Robustness in Web Services

State-of-the-art and Research Opportunities

Marco Vieira
mvieira@dei.uc.pt
University of Coimbra – Portugal

Marco's Background (1)

- BSc and MSc in Informatics Engineering
- PhD in Computer Science
- Assistant Professor at University of Coimbra
  - Teaching experience of 14 years
    - Hum… I'm getting old ☹
- Adjunct Associate Teaching Professor at CMU
- Large experience in projects with industry
  - Portugal Telecom, Critical Software, ESA, …

Marco's Background (2)

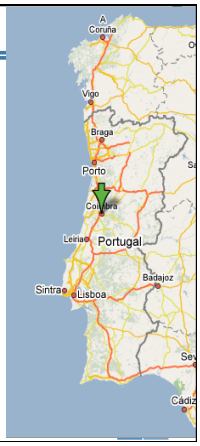
- Research areas:
  - Dependable computing
  - Databases
  - Software engineering
- Teaching areas:
  - Software engineering
  - Databases
  - And many other things…
    - Discrete math, telecommunications, data analysis, strategic information systems, etc, etc.

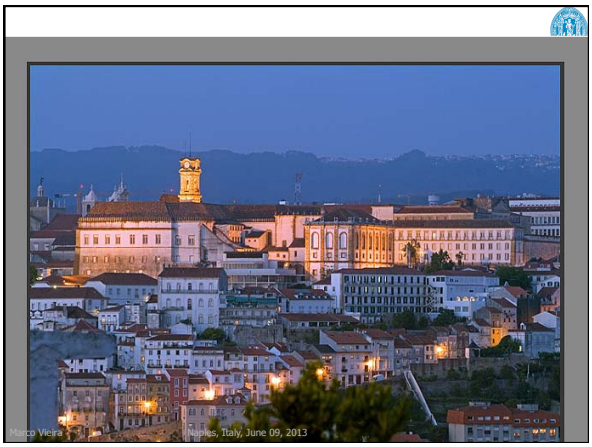
Marco's Background (3)

- With industry:
  - Software engineering
  - Databases
  - Decision support

Coimbra

- City in the center of Portugal
  - 200 Km to the north of Lisbon
- ~ 150 000 people
- All activity is around the University
- Many years of history

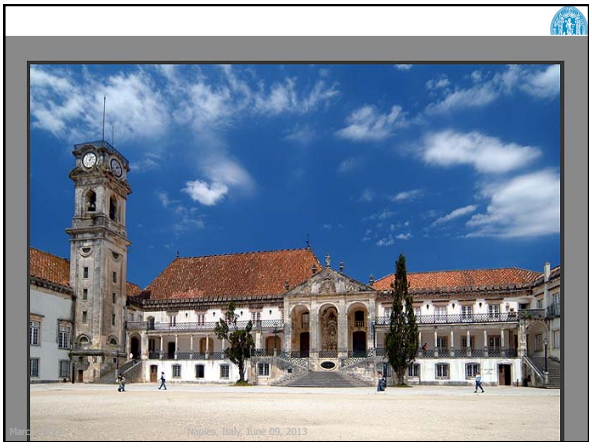



Marco Vieira
Naples, Italy, June 09, 2013


Marco Vieira
Naples, Italy, June 09, 2013


Marco Vieira
Naples, Italy, June 09, 2013
9


Marco Vieira
Naples, Italy, June 09, 2013
10


Naples, Italy, June 09, 2013


PORTUGALNOSEUMELHOR.COM
Marco Vieira
Naples, Italy, June 09, 2013
12

University of Coimbra

- University of Coimbra
  - One of the oldest in the world
  - Created in 1290
  - 9 schools (faculties)
  - Around 30000 students
  - www.uc.pt
- Faculty of Sciences and Technology
  - Biggest school (40% of the UC)
  - 14 departments
  - www.fct.uc.pt

Department of Informatics Engineering

- >50 PhDs
- 6 main programs (Bolonha format):
  - BSc in Informatics Engineering
  - BSc in Design and Multimedia
  - Master in Informatics Engineering
  - Master in Design and Multimedia
  - Master of Software Engineering (with CMU)
  - PhD in Computer Science

www.dei.uc.pt

Center for Informatics and Systems of the UC (CISUC)

