At the TAS$^3$ review on February 26, 2009, the project reviewers indicated that deliverable D10.1: Trustworthiness – State of the Art was rejected. The TAS$^3$ Consortium has addressed the reviewers comments as follows.

**Foreword**

The TAS$^3$ projects includes Workpackages specifically devoted to tackle security, data protection, trust management, authentication and authorization. In addition to those “traditional” concerns when dealing with trust, the TAS$^3$ consortium also intends to take due care of dependability, quality and usability concerns, because we are deeply convinced that such qualities also constitute an important ingredient of a system perceived trust. This is what was planned in the DOW, and is in line with the definition provided in the ICT Work Programme 2009-10 of the European Commission for a trustworthy system$^1$: secure, reliable and resilient to attacks and operational failures; guaranteeing quality of service; protecting user data; ensuring privacy and providing usable and trusted tools to support the user in security management (this is reported in the revised D10.1 page 12). In fact, trying to make such a vision clear from the outset, the WP title refers to our enlarged vision of trust with the term trustworthiness.

The contribution of WP10 has to be read as complementary to a more specialized management of trust. In view of the comments received, we fear that the scope of WP10 might have been misinterpreted at the February review, also due to the fact that the WP has not been directly presented at the meeting, and several deliverables crucial to get a comprehensive view of TAS$^3$ trust were not yet due (e.g. D5.1).

Concerning Deliverable D10.1 (due M9), as its title says, it is not yet meant to present new results, but it is still a survey work. Precisely, the idea was not to deliver a general survey of trust (the state of art for TAS$^3$ has been already surveyed in D1.1). Instead the goal of this deliverable was to recollect existing work, both from others and from WP10 partners, in the scope of WP10 objectives: functional testing, QoS, usability. In addition to surveying the state of art we wanted also to present in detail our earlier methodologies and tools that in our expertise could be usefully exploited and adapted to TAS$^3$ needs (but such adaptation/improvement work was not yet presented).

Following the reviewers negative feedbacks, the structure and scope of the Deliverable has now been radically changed.

---

While the basic objectives and scope of WP10 remain, in the revised D10.1 we have on the one hand considerably reduced the description of our previous methodologies and tools (the chapters that earlier made the second part of D10.1 have here been reduced to one section), while on the other hand we have broadened the survey also to topics and methods that in the project will be tackled in other WPs. Precisely, the specific responses to each comment are provided below.

**Detailed Responses**

Below reviewers comments are verbatim reported in bold, and related responses are inlined.

**Comments relative to D10.1 (page 10 of First Technical Review Report):**

a) **Need a broader introduction to the concept of trust.**
   We have included two new chapters: Chapter 2 introduces trust and trustworthiness and clarifies the scope of WP10. Chapter 3 deals with trust and reputation from a user-centric (subjective) perspective. We also refer to deliverable D5.1 which is entirely devoted to trust management.

   **What about KPI, how they will be linked? Should you include any reputation methods?**
   Answer is positive to both, of course TAS3 includes KPIs and reputation-based methods. As explained in Chapter 3, both are addressed in WP5 and their design is discussed in D5.1.

b) **The document describes functional testing of an infrastructure solution,**
   As explained in the foreword, this was indeed one of the intended goals of this document but completely lacks any security testing (e.g., negative tests, fuzz tests, security error handling, XML specific attacks).

   A totally new chapter, namely Chapter 4, has been included surveying security testing in extensive way.

c) **Security issues of service orchestration and service routing are not addressed, including denial of service.**
   Now included in Chapter 4, see Sections 4.1 and 4.1.1.

d) **The document layout is different.**
   We understand formality may have its weight. Nevertheless, this issue was beyond our reach: D10.1 has been released in September and the standard layout that deliverables due M12 adopted has been agreed upon in December. We now follow the standard layout as well.

Comment on page 14: “Beneficiaries seem to bring their existing work into the project rather than delivering new results towards a common goal.

Some examples include:

... ...

WP10 has delivered results (D10.1 'Trustworthiness State of the art') that are not directly related to the TAS3 project or aligned with a common vision.”
With all due respect, we totally disagree with the latter comment. Functional testing, usability, quality of service are related and important to the TAS3 project that tackles business processes of interacting services, and contrariwise to reviewers statement, we are convinced that NOT considering them since upfront would be symptom of short vision: it would be a failure for the project to release a completely secure system that is not usable and thus will not be adopted; or it would be hopeless to try to build trust if the system fails in operation: it is obvious that operational unreliability negatively affects reputation.

Concerning the point “Beneficiaries seem to bring their existing work into the project rather than delivering new results towards a common goal”, as above explained D10.1 was not meant to deliver new results, not yet. Partners contributing to WP10, in particular CN-R/ISTI and University of Saragoza, bring to the consortium different backgrounds, software engineering and usability, respectively, and some time must be allowed before the respective contribution can be usefully integrated with those of the other partners. In the months elapsed after the review lot of progress has already been done, in particular in collaboration with WP2, WP5, WP8 and WP9, which will be documented in D10.2.

