TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

IBM

Towards Efficient Measuring of Web Services API Coverage

Waldemar Hummer[1], *Orna Raz*[2], *Schahram Dustdar*[1]

[1] *Distributed Systems Group, Vienna University of Technology, Austria*

[2] *IBM Haifa Research Lab, Haifa University Campus, Israel*

3rd International Workshop on Principles of Engineering Service-Oriented Systems (PESOS), co-located with ICSE 2011

Honolulu (Hawaii), USA, May 23-24, 2011
Talk Date: May 23

Overview

- Introduction
  - Motivation, Scenario
  - Approach Overview
- Related Work
- Exact Definition of Web Service API
- Configurable API Coverage Metrics
  - User-Defined Domain Partitions
- Computation of API Coverage
- Evaluation
- Conclusion and Future Work

Motivation

- *Coverage*: extent to which a system has been tested/used
  1) Important field in Software Testing
  2) Used to derive usage statistics
- Web Services (WS) are *Black Boxes*
  - WSDL defines the Application Programming Interface (API)
- API Coverage: #(distinct invocations) / #(possible invocations)
  - Possible WS invocations determined by XML Schema Definitions (XSD)
- Problem: Actual Coverage of 100% Often Infeasible
  1) Main Reason: Subject to *combinatorial explosion*
  2) Influencing Factor: Imprecise XSD types (e.g., `integer` without ranges)
  3) Finally: Not all parameter values/combinations equally important to be tested
- Meaningful API Coverage by Reduction of Value Domains
  - Exact specification of Web services API and parameter types
  - Define which parameter values and combinations are of interest

Related Work

- Test and Analysis of Web Services
  - Vivid research area with various fields (e.g., Baresi et al. [3])
  - Most approaches define coverage criteria and generate test cases
  - Group testing of services [17], Collaborative contract-based testing [2], Execution paths in WS-BPEL processes [10,13], Testing of data-centric service compositions [14, 11], ...
- Interface Based Web Services Testing
  - WSDL and XML Schema based test case generation (e.g. [1,7])
  - Schema *perturbation* to generate invalid requests (Xu et al. [19])
- Domain Partitioning
  - Partition tests into sub-domains (Jorgensen and Whittaker [12])
  - *Category-Partition* test design pattern (Binder [8])
- *Testable* Service (Bertoloni et al. [4])
  - Exposes metadata and aggregated coverage data
  - But: Why not provide coverage calculation as a service?

Approach Overview

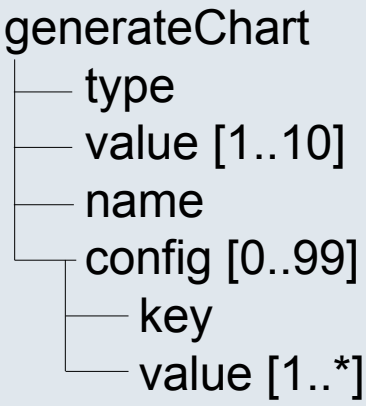
1) Prerequisite: Exact Definition Of Web Service APIs
   - Use of XSD *Facets* (String patterns, min/max values, ...)
   - Framework support for Java Web service (JAX-WS) developers
2) Log and Store Invocation Messages
   - Non-intrusive logging interceptor
   - Messages preprocessed and stored in an optimized format
3) Definition of User-Defined Coverage Metrics
   - Based on Domain Partitioning
   - Customizable, Reusable
4) Computation of API Coverage
   - Based on domain partitions
   - Generation of coverage report
   - Determine service usage statistics

Illustrative Scenario

- *ChartService*: creates charts from numeric inputs
- Parameters of operation *generateChart*:

Name	XSD Type	Constraints
`type`	string	type ∈ { 'line' , 'bar' , 'pie' }
`values`	list of integers	values ∈ $\{-100, ..., 100\}^n$, $1 \leq n \leq 10$
`name`	string	pattern "[a-z][a-z0-9]{0,4}"
`config`	list of key-value pairs	less than 100 entries

- Input Message Structure:



JAXB Schema Generation

- Scenario schema not supported by JAX-WS / JAXB
- 3 new annotations: `@Facets`, `@MinOccurs`, `@MaxOccurs`