- Research center
- Six research groups
  - Adaptive Computation
  - Cognitive and Media Systems
  - Evolutionary and Complex Systems
  - Information Systems
  - Laboratory of Communications and Telematics
  - Software and Systems Engineering

http://cisuc.uc.pt

Software and Systems Engineering (SSE)

- Key people:
  - 13 PhD (Full Members) + 8 PhD (Associate Members)
  - 27 PhD students
- Some research topics:
  - Experimental dependability assessment
  - Fault injection
  - Dependability benchmarking
  - Data Warehousing
  - Intrusion detection
  - Security benchmarking
  - Dependability and security in Web services
  - ...

Let's start with a small game...

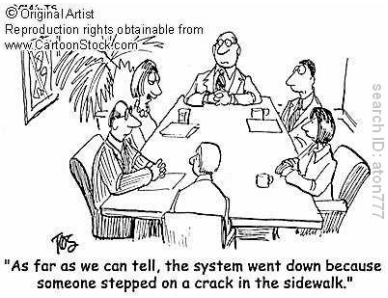
Did you pay attention?

- How many times was the ball passed between people wearing white T-Shirt?
- My answer:
  - I don't know! ☺
  - What I care is about the monkey!?
- What is going on?
  - Haven't you seen the monkey?
  - Let see it again...
  - Why???

Outline

- Robustness testing
- Web services
- Web Services robustness testing
- Case studies
- Fixing/avoiding robustness problems
- Research challenges and opportunities

Why do computers fail?



Too many reasons…

- Hardware problems

Too many reasons…

- Environment problems

Too many reasons…

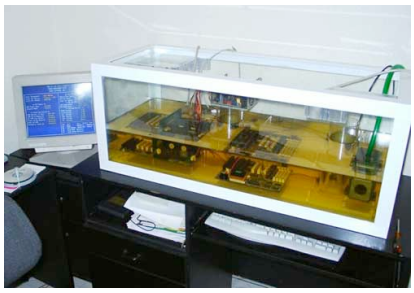
- Bad configuration

Too many reasons…

- Misuse

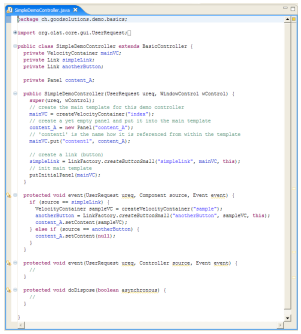
Too many reasons…

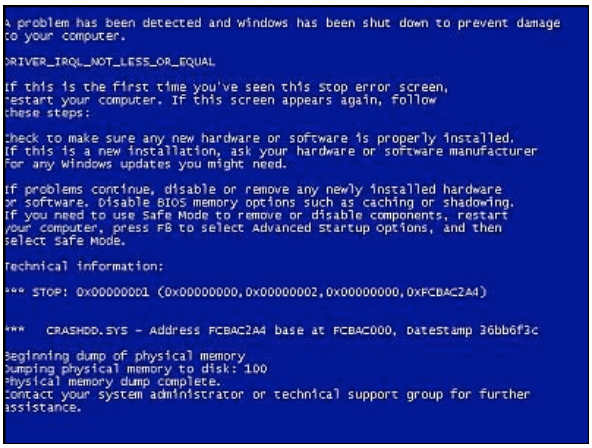
- Not proven design



But most of the times it is due to…

- Software faults





Robustness in WS

Robustness Testing

What is software robustness?

- Software is said robust if it performs well under unusual conditions
  - Conditions that stress designers' assumptions
- Software is typically buggy and fragile
  - i.e., contains errors
- Largely because programs are usually too big and too complex
  - It is difficult to discover and eliminate all the bugs
  - This is especially true with regard to subtle errors that only appear in unusual circumstances

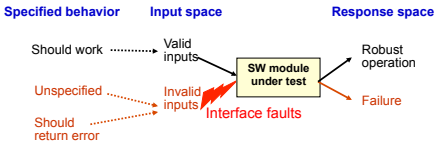
Software bugs

- Studies show a clear prevalence of software faults as the root cause of computer failures
  - [Kalyanakrishnam99]
  - [Lee95]
- Due to the complexity of today's software, the weight of software faults will tend to increase
- Interface faults are related to problems in the interaction among components/modules
  - Particularly relevant when considering component based development and service based SW development

Robustness testing?

