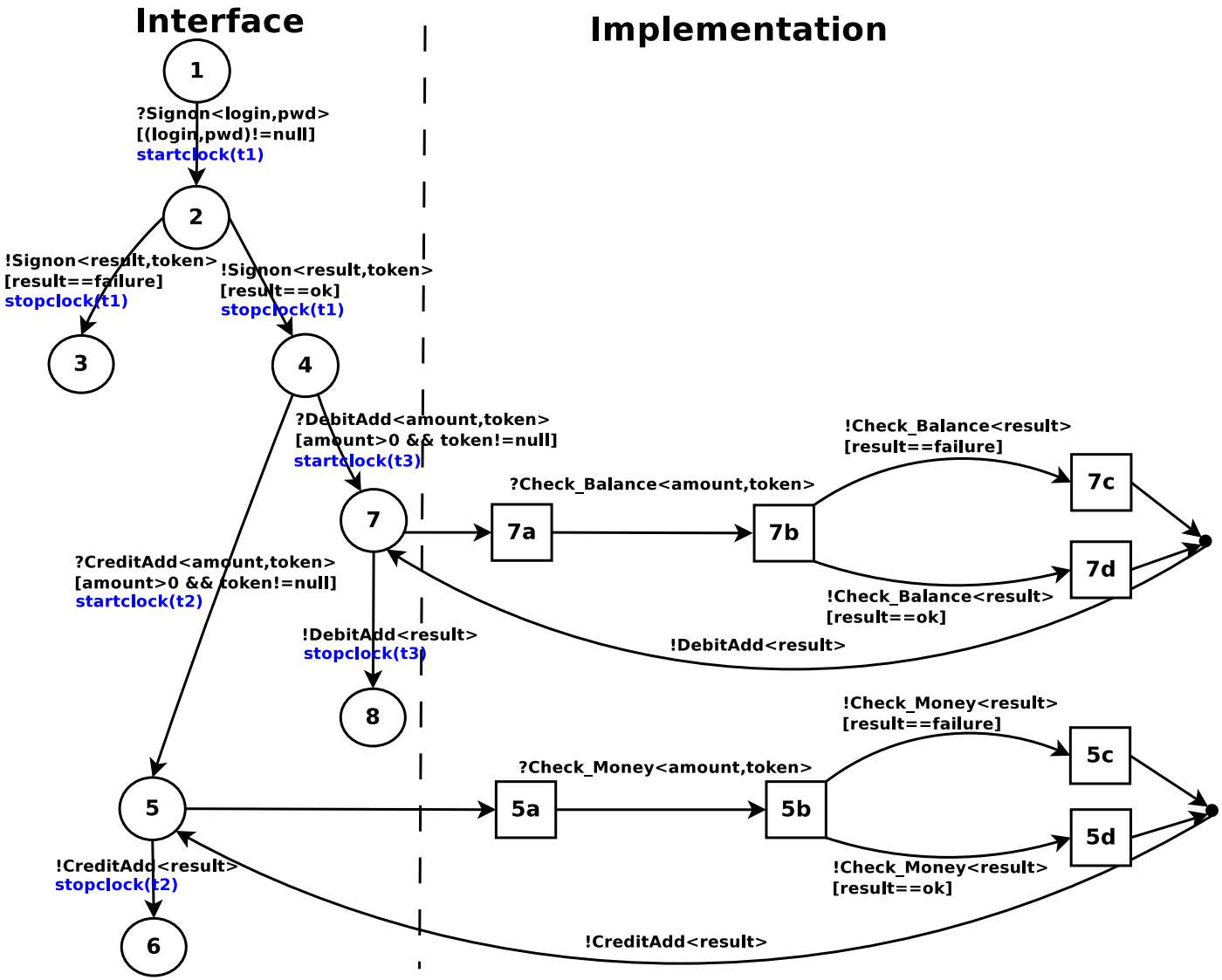
An STS-based model extended for performance monitoring is formally defined from Definition 1 as follows.

Definition (STS_t)

An STS-based model extended for testing STS_t is a tuple $<\mathcal{S}, s_1, \mathcal{V}, \mathcal{I}, \mathcal{A}, ID, \xrightarrow{id}>$ where:

- ID is the set of performance idioms;
- $\xrightarrow{id}$, 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.