The WP10 Team
Document type  Deliverable
Title  D10.1- Trustworthiness – State of the Art
Work Package  WP10
Deliverable Number  D10.1
Editor(s)  Antonia Bertolino, CNR/ISTI
Dissemination level  Public
Preparation date  1 June 2009
Version  1.5

Legal Notice
All information included in this document is subject to change without notice. The Members of the TAS3 Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the TAS3 Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.
The TAS3 Consortium

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Participant name</th>
<th>Country</th>
<th>Participant short name</th>
<th>Participant role</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>K.U. Leuven</td>
<td>BE</td>
<td>KUL</td>
<td>Coordinator</td>
</tr>
<tr>
<td>2</td>
<td>Synergetics</td>
<td>BE</td>
<td>SYN</td>
<td>Project Manager</td>
</tr>
<tr>
<td>3</td>
<td>University of Kent</td>
<td>UK</td>
<td>KENT</td>
<td>Partner</td>
</tr>
<tr>
<td>4</td>
<td>University of Karlsruhe</td>
<td>BE</td>
<td>KARL</td>
<td>Partner</td>
</tr>
<tr>
<td>5</td>
<td>Technische Universität Eindhoven</td>
<td>NL</td>
<td>TUE</td>
<td>Partner</td>
</tr>
<tr>
<td>6</td>
<td>CNR/ISTI</td>
<td>IT</td>
<td>CNR</td>
<td>Partner</td>
</tr>
<tr>
<td>7</td>
<td>University of Koblenz-Landau</td>
<td>DE</td>
<td>UNIKOLD</td>
<td>Partner</td>
</tr>
<tr>
<td>8</td>
<td>Vrije Universiteit Brussel</td>
<td>BE</td>
<td>VUB</td>
<td>Partner</td>
</tr>
<tr>
<td>9</td>
<td>University of Zaragoza</td>
<td>ES</td>
<td>UNIZAR</td>
<td>Partner</td>
</tr>
<tr>
<td>10</td>
<td>University of Nottingham</td>
<td>UK</td>
<td>NOT</td>
<td>Partner</td>
</tr>
<tr>
<td>11</td>
<td>SAP Research</td>
<td>DE</td>
<td>SAP</td>
<td>Partner</td>
</tr>
<tr>
<td>12</td>
<td>Eifel</td>
<td>FR</td>
<td>EIF</td>
<td>Partner</td>
</tr>
<tr>
<td>13</td>
<td>Intalio</td>
<td>FR</td>
<td>INT</td>
<td>Partner</td>
</tr>
<tr>
<td>14</td>
<td>Risaris</td>
<td>IR</td>
<td>RIS</td>
<td>Partner</td>
</tr>
<tr>
<td>15</td>
<td>Kenteq</td>
<td>BE</td>
<td>KETQ</td>
<td>Partner</td>
</tr>
<tr>
<td>16</td>
<td>Oracle</td>
<td>UK</td>
<td>ORACLE</td>
<td>Partner</td>
</tr>
<tr>
<td>17</td>
<td>Custodix</td>
<td>BE</td>
<td>CUS</td>
<td>Partner</td>
</tr>
<tr>
<td>18</td>
<td>Medisoft</td>
<td>NL</td>
<td>MEDI</td>
<td>Partner</td>
</tr>
</tbody>
</table>

Contributors

<table>
<thead>
<tr>
<th>Name</th>
<th>Organisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Antonia Bertolino</td>
</tr>
<tr>
<td>2</td>
<td>Guglielmo De Angelis</td>
</tr>
<tr>
<td>3</td>
<td>Francesca Lonetti</td>
</tr>
<tr>
<td>4</td>
<td>Eda Marchetti</td>
</tr>
<tr>
<td>5</td>
<td>Andrea Polini</td>
</tr>
<tr>
<td>6</td>
<td>Antonino Sabetta</td>
</tr>
<tr>
<td>7</td>
<td>Carlos Flavian</td>
</tr>
<tr>
<td>8</td>
<td>Miguel Guinaliu</td>
</tr>
<tr>
<td>9</td>
<td>Raquel Gurrea</td>
</tr>
<tr>
<td>10</td>
<td>Luis Casalo</td>
</tr>
<tr>
<td>11</td>
<td>Jeroen Hoppenbrouwers</td>
</tr>
</tbody>
</table>
Table of Contents

LIST OF ACRONYMS ........................................................................................................ 6

LIST OF FIGURES ............................................................................................................. 9

1 EXECUTIVE SUMMARY ............................................................................................... 10
  1.1 ROADMAP ................................................................................................................. 11