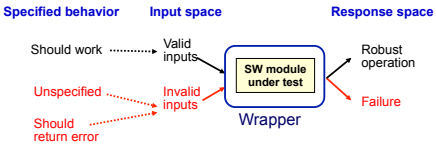
- Characterize the behavior of a system in presence of erroneous input conditions
- Robustness tests stimulate the system in a way that triggers internal errors
  - And in that way expose both programming and design errors in the error detection or recovery mechanisms
- Systems can be differentiated according to the number of errors uncovered
- Two most important robustness testing tools:
  - Ballista [Koopman99]
  - MAFALDA [Arlat99]

Classical robustness testing approach



- Has been mainly applied to operating systems and kernels
  - Ballista [Koopman99]
  - Mafalda [Arlat99]
  - Xception [Carreira98]
  - ...

Classical robustness upgrading approach



Ballista [Koopman99] (1)

- Tool that combines software testing and fault injection techniques
- The main goal is to test software components for robustness
  - Focusing specially operating systems
- Tests use combinations of exceptional and acceptable input values
- Parameter values are extracted randomly from a database of predefined tests
  - For each parameter a set of values of a certain data type is associated

Ballista [Koopman99] (2)

- Robustness is classified using the CRASH scale:
  - Catastrophic
    - OS becomes corrupted or the machine crashes or reboots
  - Restart
    - Application hangs and must be terminated by force
  - Abort
    - Abnormal termination of the application
  - Silent
    - No error is indicated by the OS on an operation that cannot be performed
  - Hindering
    - The error code returned is not correct

MAFALDA [Arlat99]

- Microkernel Assessment by Fault injection AnaLysis and Design Aid
- Allows the characterization of the behavior of microkernels in the presence of faults
- Supports fault injection into the parameters of system calls and into the memory segments
  - In what concerns to robustness testing, only the fault injection into the parameters of system calls is relevant

Questions?

Robustness in WS

Web Services

Web Services

- Strategic component in a wide range of organizations
- Components that can be remotely invoked
  - Well defined interface
- Dynamic, evolving, and supported by unreliable networks
- Deployment implies complex SW infrastructure
- Must be:
  - Robust, Secure, Reliable, Available, …

Goal

- Enable interoperability
- Widespread adoption, ubiquity
- Enable (Internet scale) dynamic binding
- Efficiently support both open (Web) and more constrained environments
- Based on standards
- Focuses on messages and documents
  - Not on APIs

Web Services Framework

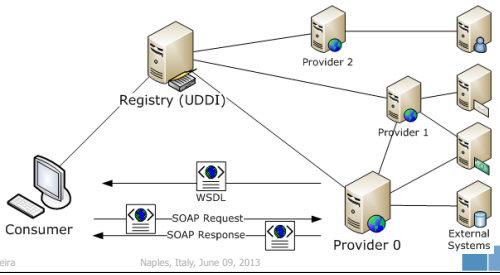
- What goes “on the wire”
  - Formats and protocols (e.g., html)
- What describes what goes on the wire
  - Description languages
- What allows us to find these descriptions
  - Discovery of services

Web Services Description Language

- Interface Description Language (IDL)
- Provides functional description of network services
- Platform independent description
- Extensible language

UDDI

- Universal Description, Discovery and Integration
- Defines the operation of a service registry



Questions?