STS extended for simulation (2)

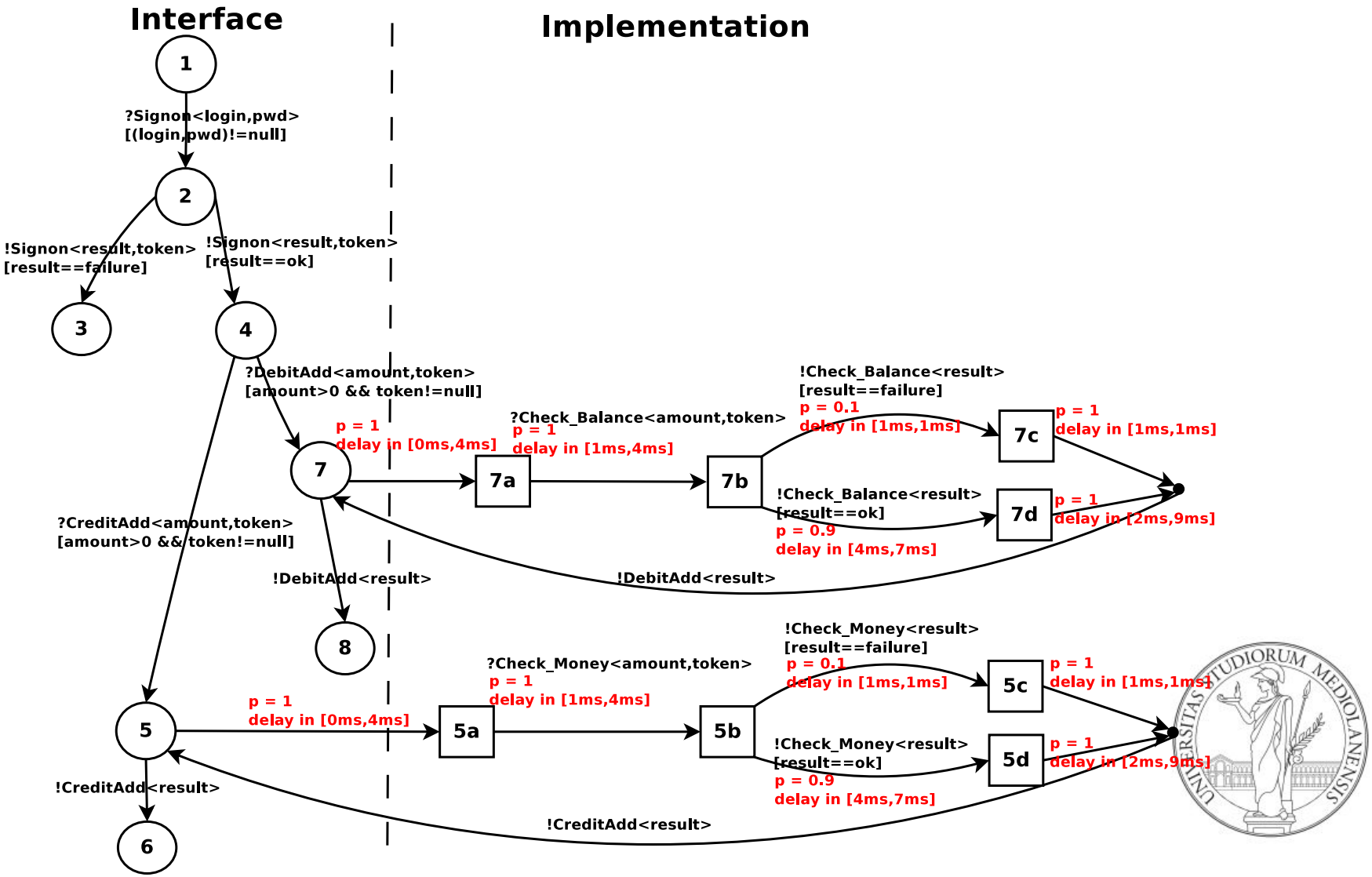
An STS-based model extended for simulation is formally defined from Definition 1 as follows.

Definition (STS_s)

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

- $prob \in [0,1]$ is a transition probability;
- $distr$ is a probability distribution of waiting times;
- $\xrightarrow{prob,distr}$ 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;**</**ns1:idiom**> that allows to annotate the model with idioms;
  - <**ns1:probability**>**value**</**ns1:probability**> that allows to define the probability associated with state transitions;
  - <**ns1:distribution**>**value**</**ns1:distribution**> that allows to define the delay distribution associated with state transitions.