2 TRUSTWORTHINESS OF SERVICE-ORIENTED SYSTEMS ......................................... 12
  2.1 CLASSICAL TRUST CLASSES ..................................................................................... 12
  2.2 TRUSTWORTHINESS IN TAS\(^3\) .................................................................................. 13
  2.3 THE TWO DIMENSIONS OF TRUSTWORTHINESS ...................................................... 14
  2.4 WP10 AND THIS DELIVERABLE .................................................................................... 15

3 TRUST AND REPUTATION METHODS ......................................................................... 16
  3.1 DEFINITION OF TRUST ............................................................................................. 16
  3.2 TYPOLOGY OF TRUST ............................................................................................... 17
  3.3 TRUST AND ELECTRONIC SERVICES ....................................................................... 18
  3.4 REPUTATION METHODS ........................................................................................... 19
  3.5 REPUTATION METHODS IN TAS\(^3\) .......................................................................... 21

4 SECURITY TESTING ....................................................................................................... 22
  4.1 WEB SERVICES SECURITY CHALLENGES ................................................................ 22
    4.1.1 Attack Patterns ..................................................................................................... 24
    4.1.2 XML security verification ...................................................................................... 26
  4.2 APPROACHES FOR SECURITY TESTING .................................................................. 27
    4.2.1 Negative testing .................................................................................................... 29
    4.2.2 Fuzz Testing .......................................................................................................... 30
    4.2.3 Information Gathering ........................................................................................... 31
    4.2.4 Injection ................................................................................................................ 32
    4.2.5 Confidentiality & Integrity ..................................................................................... 32
    4.2.6 Web Services Security Extensions ........................................................................ 33
    4.2.7 Logging .................................................................................................................. 34
4.2.8 Authentication and authorization 34
4.3 SECURITY TESTING IN TAS\textsuperscript{3} 36

5 FUNCTIONAL TESTING ................................................................. 37
5.1 APPROACHES TO OFF-LINE WS TESTING 37
5.2 TESTING BY ENHANCING THE DIRECTORY AND DISCOVERY SERVICE 38
  5.2.1 The Audition Framework 39
5.3 RUNTIME MONITORING 42
  5.3.1 Logical Architectures for Monitoring 42
  5.3.2 Some Industrial Proposals 43
5.4 FUNCTIONAL TESTING IN TAS\textsuperscript{3} 44

6 QUALITY OF SERVICE TESTING AND ASSESSMENT ......................... 45
6.1 QoS TESTING 45
  6.1.1 Puppet 47
6.2 END-USER PERCEIVED SERVICE QUALITY 48
  6.2.1 Perceived Service Quality Measurement 49
6.3 QoS IN TAS\textsuperscript{3} 50

7 USABILITY TESTING ................................................................. 52
7.1 UNIQUE ASPECTS OF HCI AND USABILITY IN THE PRIVACY AND SECURITY DOMAIN 53
7.2 USABILITY TESTING TOOLS 53
7.3 USABILITY TESTING IN TAS\textsuperscript{3} 55

8 XML-BASED TESTING ................................................................. 56
8.1 TESTING OF XML-BASED APPLICATIONS 56
  8.1.1 TAXI tool 57
8.2 WEB SERVICES SECURITY POLICY TESTING 59
8.3 XML-BASED APPROACHES IN TAS\textsuperscript{3} 60

BIBLIOGRAPHY ............................................................................. 63
List of Acronyms

