Fault Injection for Web-Services

Marek Rychlý, Martin Žouželka

Department of Information Systems
Faculty of Information Technology
Brno University of Technology
(Czech Republic)

14^{th} International Conference
on Enterprise Information Systems,
28 June – 1 July, 2012

Outline

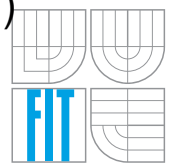
Reliability Testing vs. Robustness Testing

Reliability Testing:

- checking the ability of software to function under given conditions
  (i.e. that the system meets all requirements)
- we need to test, ideally, all possible/valid input states
  (and to compare observed outputs with those expected by the specification)

Robustness Testing:

- checking the ability of software to function correctly in the presence of invalid inputs or stressful environmental conditions
  (i.e. that the system is fault tolerant, resistant, or robust)
- we need to test reactions to invalid input states
  (and to check that the system detects faults correctly, recovers from failures, etc.)



Why Robustness Testing?

1. A software may not be used in accordance with its specification.
   (its future run-time environment can not be fully controlled)
   - it can receive invalid inputs from malfunctioning components
   - it can receive unexpected inputs from its users
   - it can run on a physically damaged/malfunctioning hardware
   - it can be under attack which takes advantage of its vulnerability, etc.

2. A software is provided with limited or without any insight into its extra-functional or safety-critical properties.
   (the trustworthiness of the software can not be guaranteed by its specification)
   - it can be acquired as a “black-box”, provided by third-party
   - it can cause security risk for a whole software system
     (i.e. “security is only as strong as the weakest link”)

Software Implemented Fault Injection

- It is a technique for robustness testing, which artificially corrupts a function of a software component or its input/output data.
- The goal is to observe how the component, its successors in a data-flow, or the whole system behaves after the injection.
- It can be done as
  1. **compile-time injection**
     (by a modification of the component's source code)
  2. **run-time injection**
     (by a modification of the component's run-time environment)

Fault Injection into Web-Services, Motivation

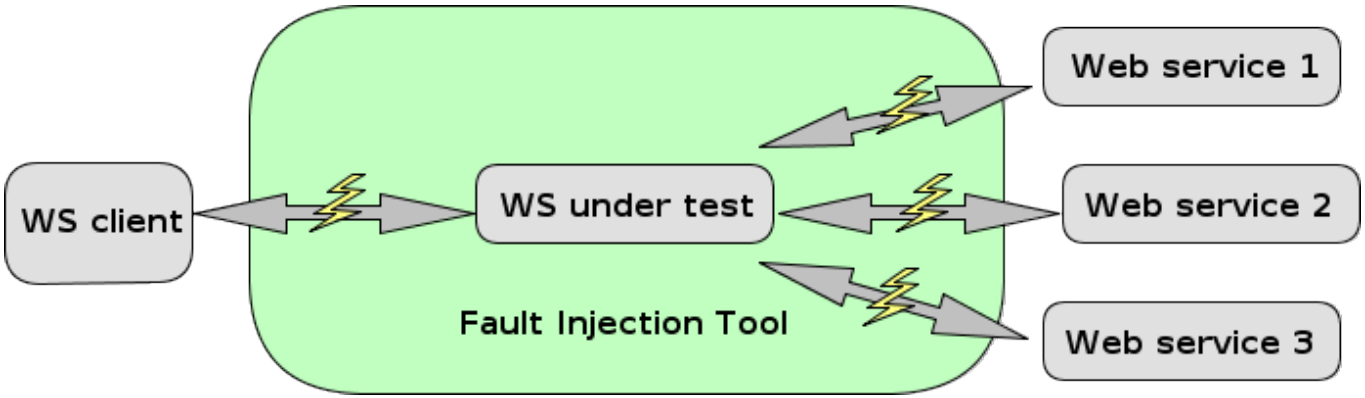
- Web-services (WSs) are designed for reuse.
  (i.e. as software components strictly encapsulated by their interfaces)
- They are typically provided by third-parties as “black-boxes”.
  (e.g. by external information systems, business partners, public authorities, etc.)
- Robustness of WSs can be tested by run-time fault injection.

However. . .