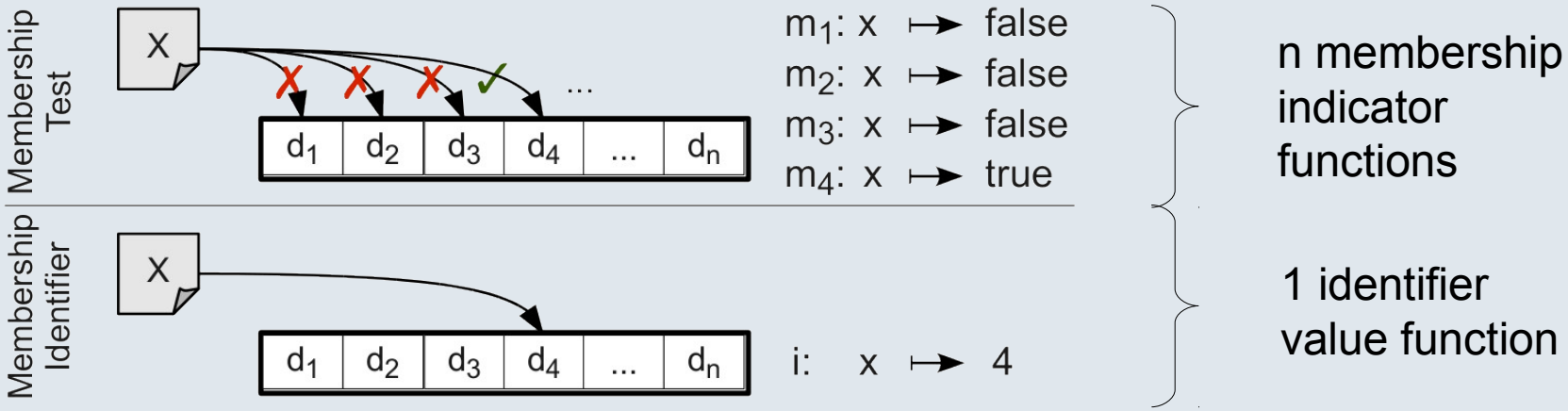
```
@XmlRootElement
public class GenerateChart {
    public static enum ChartType { line, bar, pie }
    public static class Config {
        public String key;
        public List<String> value;
    }
    @XmlElement(required=true)
    public ChartType type;
    @MinOccurs(1) @MaxOccurs(10)
    @Facets(minInclusive=-100, maxInclusive=100)
    public List<Integer> value;
    @Facets(pattern="[a-z][a-z0-9]{0,4}")
    public String name;
    @MaxOccurs(99)
    public List<Config> config;
}
```

JAXB Schema Generation

```
<complexType name="generateChart">
 <sequence>
  <element name="type">
   <simpleType>
    <restriction base="xs:string">
     <enumeration value="line" /> <enumeration value="bar" /> <enumeration value="pie" />
    </restriction>
   </simpleType>
  </element>
  <element name="value" minOccurs="1" maxOccurs="10">
   <simpleType>
    <restriction base="int">
     <minInclusive value="-100" /> <maxInclusive value="100" />
    </restriction>
   </simpleType>
  </element>
  <element name="name">
   <simpleType>
    <restriction base="string"> <pattern value="[a-z][a-z0-9]{0,4}" /> </restriction>
   </simpleType>
  </element>
  <element name="config" type="Config" maxOccurs="99"/>
 </sequence>
</complexType>
<complexType name="Config">...</complexType>
```

API Coverage Metrics (1)

- Basic Idea: Domain Partitioning
  - Split value domain *d* into subsets $P_d = \{d_1, ..., d_n\}$, $d_1 \cup ... \cup d_n = d$
  - Match messages against partitions and determine domain subset
  - Coverage = #(distinct subdomains of logged messages) / $|P_d|$
- Two Partitioning Types



- Example: Partition Integer Into {Negative, Zero, Positive}
  - $m_1(x) := (x<0)$ $m_2(x) := (x==0)$ $m_3(x) := (x > 0)$
  - $i(x) = sign(x)$

API Coverage Metrics (2)