Robustness in WS

Testing Approach

The problem

Web Services must provide a robust service to the client applications

- Development tools lack mechanisms to:
  - Characterize the robustness of Web Services code
  - Compare the robustness of alternative Web Services

Web Services robustness testing

- Erroneous Web Services call parameters
  - Generated using a set of predefined rules
  - Based on the data types of each parameter
  - Injected during the Web Services execution

GetWeather (city, day) → **GetWeather ("Naples", null)**

- Key components needed:
  - Workload
  - Robustness tests
  - Failure modes classification

The need

An approach to support the evaluation of the robustness of Web Services code

- Useful for programmers and testers:
  - Help providers evaluating the robustness of their Web Services code before deployment
  - Help consumers picking the Web Services that best fit their requirements by comparing different alternatives
  - Help providers and consumers identifying the need for Web Service wrappers

Key components

- Workload
  - Work that the service must perform during the benchmark run
- Robustness tests
  - Faultload consisting of a set of invalid call parameters
  - Applied to the target services to expose robustness problems
- Failure modes classification
  - Characterize the behavior of the service while executing the workload in the presence of the robustness tests

Additional components

- Experimental setup
  - Describes the setup required to run the tests
  - Typically includes two key elements:
    - Tests Target (TT) that represents the service that the benchmark user wants to characterize
    - The Tests management system (TMS) that is in charge of managing all the experiments
  - The TMS performs three key tasks
    - Experiments control
    - Workload emulation
    - Robustness test execution
  - The goal is to make it a completely automated process
- Testing procedure

Testing procedure

- Description of the steps and rules that must be followed during tests execution
- Closely related to the class of services being targeted, but includes:
  - Tests preparation
    - Analysis of the services under testing to gather information
    - Workload generation
  - Tests execution
    - Execution of the workload to understand the service behavior
    - Execution of the robustness tests to trigger faulty behaviors
  - Failure modes classification
    - Robustness problems identification

Workload

- Defines the work that the system must perform during the benchmark execution
- Three different types of workload can be considered:
  - Real workloads
  - Realistic workloads
  - Synthetic workloads

Robustness tests

- Applied during the execution of the workload
- Try to activate robustness issues
- Involves parameter tampering at some level
- A set of rules must be defined for parameter mutation
  - Must focus limit conditions that typically represent difficult validation aspects
    - Which are normally the source of robustness problems

Rules for parameter mutation (1)

- Null and empty values
  - e.g., null string, empty string
- Valid values with special characteristics
  - e.g., nonprintable characters in strings, valid dates by the end of the millennium
- Invalid values with special characteristics
  - e.g., invalid dates with using different formats
- Maximum and minimum valid values in the domain
  - e.g., maximum value valid for the parameter, minimum value valid for the parameter

Rules for parameter mutation (2)

- Values exceeding the maximum and minimum valid values in the domain
  - e.g., maximum value valid for the parameter plus one
- Values that cause data type overflow
  - e.g., add characters to overflow string and replace by maximum number valid for the type plus one
- These rules are similar to the ones used in classical robustness testing
- Each rule is a mutation that is applied at testing time to an incoming parameter

Example of parameters robustness tests

Type	Parameter Mutation
String	Replace by null value
	Replace by empty string
	Replace by predefined string
	Replace by string with nonprintable characters
	Add nonprintable characters to the string
	Replace by alphanumeric string
	Add characters to overflow max size
Number	…
List	…
Date	…
Boolean	…

Failure modes classification

- Services robustness can be classified according to the severity of the exposed failures
- There are several ways to classify failure modes
- An alternative is to use CRASH scale [15] as basis for services characterization
  - The scale must be tailored according to the specificities of the class of services targeted

Tailoring the CRASH scale

- Example:
  - Catastrophic
    - The service supplier (i.e., the underlying middleware) becomes corrupted, or the server or the OS crashes or reboots
  - Restart
    - The service supplier becomes unresponsive and restart must be forced
  - Abort:
    - Abnormal termination when executing the service
    - e.g., unexpected exception is thrown
  - Silent:
    - No error is indicated by the service on an operation that cannot be concluded or is concluded in an abnormal way
  - Hindering:
    - The returned error code is incorrect

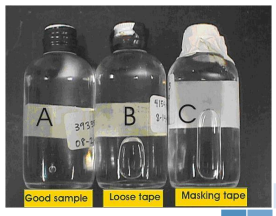
Questions?

Robustness in WS

Case Studies

Case Studies

- [Siblini05]
- [Xu05]
- [Laranjeiro10]
- [Laranjeiro08]



[Siblini05]

- One of the first examples of robustness testing applied to Web Services
- Proposes a technique to test Web Services using parameter mutation analysis
- The WSDL document is parsed and mutation operators are applied to it
  - Resulting in several mutated documents that will be used to test the service
- Very good approach
  - But, the parameter mutation operators are limited!