- WSs use various technologies/formats which should be understood.
  (e.g. SOAP/RESTful web-services, XML/JSON/etc. formatted messages)
- Run-time injection can also inject faults into WSs compositions.
  (i.e. not only into input/output data of tested service, but also into its components)
- The above mentioned is not covered by existing approaches.
  (i.e. by existing fault injection tools into WSs: WS-FIT, wsrbench, and WSInject)

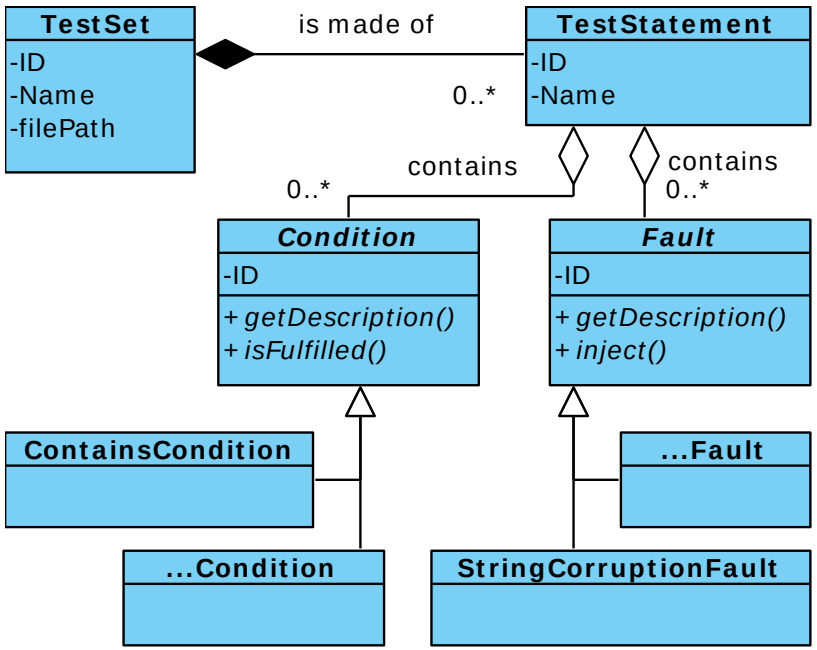
Fault Injection Tool for Web-Services

- The tool acts as a HTTP proxy encapsulating a WS under test.
  (the tool can be integrated into a network, e.g. as a transparent proxy)
- It modifies messages passing between the service and its environment.
  - between the service and its clients
  - between individual services in a service choreography
    (in a composition, between orchestrating and orchestrated services, etc.)