XML encoding of the STS (2)

- XML encoding of the STS extended with probabilities and delay distributions.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:STS>
    <ns1:location>1</ns1:location>
      ......
    <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:param>amount</ns1:param>
            <ns1:param>token</ns1:param>
      ...
    </ns1:messages>
    <ns1:switches>
        <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;&amp;amp; token!=null</ns1:rest
            <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

```
@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
$STS_s = <S, s_0, V, I, A, \xrightarrow{prob,delay}> =$ **loop_unroll**(STS_s, p_t)
foreach $s_i \in S$ **do**
flag(s_i) := "Unexplored"
return ProcessState(s_0)

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

ADD_DELAY(s)
if |children(s)| == 1 {
e = (s, children(s))
if $e.delay$!= *null*
return generate_delay($e.delay$)
flag(s) := "Visited"
}
else {
foreach edge $e_i = (s, s_i), (s_i \in children(s))$ **do** {
if $e_i.delay$!= *null*
return generate_proba_delay($e_i.delay$, $e_i.prob$)
}
flag(s) := "Visited"
}

Simulation script generated from our algorithm

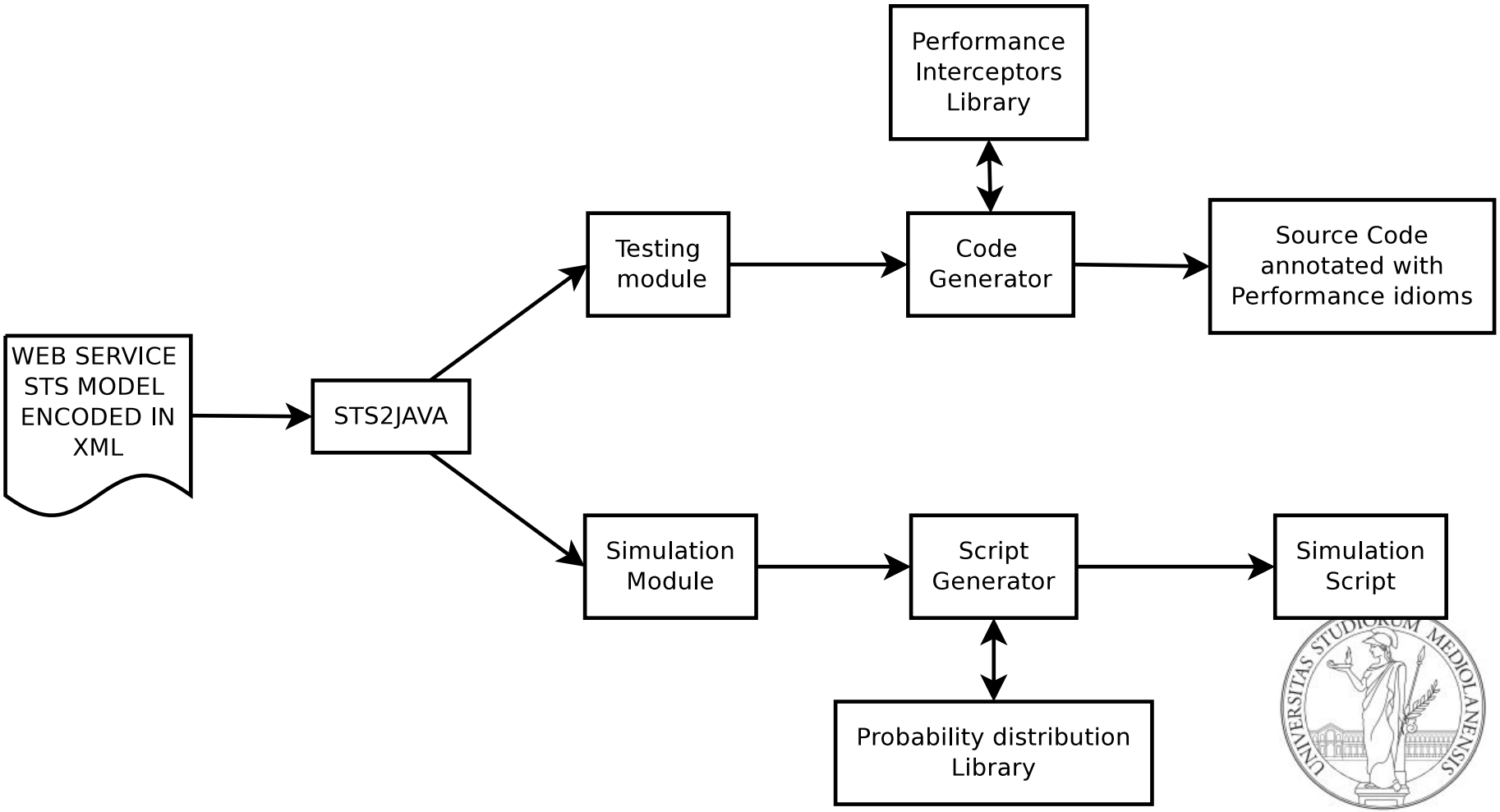
```
public long EvaluateServiceTime() {
long beginT = System.currentTimeMillis();
Distribution event = new GenerateRandomEvent
    ();

// 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();
    }
}
```

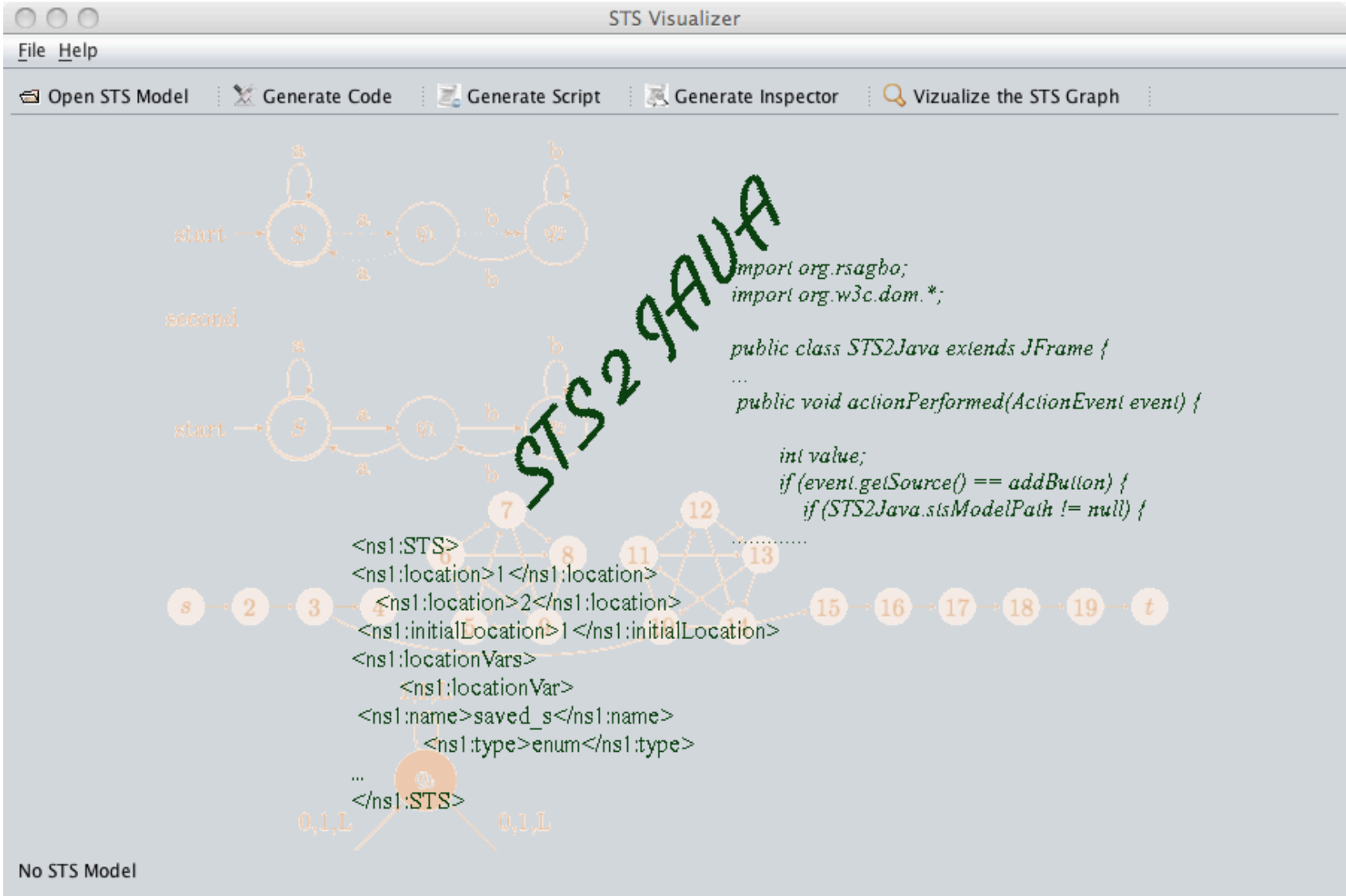
Framework STS2JAVA

- 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 (2)



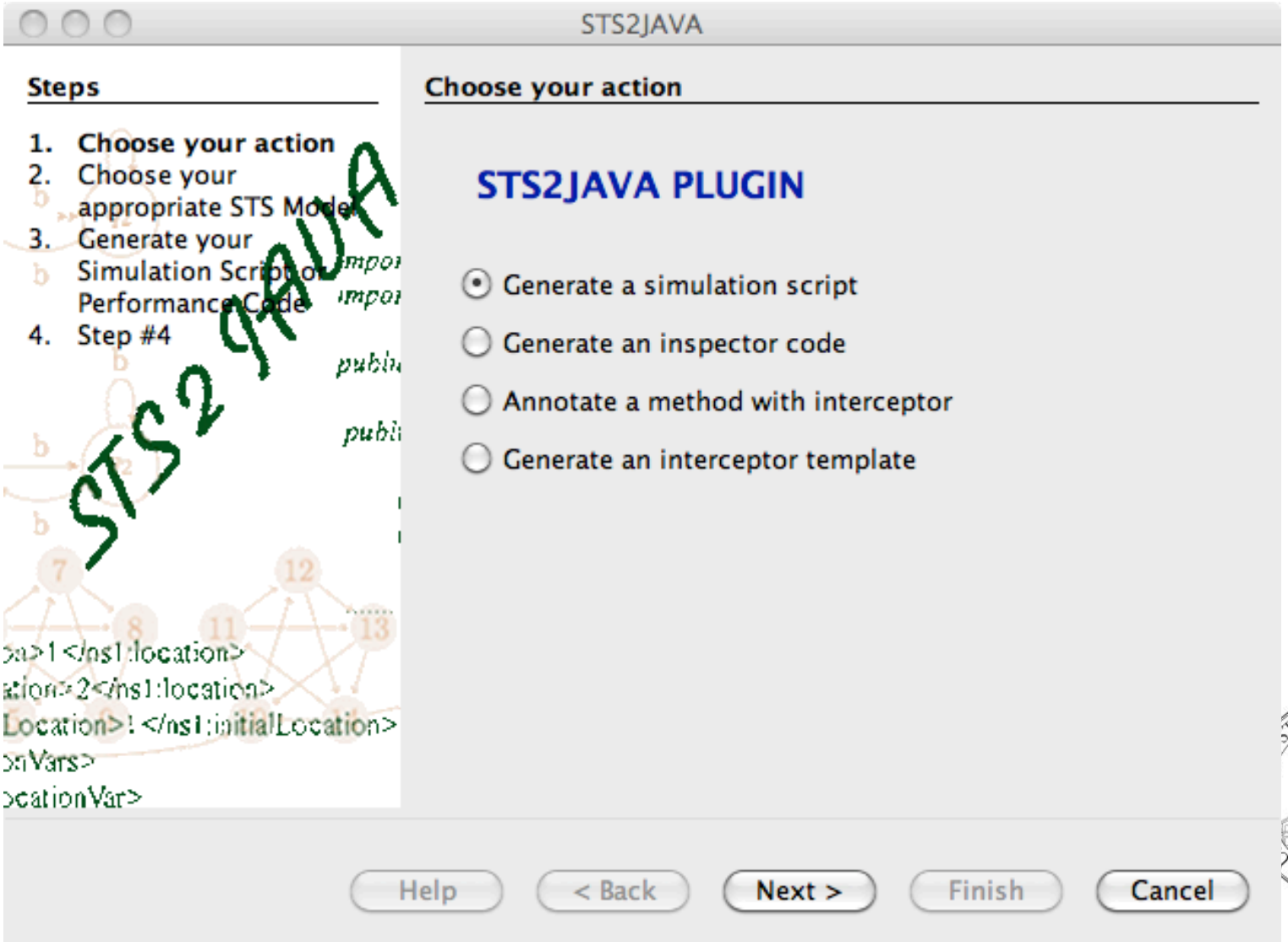