- Example Catalogue of Domain Partitions

ID	Name	Type	$\lvert P_d \rvert$	$m_1(x)$	$m_2(x)$	$m_3(x)$	$i(x)$
i	*ignore*	MT	1	`true`	-	-	-
n	*negZeroPos*	MT	3	`x<0`	`x==0`	`x>0`	-
z	*zero*	MT	1	`x==0`	-	-	-
t	*extreme*	MT	2	`x==MIN`	`x==MAX`	-	-
b	*blocksOf10*	MI	`int(abs(MAX -MIN)/10)+1`	-	-	-	`(int) x/10`
o	*outOfRange*	MT	1	`x<MIN|| x>MAX`	-	-	-
p	*pieChart*	MT	1	`x=='pie'`	-	-	-
d	*default*	predefined, based on XSD of the Web service					

Additional predefined membership indicator function m_0 embraces elements that belong to no other subset:

$$m_0(x) = (\bigwedge_{i \in \{1,..,|P_d|\}} \neg m_i(x))$$

API Coverage Metrics (3)

- Apply Partitioning to Logged Messages
  - Based on the message structure in the XSD
  - User selects **two partitions** for each node in the XSD
    - P_v : partition for the **value** (i.e., text content) of this node
    - P_o : partition for the number of **occurrences** of this node
  - Message gets mapped to domain memberships of all its nodes
- Some Examples (partition IDs refer back to last slide)

	default		ignore all		chart types		zero-config pie chart	
	P_v	P_o	P_v	P_o	P_v	P_o	P_v	P_o
generateChart	-	-	-	-	-	-	-	-
type	d	-	i	-	d	-	p	-
value	d	d	i	i	i	i	i	i
name	d	-	i	-	i	-	i	-
config	-	d	-	i	-	i	-	z
key	d	-	i	-	i	-	i	-
value	d	d	i	i	i	i	i	i
Possibilities:	(as in XSD)		1		3		2	

API Coverage Metrics (4)

- 2 Sample Messages

<generateChart>	Partition Subset		<generateChart>
<type>bar</type>	2	3	<type>pie</type>
<value>10</value>	3	3	<value>34</value>
<value>25</value>	3	3	<value>12</value>
<value>–13</value>	1	3	<value>29</value>
<value>36</value>	3	3	<value>25</value>
<value>0</value>	2	1	<name>ch2</name>
<value>–13</value>	1	2	<config>
<value>12</value>	3	1	<key>zoom</key>
<name>ch1</name>	1	1	<value>1.5</value>
<config>	1		</config><config>
<key>zoom</key>	1	1	<key>titles</key>
<value>1.2</value>	1	1	<value>x=foo</value>
</config>		1	<value>y=bar</value>
</generateChart>			</config>
			</generateChart>

Selected Partitions:

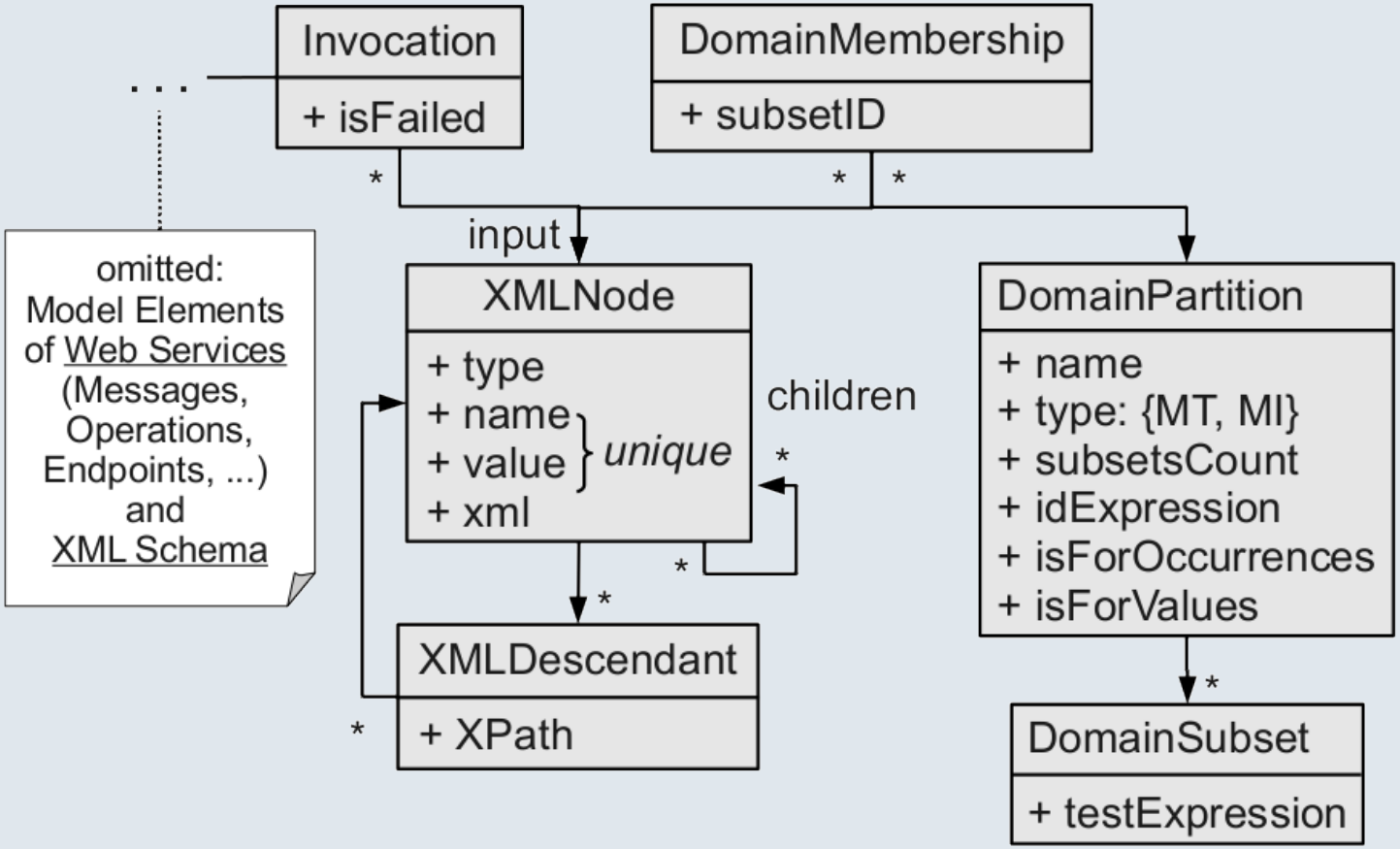
	P_v	P_o
gen.Chart	-	-
type	d	-
value	n	i
name	i	-
config	-	d
key	i	-
value	i	i

Partition Hash Vectors:

2	**7**	3	3	1	3	2	1	3	1	**1**	1	1	**1**	null	null	null
3	**4**	3	3	3	3	null	null	null	1	**2**	1	1	**3**	1	1	1
type	values								name	c.	c.keys		c.values			

Coverage Computation (1)

- Data Preparation of Logged Messages
  - Parse the XML tree and store all nodes and relationships
  - Determine the partition memberships of all nodes



Coverage Computation (2)

- We need: #(possible invocations) and #(distinct invocations)
- Data Stored in a Relational Database
- Queries Performed Using SQL
  - Goal: select distinct partition hash vectors
  - Number of columns depends on message schema
  - → SQL string generated on demand
- Query String for Our Scenario
  - Each "(…)" represents a sub-query
  - Computationally intensive (DBMS needs to perform many joins)