AMNESIA  Analysis and Monitoring for NEutralizing SQL Injection Attacks
AOP  Aspect-Oriented Programming
API  Application Programming Interface
BGP  Border Gateway Protocol
BPEL  Business Process Execution Language
BPEL4WS  Business Process Execution Language for Web Services
CB  Component Based
CP  Category Partition
DOM  Document Object Model
DTD  Document Type Definition
ECP  Equivalence Class Partitioning
EP  Evaluated Performance
e-SQ  Electronic Service Quality
FTC  Federal Trade Commission
HCI  Human Computer Interaction
HTTP  Hypertext Transfer Protocol
ICT  Information and Communication Technologies
jUDDI  Java Universal Description, Discovery, and Integration
KAOS  Knowledge Acquisition in autOmated Specification
KPI  Key Performance Indicators
LDAP  Lightweight Directory Access Protocol
NQ  Norman Quality
OASIS  Organization for the Advancement of Structured Information Standards
OCT  On-line Compliance Testing
OrBAC  Organization Based Access Control
P2P  Peer-to-Peer
PDP  Policy Decision Point
PeSQ  Perceived e-Service Quality
PLASTIC  Providing Lightweight & Adaptable Service Technology for pervasive Information & Communication
Promela  Process Meta Language
PUPPET  Pick UP Performance Evaluation Test-bed
PUT  Process Under Test
QOS  Quality of Service
RBAC  Role-Based Access Control
SAFELI  Static Analysis Framework for discovering SQL Injection vulnerabilities
SAML  Security Assertion Markup Language
SAX  Simple API for XML
SLA  Service Level Agreement
SLS  Service Level Specification
SOA  Service-Oriented Architecture
SOAP  Simple Object Access Protocol
SQC  Schema Quality Checker
SQL  Structured Query Language
SSL  Secure Sockets Layer
SSM  Service State Machine
STS  Symbolic Transition System
SUT  System Under Test
TAFT  Tag-Aware text file Fuzz testing Tool
TAW  Test Accessibility Web
TAXI  Testing by Automatically generated XML Instances
TCTL  Timed Computation Tree Logic
TDS  Testing Driver Service
TP  Test Process
TLS  Transport Layer Security
TSS  Test Strategy Selection
TTP  Trusted Third Party
UDDI  Universal Description Discovery and Integration
URI  Uniform Resource Identifier
WebSSARI  Web application Security by Static Analysis and Runtime Inspection
WS  Web Service
WS-Col  Web Service Constraint Language
WSDL  Web Services Description Language
WS-GUARD  WS-Guaranteeing Uddi Audition at Registration and Discovery
WSLA  Web Service Level Agreement
WSS  Web Services Security
WSSSMS  Web Services Security: SOAP Message Security
XACML  eXtensible Access Control Markup Language
XML  Extensible Markup Language
XPath  XML Path Language
XPT  XML-based Partition Testing
XSA  XML Schema Analysis
XSD  XML Schema Definition
XSV  XML Schema Validator
List of Figures

Figure 2.1: Trustor and Trustee relationship ............................................................... 13
Figure 2.2: Trust purposes and TAS\textsuperscript{3} WPs ............................................. 14
Figure 4.1: Transport Level: The whole message is encrypted ............................... 23
Figure 4.2: Message Level: Only the content of the message is encrypted .......... 24
Figure 5.1: The Audition Framework ................................................................. 40
Figure 6.1: General idea of PUPPET ................................................................. 47
Figure 8.1: TAXI interface for test strategy selection ........................................... 58
1 Executive Summary

This document is the first deliverable of TAS\textsuperscript{3} WP10: Quality Measures and Trustworthiness. Trustworthiness of networked and service-oriented systems is a very broad concept that is addressed in the various TAS\textsuperscript{3} WPs from different perspectives.

**WP10 scope and objectives.** Within WP10 we intend to focus on important system qualities that are not traditionally considered in trust literature, but nonetheless also contribute to the overall trust perception: we will contribute to the TAS\textsuperscript{3} infrastructure and governance with tools and mechanisms to verify the compliance of TAS\textsuperscript{3} services to functional and QoS specifications and to measure user-perceived usability and quality. We recall the WP10 objectives, as stated in the DoW:

- Validating the functionality of the TAS offered services;
- Validating the usability of the TAS architecture;
- Evaluating agreed QoS parameters;
- Analyzing end-user perceived QoS.

The above objectives pose several outstanding research challenges in each specific subject field. Moreover, altogether they share the TAS\textsuperscript{3} innovative goals of dealing with such complementary views of trust in a holistic way, and in harmony with the more traditional “handles” of security.

**This deliverable.** The objectives of this deliverable are twofold: to clarify the positioning of WP10 with respect to TAS\textsuperscript{3} trust provision; and to provide a survey of challenges and existing tools and technologies for the assessment of the trustworthiness dimensions addressed in WP10. We also point at some solutions that at current stage we have identified as potentially useful to address the above WP objectives. In reading this deliverable, it is important to consider that:

- the project has already delivered a comprehensive state-of-art survey in Deliverable D1.1, to which we refer for getting a full exposition to TAS\textsuperscript{3} background. This deliverable is meant to complement D1.1 with deeper study of WP10 topics.
- This deliverable has been extensively revised after the first TAS\textsuperscript{3} review. For completeness, and to fulfill the explicit reviewers’ requests, the deliverable content has been widened beyond the actual focus of WP10 to also include a broader overview of trust and reputation methods and a survey of security issues and testing. However, properly the place where trust management and reputation methods, including TAS\textsuperscript{3} targeted Key Performance Indicators (KPIs), will be addressed in TAS\textsuperscript{3} is not WP10 but WP5: Trust Policy Management. Security testing, as shown in Chapter 4, is very wide and a complete coverage of all its facets is outside the scope of WP10, for obvious reasons of resource limitations. In future deliverables, we will tackle anyhow some negative and fuzz testing, and will develop test suites from XACML policies (ongoing work, see hints at Chapter 8).
1.1 Roadmap