[Xu05]

- Similar approach
- Represents a more complete study
- Coupling with the XML (eXtensible Markup Language) technology
  - Invalidates any kind of test generalization
  - i.e., it does not apply to other technologies as it is tightly coupled to XML

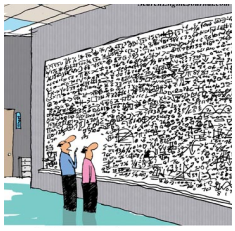
[Laranjeiro10]

- State-of-the art on SOAP Web Services robustness testing
- Key components:
  - Workload, robustness tests, failure modes classification
- Main steps:
  - Preparing the tests
  - Executing the tests
  - Characterizing the Web Services
- Experimental evaluation:
  - 250 publicly available Web Services

The *wsrbench* tool…

- Implements the Web Services testing approach
- Available online

http://wsrbench.dei.uc.pt



Preparing the tests

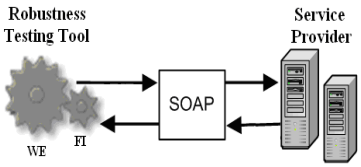
- Obtain Web Service definitions
  - List of operations
  - Parameters
  - Data types
  - Domains

- The WSDL file is processed automatically to obtain the required information
- The domain for each parameter cannot be deduced from the WSDL description
  - Must be provided by the benchmark user

Workload

- A workload is needed to exercise each operation of the Web Service
- A generic workload that fits all Web Services is not feasible
  - We need to generate a workload for each Web Service tested
- Workload generation:
  - User defined workload
  - Random workload

Executing the tests



- Two steps:
  - Step 1: workload is run without considering invalid call parameters
  - Step 2: workload is run in the presence of invalid call parameters (robustness tests)

Parameters values mutation

Type	Parameter Mutation
String	Replace by null value
	Replace by empty string
	Replace by predefined string
	Replace by string with nonprintable characters
	Add nonprintable characters to the string
	Replace by alphanumeric string
	Add characters to overflow max size
Number	…
List	…
Date	…
Boolean	…

Characterizing the Web Services (1)

- wsCRASH scale:
  - Catastrophic: Application server crashes or reboots
  - Restart: Web Service execution hangs
  - Abort: Abnormal termination of the Web Service
  - Silent: After a timeout no error is indicated
  - Hindering: Incorrect error code or delayed response
- Consumers cannot distinguish between a catastrophic and a restart failure
  - Do not have access to the server where the service is running

Characterizing the Web Services (2)

- Simplification for consumers
  - Correct: Good behavior
  - Crash: Abnormal termination of the Web Service
  - Indeterminable: Nothing can be said

Complement the charactization

- Complement the failure modes with the analysis of the Web Service behavior
  - Understand the source of the failures
- e.g. of some sources for robustness problems:
  - Database access operations
  - Arithmetic operations
  - Null References
- Source of the failures depends on the WS
  - No predefined classification

Public Web Services evaluation (1)

- Field study:
  1. Selection of a set of web services to test
  2. Execution of robustness tests with wsrbench
     - http://wsrbench.dei.uc.pt
  3. Identification of robustness problems
- 250 randomly selected public web services
  - 1204 operations; 4085 parameters, 44 country domains; 150 different providers
    - Microsoft, Volvo, Nissan, Amazon, governmental services, banking services, software companies, Internet providers, etc
  - Aprox. 1700 tests per service
    - 420375 responses

Public Web Services evaluation (2)

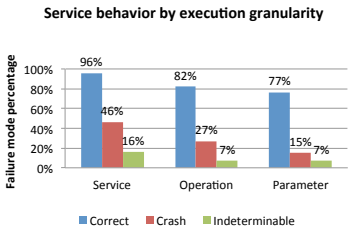
- Three key questions:
  - Can robustness benchmarking be used by providers and consumers to test Web Services?
  - Can robustness benchmarking be used to improve the robustness of Web Services code?
  - Can the benchmark be used to compare different implementations of a given Web Service?