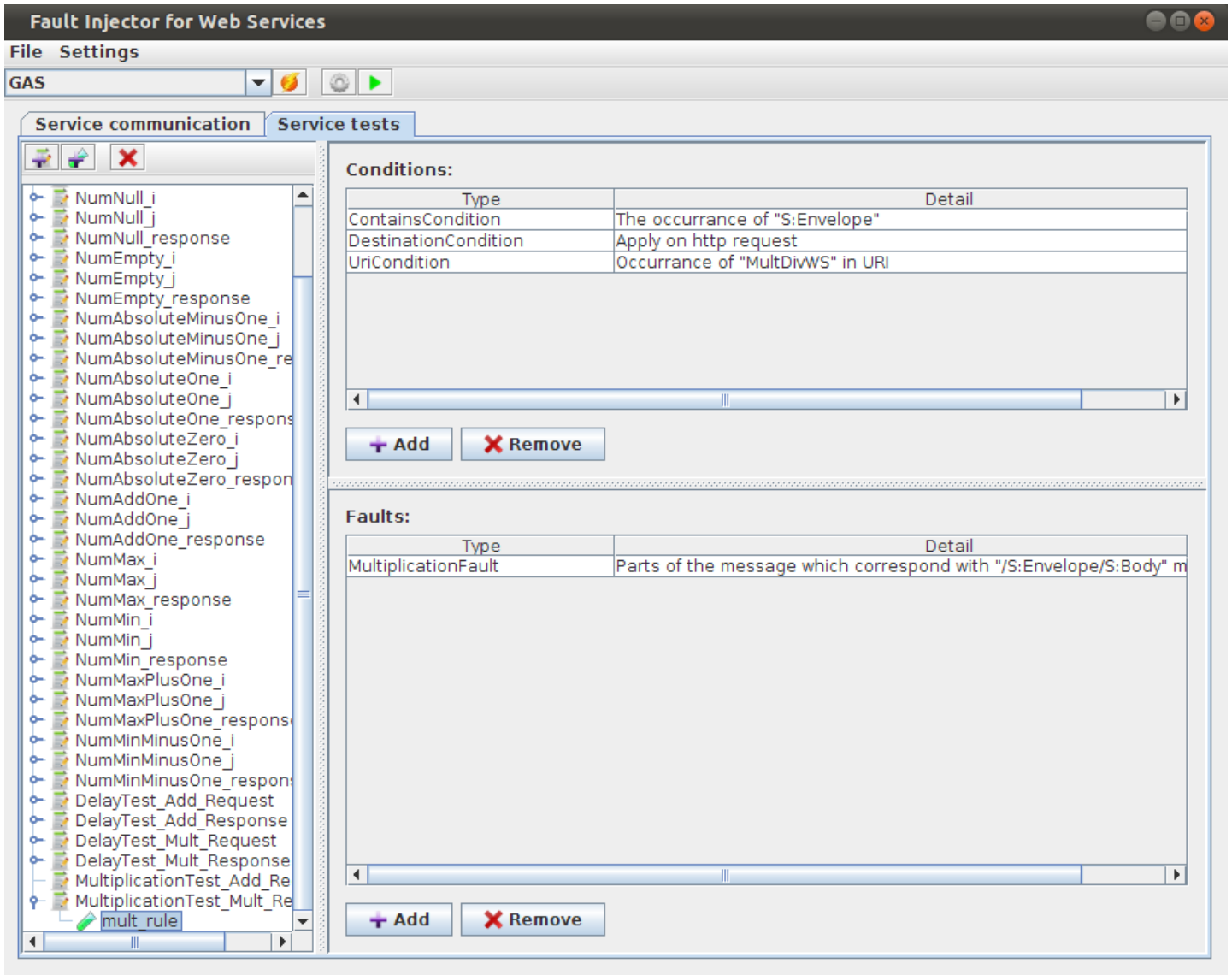


Test Statements Defined by Conditions and Faults

- Test statements are defined by injection conditions and injected faults.
- When a message meets defined conditions of a given test statement, corresponding faults are injected into the message.
- The message can be just modified or can even be delayed or destroyed.