Framework STS2JAVA (3)



Plugin STS2JAVA

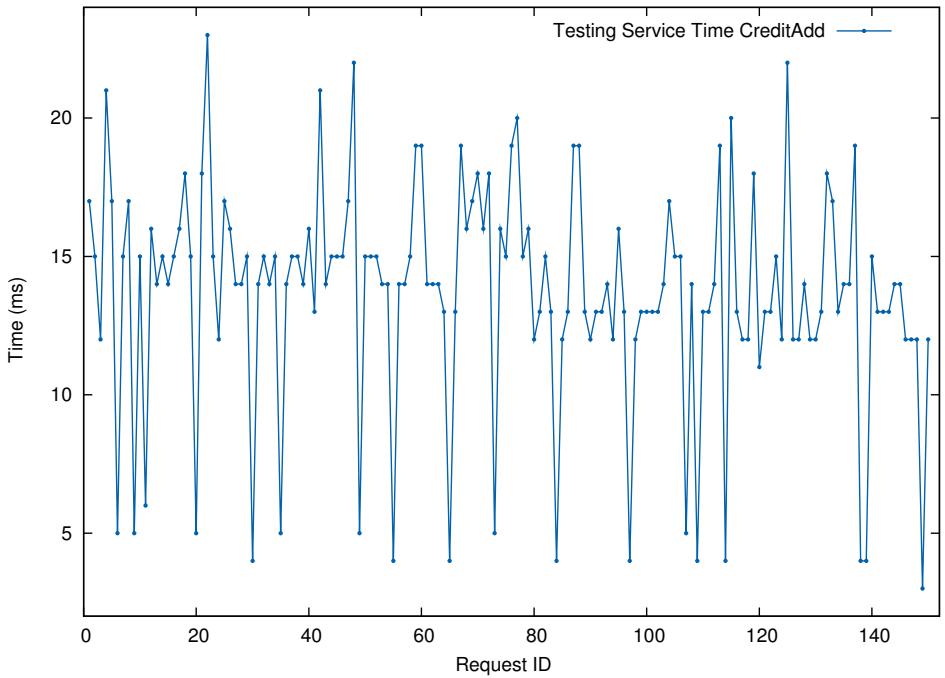
- Make available our framework STS2JAVA as a plugin for the main Java IDE, Eclipse and Netbeans.
- Enable the Java IDE to offer the annotation of the web service code with the performance monitoring code.
- Enable the Java IDE to generate also the performance interceptors directly from the appropriate 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.

Plugin STS2JAVA (2)

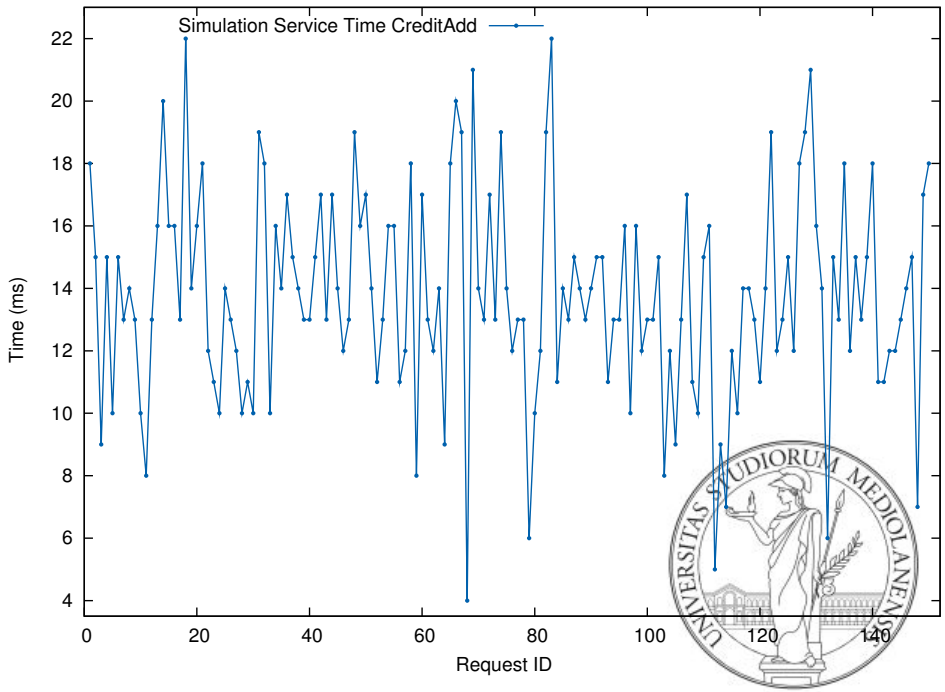


Experimental results