Public Web Services evaluation (3)

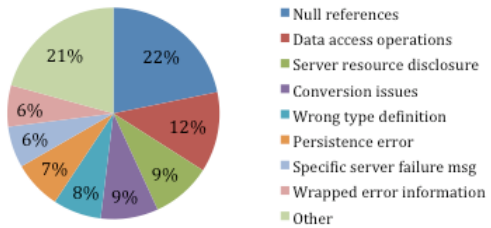
- Web Services are owned by different relevant parties
  - e.g., Microsoft and Xara
- Some Web Services implement the same functionally
  - e.g., Text Disguise and Free Captcha Service
- Some Web Services are used in real businesses in the field
  - e.g., Portuguese Postal Office Orders Cost and UPS Online Tracking Web-Service

Overall results

- 46% of the services had some robustness problem

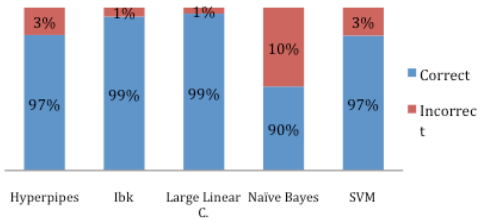


More details…



Automatic classification

- Use well-known text classification algorithms
  - Naive Bayes algorithm
  - SVM classifier
  - Hyperpipes
  - K-nearest Neighboor
  - Large Linear Classification



[Laranjeiro08]

- State-of-the art on JMS robustness testing
- Practical way to evaluate the robustness of JMS middleware
  - Based on a fault injection approach
  - Set of robustness tests (i.e., invalid parameters)
- Help providers evaluate robustness of their implementations
- Help clients to evaluate and select alternative JMS middleware platforms

JMS robustness testing

- Evaluate the robustness of JMS middleware
  - Help providers evaluating their implementations
  - Help clients selecting alternative middleware
- Midleware considered
  - JBoss MQ 4.2.1.GA
    - Latest production ready implementation
    - Ran against Sun Microsystems compliance tests
    - 100% Sun Compatibility Test Suite (CTS) JMS compliant
  - JBoss MQ 3.2.8.SP1
    - Previous major version
  - ActiveMQ 4.1.1
    - Widely used open source implementation
    - Apache Geronimo

Results for JBoss MQ 4.2.1.GA

- JMSMessageID set to null
- Apparently works, but
  - Reading the message raises a null pointer exception
  - Any subsequent reads of that message are unsuccessful
  - Unable to retrieve any valid messages sent afterwards
  - Catastrophic failure
- Recovery
  - Delete directly the invalid message from the database
- Security issues
  - Denial Of Service attacks

Results for JBoss MQ 3.2.8.SP1

- Same results as JBoss 4.2.1.GA
- The same problems in different versions
  - Software is not improving!

Active MQ Results

- Failure observed when testing JMSMessageID
  - Set 1 message to predefined value revealed no problems
  - Set 2 messages with predefined value and send both
  - The first message is never received
  - Silent failure
- Recovery
  - Reboot
- Security issues
  - Denial Of Service attacks by message suppression

Are Web Services robust?

- Faulty Web Services are frequently deployed
  - Unacceptable situation for providers and also for clients
- Robustness problems may lead to security issues
  - Some services are vulnerable to SQL injection attacks
- Robustness testing is essential when developing an infrastructure for Web Services
  - Test and fix Web Services code
  - Select alternative Web Services
  - Build wrappers to mitigate robustness problems

Questions?

Robustness in WS

Fixing/Avoiding Robustness Problems

Fixing/Avoiding Robustness Problems

- Improving the development process
- Conduct code inspections
- Tune the code

Fixing the code…

- A practical way to improve web services robustness
  - Robustness assessment methodology
  - Domain expression language
  - Transparent server-side domain validation
- Very important for creating highly robust services
- Can also be used in legacy services
  - No source code is available
  - Changes applied by using bytecode instrumentation

Robustness improvement approach

1. Accurately describe the Web Service
2. Generate and execute a service workload
3. Execute a set of robustness tests
4. Correct disclosed robustness issues and verify the service behavior