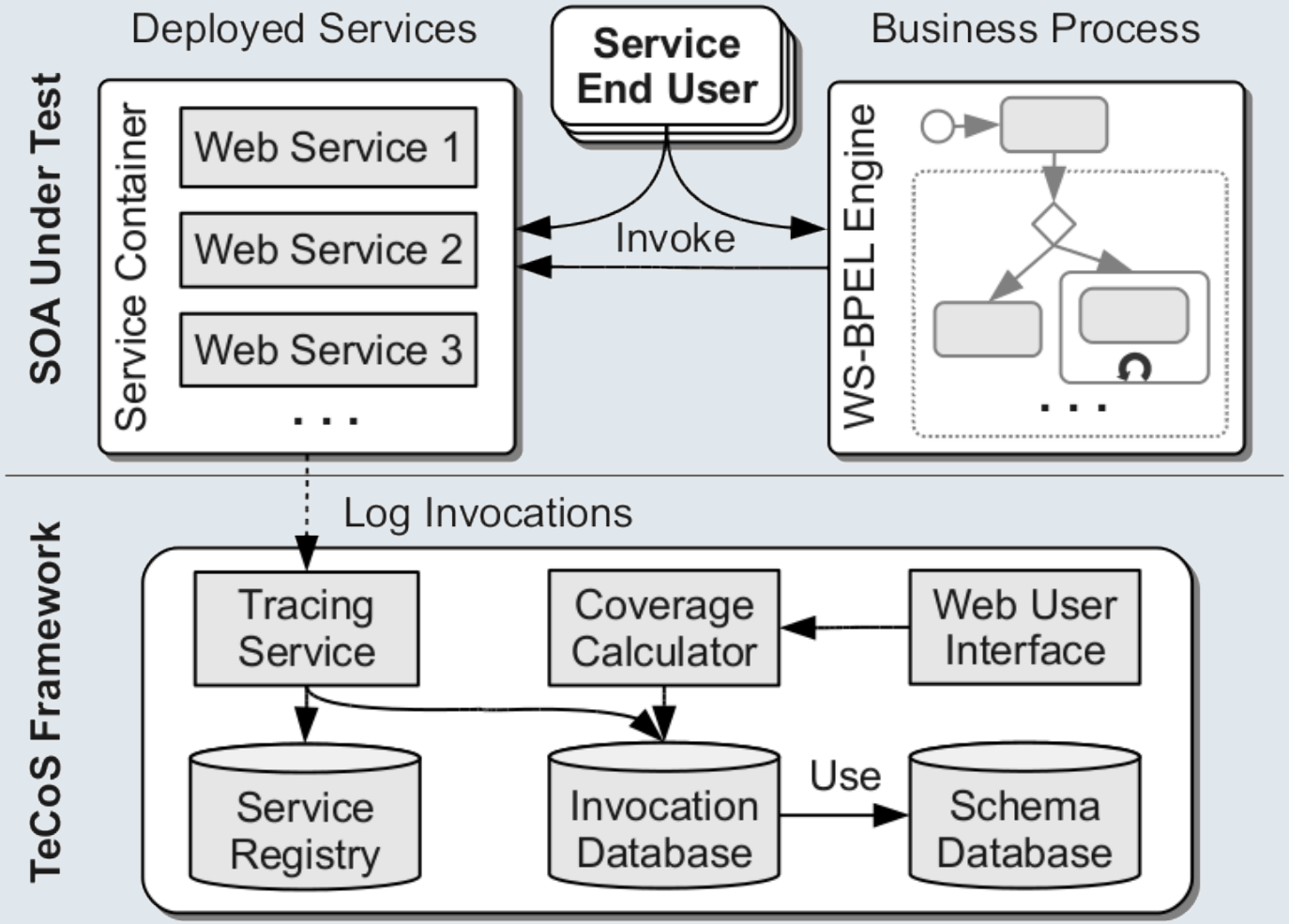
select distinct (…) as **gt**, (...) as **gv_o**, (…) as **gv_1**, (…) as **gv_2**, ..., (…) as **gv_7**, (…) as **gn**, (…) as **gc_o**, (…) as **gck_1**, (…) as **gck_2**, (…) as **gcv_o**, (…) as **gcv_1**, (…) as **gcv_2**, (…) as **gcv_3** from ...