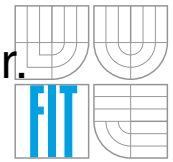


User Interface for Test Statement Specifications

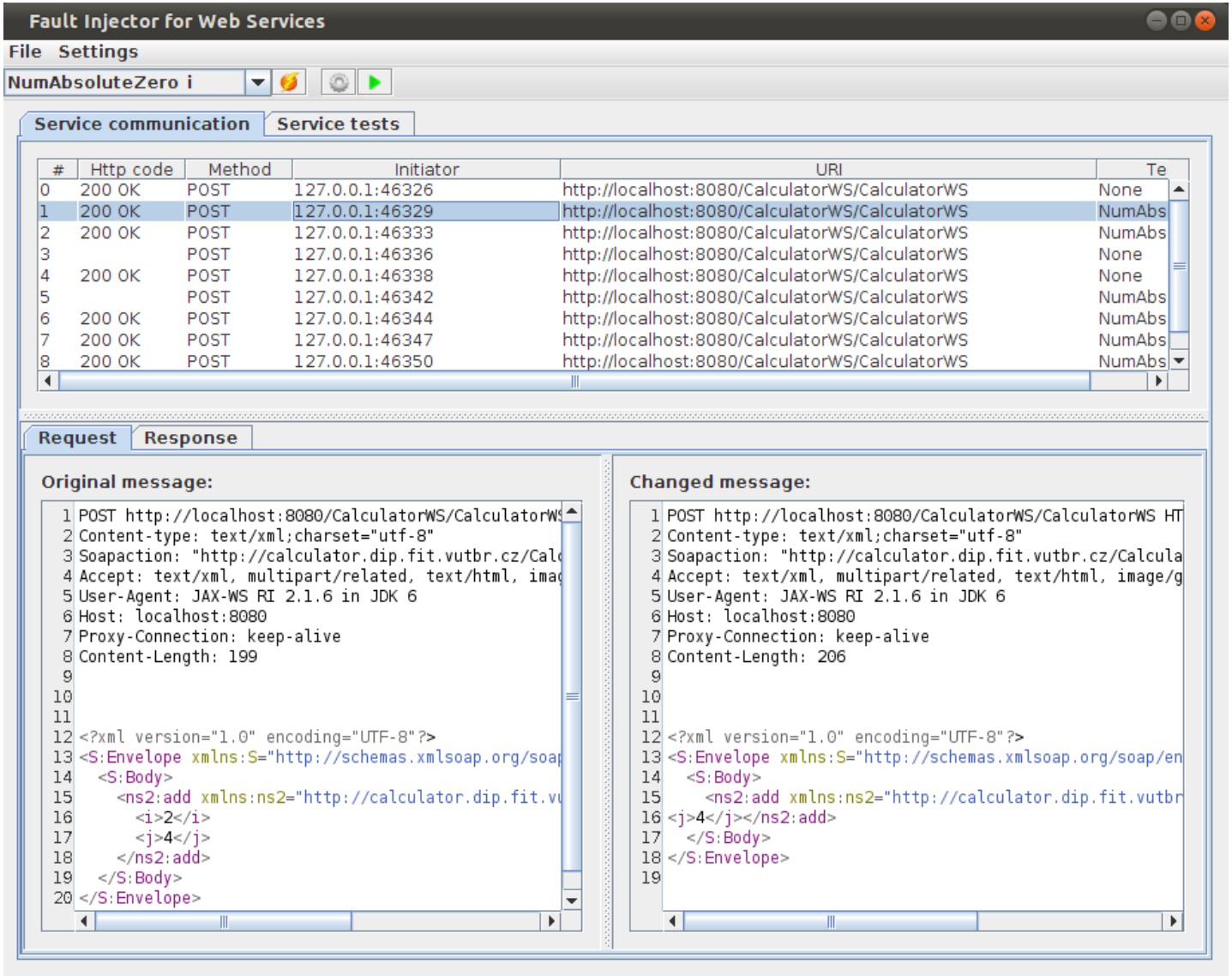


Injection Conditions and Injected Faults

- All HTTP web-services are supported, i.e. SOAP, XML-RPC, and REST.
- HTTP messages can be processed in both directions.
  (i.e. as requests and responses, distinguished by “destination linked condition”)
- Various predefined faults can be injected, e.g. modifying
  - a whole HTTP message or its delivery
    (“Header Corruption”, “Emptying Fault”, and “Delay Fault”)
  - a HTTP message as a raw data
    (“String Corruption” for substitutions regardless of the message’s format)
  - a HTTP message as a XML document
    (“XPath Corruption” and “XPath Multiplication”)
  - a HTTP message as a call/response of a SOAP operation
    (“WSDL Operation Corruption” which uses a given WSDL service description to understand SOAP messages)
- Additional injection conditions and injected faults can be defined by user.
  (as classes implementing “Condition” and “Fault” interfaces)



User Interface for Fault Injections



Results of Experiments Using the Tool

We performed several experiments with robustness testing of

- composite web-services included in Netbeans IDE, with GlassFish v3.2
  (JAX-WS and JAX-RS examples for SOAP-based and RESTful web-services)
- publicly available web-services accessed by their client
  (provided by a third-party, tested without any knowledge of their implementation)

Results:

- SOAP services silently interpreted corrupted input data as default values (e.g. “0”)
- corrupted service calls of a RESTful service accessing a database resulted in inconsistencies in the queried database while no errors were indicated
- calls of the publicly available services mostly resulted in the HTTP 400 and 500 notifications, however, in some cases, unexpected behaviour was detected

Summary and Future Work

- The tool for robustness testing of SOAP, XML-RPC, and RESTful WSs.
  (in service-client communication, service compositions, and choreographies)
- By fault injection into HTTP messages of service calls and responses.
  (a predefined set of injection conditions and injected faults, can be extended)
- We performed testing of sample JAX-WS and JAX-RS web-services and several publicly available web-services provided by third-parties.
  (in the most cases, the services were not fault-tolerant)

Future work

- Integration into IDEs for design and implementation of web-services.

Thank you for your attention!

Marek Rychly <rychly@fit.vutbr.cz>