1) Service description

- Inspect service details
  - Operations and parameters (WSDL)
  - Data types (XSD Schema)
- Collect domain information (typically unavailable)
  - Lack of adequate development tools for domain expression
  - XSD Schema inability to describe parameter dependencies

Example: Service domain

- Ex: Operation1 takes 2 integer parameters: A and B
- Valid domain for A: [1, 5] U [6, 10]
- Valid domain for B: [10, 20] U [30,40]
- However, the service requires that:
  - When A is in [1, 5] B must be in [30, 40] and vice-versa

XSD Schema (incomplete syntax)

```
<xs:element name="parameterA">
   <xs:restriction id="r1" base="xs:long">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="5" />
   </xs:restriction>
   <xs:restriction id="r2" base="xs:long">
      <xs:minInclusive value="6" />
      <xs:maxInclusive value="10" />
   </xs:restriction>
</xs:element>

<xs:element name="parameterB">
   <xs:restriction id="r3" base="xs:long">
      <xs:minInclusive value="10" />
      <xs:maxInclusive value="20" />
   </xs:restriction>
   <xs:restriction id="r4" base="xs:long">
      <xs:minInclusive value="30" />
      <xs:maxInclusive value="40" />
   </xs:restriction>
</xs:element>
```

We need to annotate our service as accepting:

r1 and r4

Problem: XSD is unable to express this!

Solution: Use an XSD extension that can express domain dependencies

Extended Domain Expression Language

- EDEL Simple example:

```
...
<dependencies>
  <dependency id="1">
    <function id="f1" name="aggregator" strategy="and">
      <param index="0" name="r1" />
      <param index="0" name="r4" />
    </function>
  </dependency>

  <dependency id="2">
  ...
  </dependency>
</dependencies>
...
```

- Logical OR and XPath functions can be used for complex domains (starts-with, contains, etc)

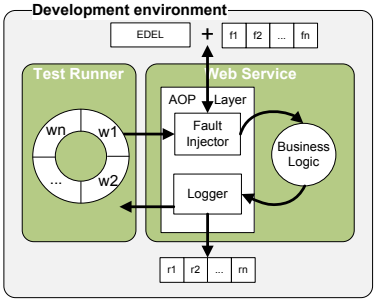
2) Workload generation and execution

- Valid values generated based on the web service description
- Used to exercise the service in "normal" conditions

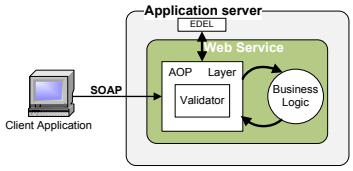
3) Robustness tests execution

- Robustness tests are automatically generated using the services definitions
- Exercise web services operations:
  - Through their publicly available interface
  - Through external services responses (composite services)
- We also perform exception injection
  - All declared exceptions and a set of Runtime exceptions
- At the end we check any response domain violation

Fault injection process



4) Robustness problems removal (1)



- The service is instrumented using AOP (AspectJ)
- Incoming requests undergo a transparent and complete validation
  - According to the announced domain

4) Robustness problems removal (2)

- Well known exceptions are thrown for each invalid request
- Robustness tests are then repeated to check any remaining or new robustness issues
- The original workload is re-executed to check that the service behavior has not been altered

Experimental evaluation

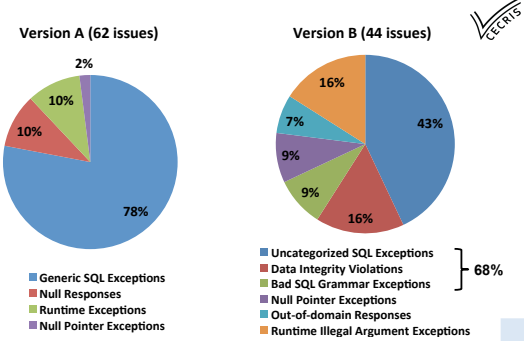
- 3 versions of TPC-App web services
  - A, B, C

1) We analyzed the WSDL and XSD of each service
2) Manually extended each XSD to use EDEL
   - Operations domains were fully defined
3) Created a test workload, and measured its coverage
4) Cobertura indicated +80% general coverage

Robustness issues identification

Web Service	Robustness Problems		
	A	B	C
ChangePaymentMethod	14	13	0
NewCustomer	44	26	0
NewProducts	4	1	0
ProductDetail	0	4	0