The deliverable is structured as follows. In the next Chapter we provide a broad introduction to trust and trustworthiness, and clarify WP10 scope. Chapter 3 overviews further Trust, in particular considering reputation methods. Target applications of TAS\(^3\) are those involving management of personal data, and hence security remains certainly a major concern. Chapter 4 overviews security challenges and surveys security testing. Then, in Chapter 5 we consider functional testing of services, which is relevant in TAS\(^3\) because we need to check trustworthiness of business processes. Chapter 4 overviews evaluation and testing of Quality of service both from a technical/objective and a user-perceived/subjective perspective. Chapter 7 explores approaches to usability testing: since in TAS\(^3\) the user is at the center, having systems that are usable increases users confidence and perceived trust. Finally, Chapter 8 delves on deeper technological level: as the service paradigm is based on standard exchange notations, such as XML, we exploit this format to automatize test suite generation (to be exploited for functional and security testing).
2 Trustworthiness of Service-oriented Systems

Information and Communication Technology (ICT) is today increasingly pervasive, and the wave of service-oriented computing is at the hearth of modern business and social endeavors. On the one side services support the vision of dynamic, adaptive and transparent access to any desired resource and of unlimited capabilities of communication, fostering new models of trading and networking. A leading principle of service-orientation is that service consumers do not own the service and therefore need not be concerned with all the aspects generally associated with ownership (such as infrastructure, technology, integration and maintenance) [1]. We only have to choose the service which best meets our business needs. On the other side, however, under this business model our personal and societal welfare become increasingly vulnerable, because the ease of use and dynamism of service technology inevitably lead to the establishment of more and more complex and large-scale networks in which critical information is stored and manipulated online.

As a consequence, a key exigency for networks and service platforms is to be made “trustworthy”, which in the ICT Work Programme 2009-10 of the European Commission is defined as\(^1\): secure, reliable and resilient to attacks and operational failures; guaranteeing quality of service; protecting user data; ensuring privacy and providing usable and trusted tools to support the user in security management.

All the above listed properties describe systems that merit the trust of their users: in fact, literally the meaning of trustworthy is “worthy of confidence”. We emphasize here the subtle but significant difference between the two terms: “trust” and “trustworthiness”. Although strictly related, they are not synonyms. A useful way to explain the difference can be to consider the mutual trustor-trustee relationship that is established between the consumer and the provider of a service, respectively (see Figure 2.1). The former term, trust, looks to this relationship from the trusting party viewpoint, and carries along inevitably a subjective flavor: trust eventually involves a decision from the trustor to take the risk of depending on the trustee, even though negative consequences are possible. Trustworthiness, instead, looks closely to the trusted party side, and concerns the evidences brought forward to deserve being trusted: we might say that trustworthiness is shown, trust is given (because of trustworthiness, or even despite lack of trustworthiness).

As discussed in [2], trustworthiness of a system is relevant throughout its life cycle, from conception to requirements to detailed architectural designs to implementation, and then needs to be continuously pursued into use, operation, maintenance, and recycled through revisions as appropriate. More importantly, trustworthiness is very difficult to achieve unless it is systematically integrated throughout the development cycle.

The project TAS\(^3\) aims at providing a trustworthy architecture and governance framework for personal data and will do so by providing/integrating different possible types of evidences.

2.1 Classical trust classes

In general the very notion of trust may cover quite different interpretations, depending on the context to which it is applied and the domain in which it is considered (this is detailed in the

next chapter). In the huge literature on trust, various classifications have been attempted. In their recent survey [3], Jøsang et al. report a list of five trust classes (elaborated from [4]), based on the purpose for which the trustor needs to put trust on the trustee. We summarise shortly this list below (see also Figure 2.2):

- Provision, or business: the trustor relies on services provided by the trustee under some (contract) agreement;
- Access: the trustor needs access to resources owned or managed by the trustee;
- Delegation: the trustee, or also “delegate” in this case, acts and behaves on behalf of the trustor;
- Identity: the identity of the trustee is as claimed;
- Context: this refers to the environment and infrastructure within which the trustee operates.

2.2 Trustworthiness in TAS³

Each of the above concerns for trust is relevant in TAS³. A key innovation of the project is the goal to enable stakeholders to perceive trust in a highly distributed and dynamic service-oriented system that allows users to manage their personal information in a secure, trustworthy and privacy-friendly infrastructure, and service providers to effectively process the information and resources under controllable data protection and privacy mechanisms. This vision forms the basic fundament of the pursued holistic users’ trust perception in TAS³. Appropriate verification, enforcement and compliance checking procedures need to be put in place to gain and maintain trust.

Given the 360 degrees perspective from which the project wants to address trust, the consortium is taking a very comprehensive attitude and no relevant aspect of trust is being a priori excluded. Therefore, the project is addressing trust in all its constituents aspects, relying on different types of evidences and measures, covering different ranges of objectivity or subjectivity. WP12 is the place where integration is kept under control and facilitated.
Figure 2.2: Trust purposes and TAS³ WPs

Considering the list of Jøsang et al. given in Section 2.1, since TAS³ addresses the trust problem in a holistic way, provision trust which is a very broad notion is further subdivided into several constituents concerns. In particular, we focus on compliance trust, Quality of Service trust and usability trust, which we qualify as follows:

- **Compliance**: trustor relies on the trustee to comply with agreed behaviour or policy specifications, or a given business model;
- **QoS**: the trustee guarantees specified or agreed parameters of Quality of Service (e.g., latency, reliability, ...);
- **Usability**: trustor can easily and effectively access provided services.