SQL Result

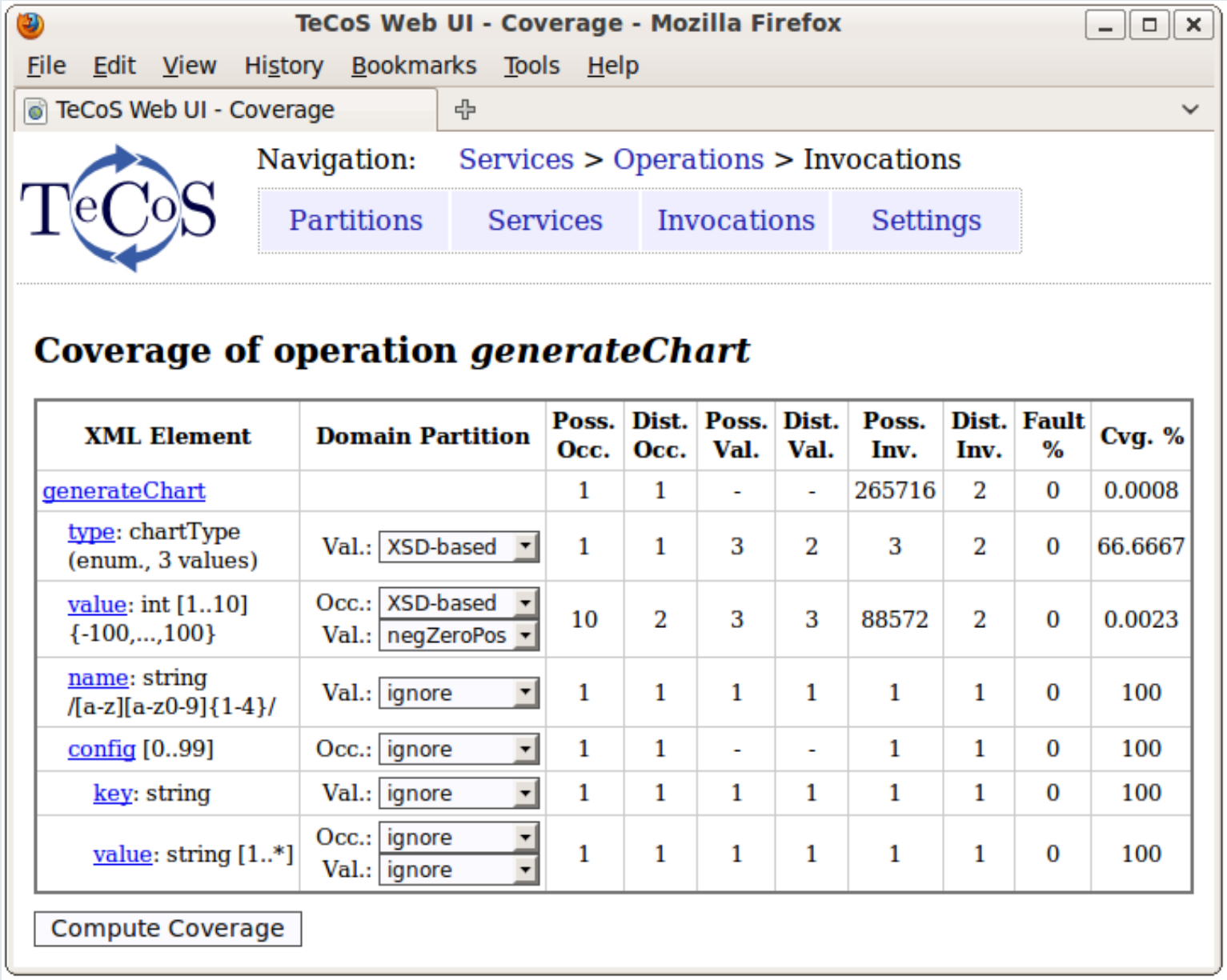
2	7	3	3	1	3	2	1	3	1	1	1	1	1	null	null	null
3	4	3	3	3	3	null	null	null	1	2	1	1	3	1	1	1

Implementation (1)

- *TeCoS* framework [11] (Test Coverage for Service-based systems)