- CreditAdd operation service time with performance idioms

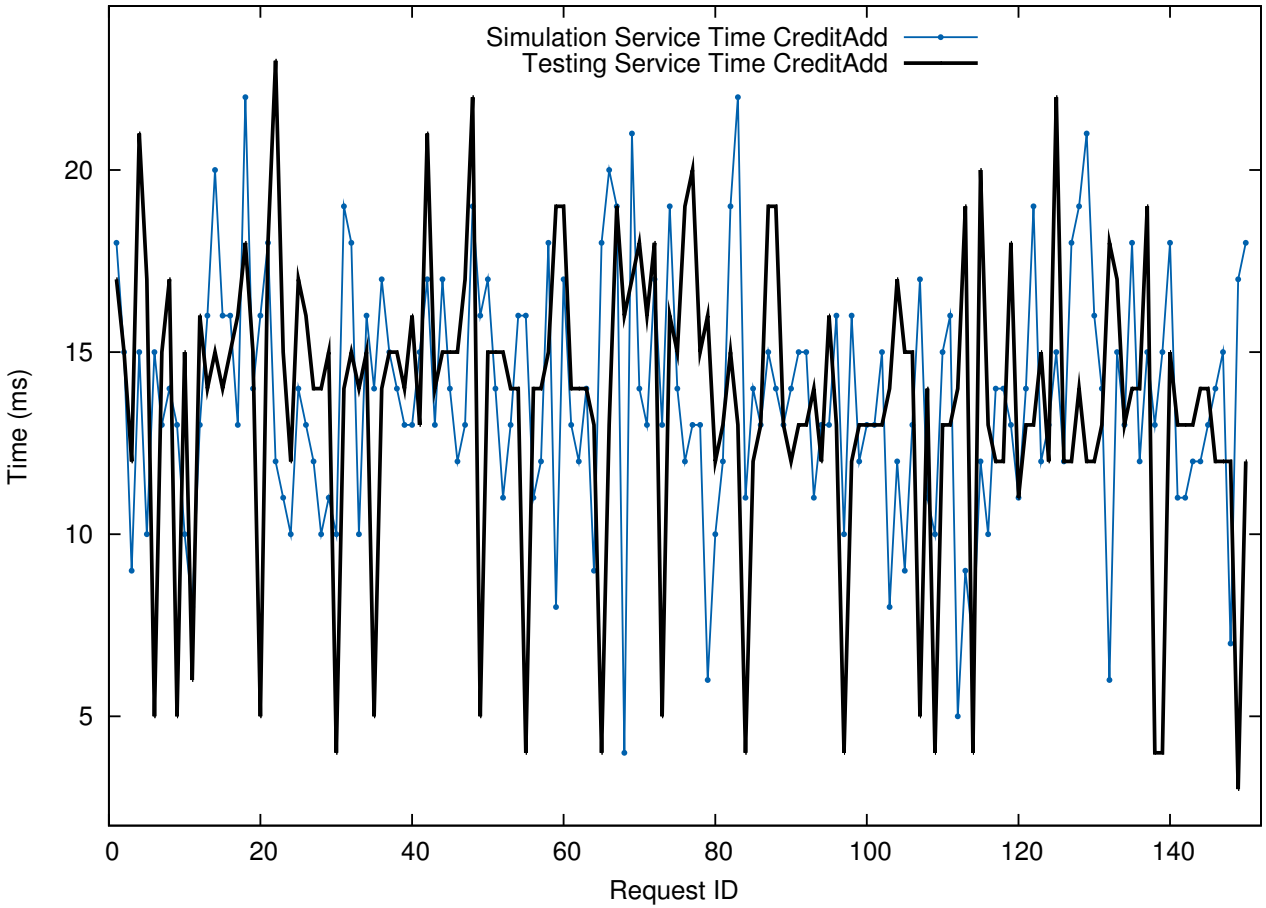


- CreditAdd operation service time with simulation script



Experimental results: Comparison

- Comparison of simulation and testing results for CreditAdd operation



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: Distance between the testing and the simulation distributions

Test Cases	**Distance** χ^2 [18, 19]
Creditadd	2.597 $->(P>0.96)$
Debitadd	2.583 $->(P>0.96)$

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.