Figure 2.2 illustrates the introduced TAS³ concerns, and links them to those project WPs where these are mainly addressed. Of course this is a simplified overview, as several of the concerns are transversal, and vice versa some of the WPs, such as WP2, are relevant to all classes. As shown, WP10 is the place in TAS³ where the three identified sub-concerns for provision trust are centrally addressed.

### 2.3 The two dimensions of trustworthiness

The subject of Trustworthiness in TAS³ can also be classified from a different perspective between a technical and a social side. Traditionally they are not approached in the same way.

Technically, trustworthiness is about formal compliance, reliability, verification of authentic sources, authorization control, and other known techniques that come from both dependabil-
ity and the traditional security areas. Socially, trustworthiness is about trusting the people or organisations behind the technology to keep their promises.

WP10 approaches both sides. Concerning Compliance, the project aims to certify new services that want to participate to TAS\textsuperscript{3} choreographies, and service providers that apply to become members of the (future) TAS\textsuperscript{3} secure business ecosystem. This certification process tries to provide objective guarantees that both the services and the organization of the new partner are up to the requirements for trustworthy business. Some technical checks can be made off-line using the preproduction version of the new partner’s system, by exposing it to predefined test cases and various carefully designed situations that exploit edge cases and other known weak spots. Some social and business checks need access to the new partner’s business processes, where technology cannot help, and human auditors need to visit the company and investigate from the inside.

When the new partner is accepted and goes into production, its certification must be regularly rechecked, and live monitoring also needs to take place to measure other forms of conformance that the TAS\textsuperscript{3} network needs to know.

Finally, non-functional aspects of service behaviour, or Quality of Service as they are commonly called can again be divided into technical and social/business sides. Next to the obvious low-level performance qualities that usually are covered in Service Level Agreements, statistics about the amount of transactions that completes or fails due to security problems are interesting to know. Inherent in TAS\textsuperscript{3} is a trustworthiness exam, the results of which are kept by the system and may be used by businesses to (pre)select partners before the transaction is initiated.

## 2.4 WP10 and this deliverable

This deliverable is meant to present the State of the Art in trustworthiness for Service-oriented system, more closely considering the above mentioned specific concerns: compliance, QoS and usability. Indeed, even only considering such aspects, the literature is immense and no survey can be considered exhaustive. The approach we have taken here has been to direct the survey to those technologies that we believe should be prioritized for further investigation in this WP, because they are usually neglected in projects addressing security and trust. The combined holistic approach to trustworthiness is a key innovation of TAS\textsuperscript{3}.

For the sake of completeness we have also included here some concerns and technologies that are not directly the focus of WP10, or that will not be addressed in the life of TAS\textsuperscript{3} because of time and resources limitations, but nevertheless are topical to the project scope. A note to keep in mind in reading D10.1 is that this goes coupled with deliverable D1.1, in which the TAS\textsuperscript{3} consortium has provided a comprehensive state-of-art survey spanning over all WPs. This deliverable does not substitute that one, but rather complements and further refines D1.1 concerning the scope of WP10.
3 Trust and Reputation Methods

Trust has a clear influence on the adoption of new communication technologies. Trust is a very important element in the relations between the human beings and computers, since it increases the levels of the individual's confidence, comfort and satisfaction. In this chapter we discuss on the different views on trust from a user-oriented point-of-view. In particular, we overview different trust definitions and typologies, and reputation methods. We recall that technical concerns for trust management and reputation methods are addressed in WP5 (see Deliverable D5.1: Trust Management Architecture Design).

3.1 Definition of Trust

There is widespread consensus with regard to the importance of trust in the creation, development and maintenance of lasting relationships (e.g. [5] [6]). Disciplines such as Sociology, Psychology, Political Sciences, Economics, Anthropology, History and Sociobiology have dealt with this concept in great depth [7]. However, this consensus does not exist when it comes to establish a single definition which unites the perspectives of the different disciplines which have studied the concept [8].

This divergence of opinions has made it more difficult for the research community to understand the term, while the resulting arguments and discussions among authors from different fields of research have generally involved each side attempting to impose its ideas on the rest [9]. However, this is a pointless debate. Each discipline needs to approach the subject of trust from a different perspective. It is not possible for an economist to consider the term from the same point of view as an engineer, at least not completely, and therefore it is advisable for the researcher to clarify right from the beginning the proposed definition so as to eliminate doubts about any conclusions which may be drawn from the study. Thus, authors cannot be accused of making mistakes in their initial hypotheses as a result of having considered mistaken starting points. Each researcher can therefore consider the term in a different way but without making such mistakes.