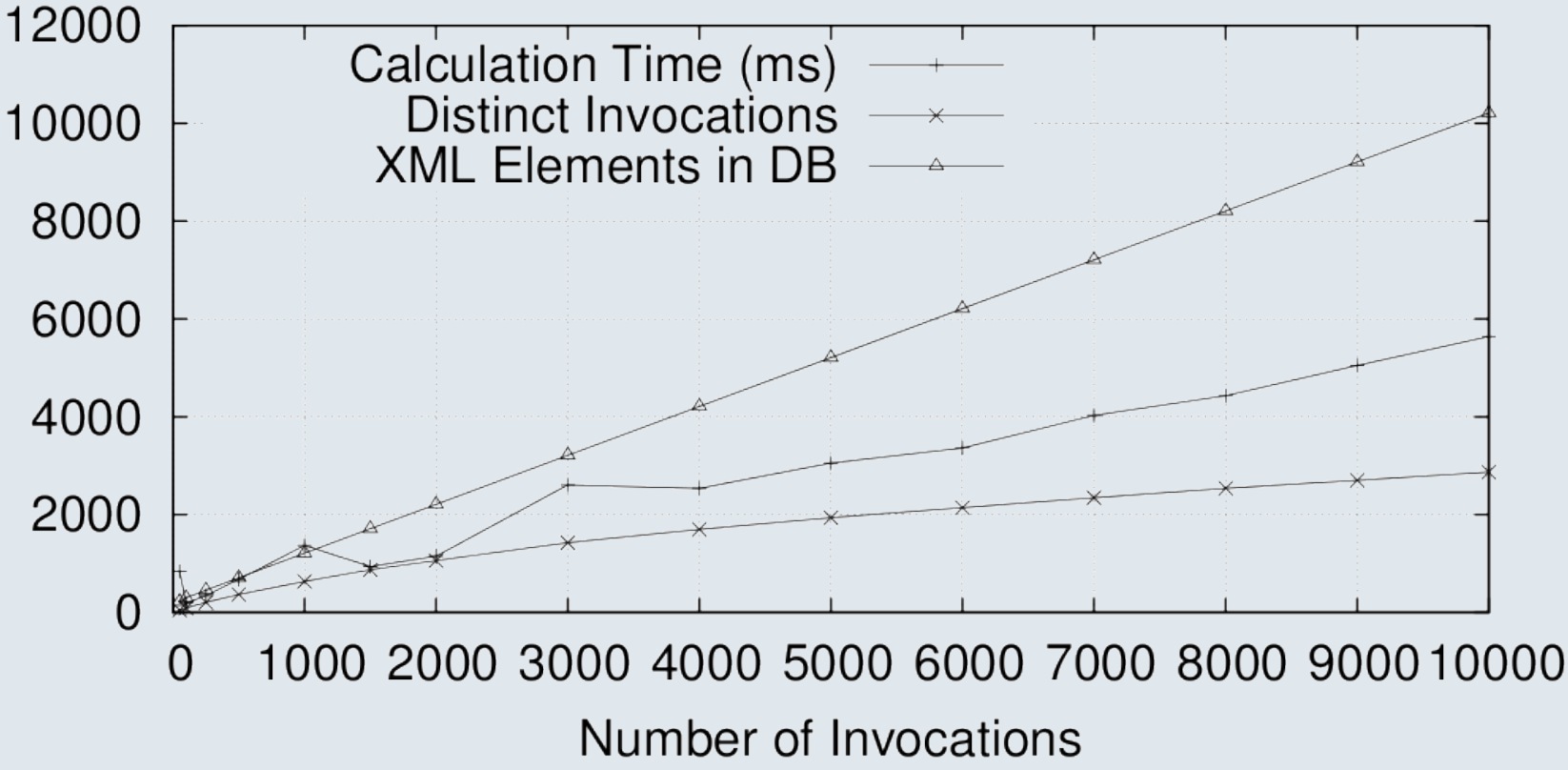


Implementation (2)



Evaluation (1)

- Experimentation Based on Scenario Charting Service
  - Generated 10.000 invocations with randomized test data
  - Cvg. Calculation in real time (< 6 seconds for 10.000 invocations)
  - Larger scenarios require distributed storage and computation



Evaluation (2)

- Wide Range of Possible Metrics
  - Starting point: ***i*** (ignore) and ***d*** (default) partitions
  - Refine the metric using custom partitions