Available thesis

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

[1] C. A. Ardagna, E. Damiani and K. A. R. Sagbo. Early Assessment of Service Performance Based on Simulation. In Proc. of SCC 2013, Santa Clara Marriott, CA, USA, June 2013, to appear.

[2] K. A. R. Sagbo and P. Houngue. Quality architecture for resource allocation in cloud computing. In Proc. of ESOCC 2012, volume LNCS 7592, pages 154-168. Springer, Bertinoro, Italy, September 2012.

[3] D. C. Petriu. Software model-based performance analysis. John Wiley & Sons, 2010.

[4] C. Pahl, M. Boskovic and W. Hasselbring. Model-Driven Performance Evaluation for Service Engineering. In Proc. of the 2nd ECOWS Workshop on Web Services Technology, Halle (Saale), Germany, November 2007.

[5] J. Tekli, E. Damiani, R. Chbeir and G. Gianini. Soap processing performance and enhancement. IEEE Transactions on Services Computing, 5(3):387-403, 2012.

[6] V. Rusu, L. du Bousquet and T. Jéron. An approach to symbolic test generation. In proc. of International Conference on Integrating Formal Methods (IFM'00), Pages 338-357, November 2000.

[7] L. Frantzen, J. Tretmans and T.A.C. Willemse. Test generation based on symbolic specifications. In J. Grabowski and B. Nielsen, editors, *FATES 2004*, volume LNCS 3395, pages 1–15. Springer, September 2005.

[8] L. Frantzen, J. Tretmans and T.A.C. Willemse. A symbolic framework for model-based testing. In *Proc. of FATES/RV 2006*, volume LNCS 4262, pages 40–54. Springer, 2006.

[9] J. Tretmans. Model-based testing and some steps towards test-based modelling. In *Proc. of SFM 2011*, Bertinoro, Italy, June 2011.

[10] L. Frantzen, J. Tretmans and R. d. Vries. Towards model-based testing of web services. In Proc. of WS-MaTe 2006, Palermo, Italy, June 2006.

[11] A. Bertolino, G. De Angelis, L. Frantzen and A. Polini. Model-Based Generation of Testbeds for Web Services. In *Proc. of TESTCOM/FATES 2008*, volume LNCS 5047, pages 266–282. Springer, 2008.

References II

[12] M. Aiguier, C. Gaston, P. Le Gall, D. Longuet and A. Touil. A temporal logic for input output symbolic transition systems. In *Proc. of APSEC 2005*, Taipei, Taiwan, December 2005.

[13] W. Kehe, W. Zhuo, Z. Xing and M. Gang. Design and implementation of the monitoring system for ejb applications based on interceptors. In *Proc. of ICACTE 2010*, Chengdu, China, August 2010.

[14] S. Roubtsov, A. Serebrenik, A. Mazoyer and M. van den Brand. I2sd: Reverse engineering sequence diagrams from enterprise java beans with interceptors. In *Proc. of SCAM 2011*, Williamsburg VA, USA, September 2011.

[15] C. Keum, S. Kang and I. Y. Ko. Generating test cases for web services using extended finite state machine. In *Proc. of IFIP TestCom 2006*, volume LNCS 3964, pages 103–117. Springer, 2006.

[16] C. Schwarzl, B. K. Aichernig and F. Wotawa. Compositional random testing using extended symbolic transition systems. In *Proc. of IFIP ICTSS 2011*, volume LNCS 7019, pages 179–194. Springer, 2011.

[17] R. Elfwing, U. Paulsson and L. Lungberg. Performance of SOAP in Web Service Environment Compared to CORBA. In Proc. of APSEC 2002, pages 84-. IEEE Computer Society, 2002.

[18] Notes on the Chi-Squared Distribution. `http://people.math.gatech.edu/~ecroot/3225/chisquare.pdf`

[19] Chi Square table. http://www.medcalc.org/manual/chi-square-table.php

[20] D. M. Endres and J. E. Schinelin. A new metric for probability distributions. IEEE Trans. Inf. Theory, vol. 49, no. 7, pp. 1858-1860, Jul. 2003.

Questions ?

THANK YOU

MERCI

GRAZIE

...