The nature of trust, as already hinted at in the previous chapter, is undoubtedly complex, “it has many meanings” ([10], p. 453), but this does not mean that discussion of the subject should be limited to the definition of the term, especially when researchers from so many different fields are involved in these discussions, as it is highly unlikely that they will ever completely agree with each other. Consensus would only be recommendable within each discipline, since otherwise it will be very difficult for knowledge to be efficiently transferred between researchers. Trust should be analysed in terms of both its causes and its effects, and its antecedents and consequences for the behaviour of individuals must be identified. This is definitely the analysis which provides the best results.

An exploration of the possibilities of consensus in the definition of the term reveals that only a few parallels exist in certain aspects [11]. In this respect, the presence of risk should be highlighted. Thus, it is understood that trust arises in situations in which one of the parties - the one that trusts - finds itself in a vulnerable situation with respect to the actions of the other - the trusted one. Gambetta ([12], p. 216) establishes that trust is “a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before s/he can monitor such action (or inde-
pendently of his capacity ever to be able to monitor it) and in a context in which it affects his own action”. Vulnerability is therefore irremediably linked to the need for trust, and therefore in relationships in which the participants - or at least one of them - are not at risk from the opportunist actions of the others, trust will not appear because it is not necessary.

Likewise, the emergence of trust is typical of situations of uncertainty in which it is not possible to delimit all the contingencies which may occur, and therefore it is necessary to develop mechanisms which compensate for this lack of information and allow relational and exchange processes to take place. In this respect, “if we were blessed with an unlimited computational ability to map out all possible contingencies in enforceable contracts, trust would not be a problem” ([12], p. 216). The impossibility of anticipating all the possible events which may occur during the course of relationship, as well as the desire to reduce the costs of transaction derived from complete contracts, demands alternative and efficient mechanisms such as trust.

### 3.2 Typology of trust

Numerous and varied efforts have been made by researchers to establish a typology of trust. In Chapter 2 we have proposed a classification based on the purpose of trust. Another useful criterion can be the type of agents involved. Thus we have interpersonal trust, interorganizational trust, company-final user trust, citizen-government trust, etc.; trust can also be regarded as the capacity of fulfilment (credibility) or good faith (benevolence) [13]; another alternative is to consider it as a rational process [9] or a subjective process [14], among other possibilities. One interesting classification is that proposed by McKnight, Cummings and Chervany ([15], pp. 6-14), who distinguish up to five types of categories of trust: (1) Interpersonal Trust, (2) Trusting Beliefs, (3) System Trust, (4) Dispositional Trust (5) Decision to Trust. Of the five possibilities, the most interesting in accordance with the aims of TAS³ is the “System Trust”.

System trust is, according to McKnight, Cummings, and Chervany ([15], p. 12), “the belief that proper impersonal structures are in place to enable one to anticipate a successful future endeavour. Impersonal structures include such safeguards as regulations, guarantees, or contracts. System trust may also refer to belief in the proper structure of one’s own role and others’ roles in the situation”. In short, system trust is characterised by [6]:

- **Situational Normality**: this implies that the context in which the relationship is going to be carried out presents a situation of normality. The authors offer the following example: “a person who enters a bank tends to expect a setting conducive to both customer service and fiduciary responsibility that is reflected in the workers’ professional appearance, the prosperous and secure physical setting, and the friendly, yet safe, money-handling procedures”. According to this view, a situation of abnormality in the environment would produce a feeling of mistrust in the individual which could alter the behaviour. In fact it could generate emotional insecurity, this being an extremely important factor;

- **Structural Assurance**: this structure is made up of (1) legal regulations which protect the fulfilment of the obligations undertaken by the parties; (2) guarantees or assurances which compensate the defrauded party and (3), social frameworks which promote fulfilment. In short, system trust, in relation to this “structural assurance”,
consists of contracts and promises. These promises - which correspond to alternative (3) - require two conditions. The first is that the promise must be significant enough for the party that makes the promise and has incentives to fulfil it due to fear of possible punishments. The second condition is that, regardless of the existence of possible punishments, the ways of customers of the social environment encourage fulfilment.

The authors of [16] consider trust from a broad perspective as an environment of trust which affects the relationship between satisfaction and loyalty. In this respect, the authors identify three components in this environment, one of which is defined by the macroeconomic, political, social and technological characteristics within which the individual carries out the relationships (Political, Economic, Social, and Technological Environment). This political environment is characterised, according to [16], by organizations such as the Federal Trade Commission (FTC, United States). These institutions supervise the commercial transactions which are carried out in the market and help to rectify the asymmetries of information which occur between the parties involved in the operations. These organisations influence the perceptions of individuals in two ways. On the one hand they generate a climate of calm (normality) and, on the other, they facilitate decision-making. In TAS$^3$ architecture the role mentioned above is represented by the Trust Guarantor, the Governing entity of a TAS$^3$ environment. The top level Trusted Third Party administers the Trust Network.