Robustness issues classification



Robustness improvement and verification

- We then deployed 3 EDEL-based protected versions of the same web services
- No robustness issues were uncovered!
- We re-ran the workload, and identified no out-of-domain responses
  - Responses were double-checked

Performance impact

- Baseline 30 min tests
- 10000 invocations for each protected service
- Worst case → 16,093 ms ± 0,961 (48%)
- Best case → 7,338 ms ± 0,746 (6%)

Robustness in WS

Research Challenges and Opportunities

Challenges and opportunities (1)

- Fact: Robustness testing seems to be effective when applied to Web Services
- Workload generation
  - Increase representativeness of the workload
  - Guarantee high coverage
- Focus on different types of services
  - Other technologies
- Automated identification of problems
  - In most cases human intervention is needed

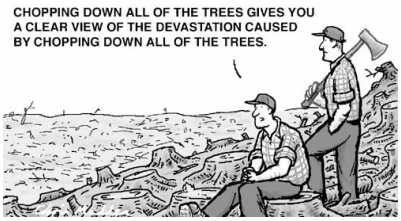
Challenges and opportunities (2)

- Web Services inputs are not only the ones provided during invocation
  - Responses from other services (including databases, gateways, etc) may also be invalid and cause robustness problems
  - How to deal with these?
- Fact: Web Services are typically deployed with robustness problems
- How to improve the current situation?

Challenges and opportunities (3)

- Improve the software development process?
  - More effective testing?
  - More effective code reviews and inspections?
    - Use targeted checklists?
- Provide tools for the automated removal of robustness problems?
  - How to specify input domains?
    - These are not in the WSDL file
- How to take advantage of Web Services diversity to change the current situation?

Questions?



Open Discussion

Your Time!

Key references (1)

- [Arlat99] M. Rodríguez, F. Salles, J.-C. Fabre and J. Arlat, "MAFALDA: Microkernel Assessment by fault injection and design aid", 3rd European Dependable Computing Conference, EDCC-3, September 1999.
- [Carreira98] J. Carreira, H. Madeira and J. G. Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers", IEEE Trans. On Software Engineering, 24 (2), pp.125-36, February 1998.
- [Chillarege92] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. Moebus, B. Ray, M. Wong, "Orthogonal Defect Classification – A Concept for In-Process Measurement", IEEE Trans-actions on Software Engineering, vol. 18, no. 11, pp. 943-956, November 1992.
- [Kalyanakrishnam99] M. Kalyanakrishnam, Z. Kalbarczyk, R. Iyer, "Failure Data Analysis of a LAN of Windows NT Based Computers", Symposium on Reliable Distributed Database Systems, SRDS18, Switzerland, October, 1999.
- [Koopman99] P. Koopman and J. DeVale, "Comparing the Robustness of POSIX Operating Systems", 29th Intl Symposium on Fault-Tolerant Computing, FTCS-29, Madison, WI, USA, pp.30-7, 1999.
- [Lee95] I. Lee and R. K. Iyer, "Software Dependability in the Tandem GUARDIAN System", IEEE Trans. on Software Engineering, vol. 21, no. 5, pp. 455-467, May 1995.

Key references (2)

- [Laranjeiro08] Laranjeiro, N. and Vieira, M. and Madeira, H. , "Experimental Robustness Evaluation of JMS Middleware", IEEE International Conference on Services Computing (SCC 2008), Honolulu, Hawaii, USA, July 2008.
- [Siblini05] Siblini, R., Mansour, N., "Testing Web Services", The 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005.
- [Vieira07] Vieira, M. and Laranjeiro, N. and Madeira, H. , "Benchmarking the Robustness of Web Services", The 13th IEEE Pacific Rim Dependable Computing Conference (PRDC 2007), Melbourne, Victoria, Australia, December 2007.
- [Xu05] Xu, W. et al., "Testing Web Services by XML perturbation", 16th IEEE International Symposium on Software Reliability Engineering, 2005.

Thanks for your participation!

Marco Vieira
Center for Informatics and Systems
University of Coimbra
mvieira@dei.uc.pt