#	Example Metrics	v(gt)	o(gv)	v(gv)	v(gn)	o(gc)	v(gck)	o(gcv)	v(gcv)
		User-Specified Partitions							
1	Usage Counter	d	d	d	d	d	d	d	d
2	Extreme Values	i	t	t	i	i	i	i	i
3	Chart Types	d	i	i	i	i	i	i	i
4	Blocks of 10	i	i	b	i	i	i	i	i
5	Pie Charts	p	d	d	d	d	d	d	d
6	Out of Range	i	i	o	i	i	i	i	i
7	Config. Settings	i	i	i	i	d	i	i	i
8	Config. Keys	i	i	i	i	i	d	i	i
9	Values of Default Pie Charts	p	i	d	i	z	i	i	i

Conclusion

- API Coverage
  - Extent to which a service has been used/tested
  - Important measure for testing and for usage statistics
- Model for API Coverage Metrics
  - Based on domain partitioning
  - Applied to values and occurrences of XML nodes
- Important Precondition: Exact Interface Definition
  - Proposed extension for XSD facets support in JAXB
- Future Work
  - Extend the scope of API Coverage to invocation sequences
  - Distributed storage and computation of coverage metrics
    - Tailor-Made Implementation and „Divide and Conquer“
    - Apache Hadoop / HBase / Hive
  - Privacy and trust concerns

Discussion

References (1)

- [1] X. Bai, W. Dong, W.-T. Tsai, and Y. Chen. WSDL-based automatic test case generation for Web services testing. In Int. Workshop Service-Oriented Syst. Eng., pages 215–220, 2005.
- [2] X. Bai, Y. Wang, G. Dai, W.-T. Tsai, and Y. Chen. A framework for contract-based collaborative verification and validation of web services. In CBSE ’10, pages 258–273, 2007.
- [3] L. Baresi and E. D. Nitto. Test and Analysis of Web Services. Springer-Verlag New York, Inc., 2007.
- [4] C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti. Whitening SOA Testing. In ESEC/SIGSOFT FSE ’09, 2009.
- [5] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini. WS-TAXI: A WSDL-based Testing Tool for Web Services. In ICST 2009, pages 326–335, 2009.
- [6] B. Beizer. Software testing techniques (2nd ed.). Van Nostrand Reinhold Co., New York, USA, 1990.
- [7] A. Bertolino, J. Gao, E. Marchetti, and A. Polini. Automatic Test Data Generation for XML Schema-based Partition Testing. In Int. Workshop Automation of Software Test, 2007.
- [8] R. V. Binder. Testing object-oriented systems: models, patterns, and tools. Addison-Wesley Longman, 1999.
- [9] G. Canfora and M. Di Penta. Testing Services and Service-Centric Systems: Challenges and Opportunities. IT Professional, 8(2):10–17, 2006.

References (2)

- [10] J. Garcïa-fanjul, J. Tuya, and C. D. L. Riva. Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN. In WS-MaTe 2006, pages 83–94, 2006.
- [11] W. Hummer, O. Raz, O. Shehory, P. Leitner, and S. Dustdar. Test coverage of data-centric dynamic compositions in service-based systems. In 4th IEEE International Conference on Software Testing, Verification and Validation, 2011.
- [12] A. Jorgensen and J. Whittaker. An API Testing Method. In STAREAST Conf. on Softw. Testing Analysis & Review, 2000.
- [13] D. Lübke, L. Singer, and A. Salnikow. Calculating BPEL Test Coverage Through Instrumentation. In Int. Workshop on Automation of Software Test, pages 115–122, 2009.
- [14] L. Mei, W. Chan, and T. Tse. Data flow testing of service-oriented workflow applications. In ICSE, 2008.
- [15] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. Computer, 40(11):38–45, 2007.
- [16] J. O. Paul Ammann. Introduction to Software Testing. Cambridge University Press, 2008.
- [17] W. T. Tsai, Y. Chen, R. Paul, N. Liao, and H. Huang. Cooperative and Group Testing in Verification of Dynamic Composite Web Services. In COMPSAC, pages 170–173, 2004.
- [18] World Wide Web Consortium (W3C). XML Path Language (XPath). http://www.w3.org/TR/xpath/, 1999.
- [19] W. Xu, J. Offutt, and J. Luo. Testing Web Services by XML Perturbation. In 16th IEEE Int. Symposium on Software Reliability Engineering, pages 257–266, 2005.