### 3.3 Trust and electronic services

The low levels of trust existing in many real and potential Internet users act as a decisive obstacle for the growth of e-services such as e-government. Therefore, trust is a critical factor in the stimulation of Internet purchase.

In connection with the interaction human being-technology, some authors are sceptical about the possibility that technology may become an object to rely on, basically because trust seems to have sense only in interpersonal relationships. However, some researches show how individuals tend to perceive in technological systems personal qualities, behaving with them in the same way as they behave with other individuals ([17] [18]). This fact is very relevant since it supports the theories which try to apply the results from those contextualised researches to interpersonal relationships in new media, in which technologies act as a fundamental intermediary. From the perspective of trust creation, a dis-embedding process exists. This means that interpersonal trust is gradually being replaced with trust in abstract systems and institutions (trust in control instruments). As we cannot have a rich relationship with an individual, we choose to trust external institutions in the expectation that they will control those invisible and frequently anonymous entities at the other end of digital communication link. According to these arguments TAS$^3$ must be perceived as a partner, not as a black machine. TAS$^3$ should have an own personality represented by a reliable brand. People will think of TAS$^3$ in terms what honest and sincere is.

For some authors, trust and computer-mediated environments are more a matter of attitude than technology. Thus, the problem is not the lack of reliable and safe technologies, but the individual’s attitude and prejudices to the new channel, especially the aspects of the system’s security, the effectiveness of applicable laws and the differences with traditional channels. The generation of trust in computer-mediated environments faces the challenge of an impersonal environment, characterised by the distance between the parties. Trust is
fundamental in electronic relational exchanges, since they are characterised by uncertainty, anonymity, loss of control and a potential opportunism from the parties involved. Aspects like the existence of vulnerability, dependence and uncertainty, which demand the generation of trust to obtain lasting relationships, seem to be present in electronic exchanges, since in these exchanges the user has no physical interaction with the other part. Besides, in computer-mediated environments, payment is usually prior to the goods and services delivery, which makes the vendor’s honesty, competence and benevolence specially important. On the whole, computer-mediated environments are perceived as riskier operations than those offline, mainly due to an insufficient legal framework and the distance between parties. The virtuality, together with occasional announcements of attacks from hackers who steal credit cards and the appearing of viruses, a general ignorance of available encrypting systems and the present debate on the protection of the individual’s privacy, are all critical factors when generating trust [19]. All in all, the development of trust on computer-mediated environments is more difficult than in the traditional media since very often both parties are in different geographical areas or there is no previous record of the relationships between both. Again another critical question for TAS$^3$’s success. Formative and communication activities will be essential in order to TAS$^3$ is perceived as a trustworthiness system in terms of usefulness, security and usability.

The study of trust in computer-mediated environments can be classified according to three different scenarios:

- Technology to Technology interactions: autonomous agents (whether hardware-based, software-based, mobile or otherwise) that have their behavior programmed. Developments in autonomous agents suppose a philosophical challenge due to we can replicate our social relationships inside the technology domain;

- Human to Technology: in general, confidence in technology comes at two different levels. The first is the consideration regarding the immediate interface between human and technology (usable systems). Finally, confidence in technical systems can be seen as an interaction of privacy, security and reliability;

- Technology to Human: this is an area that has several direct links to security, in the form of authentication, authorization, identification, access control, etc. It refers to different methods by which the device can gain certain confidence about humans (e.g. passwords, fingerprints, smart cards, methods to recognize human emotional states).

The studies on digital trust have dealt with several aspects, especially those referring the role of privacy or security and the attributes of websites [20]. Some works have proposed different mechanisms to generate trust such as the participation of a reliable third party and trust marks, that is, emblems or badges given by a third party to a website to certify the application of commercial policies which are respectful and abiding by the law. Likewise, reputation methods are usually considered as a way to reduce uncertainty and favours higher levels of trust in computer-mediated environments.

### 3.4 Reputation methods

The level of reputation that someone assigns to an organisation depends on the extent to which the organisation is seen as honest and the concern shown for its customers [21].
Therefore reputation will be created in accordance with the extent to which the organisation meets its commitments to its customers and how it handles them.

Generally, reputation is a means of building trust, as one can trust another based on a good reputation. According to [22], reputation is an expectation about and agent behaviour based on information about or observations of its past behaviour. This definition implies that reputation is based on (1) the past experience and (2) the collected referral information, so a reputation management system should include:

- how an agent rates the correspondent based on the past interaction history;
- how an agent finds the right reviewer in order to select a new partner;
- how the agent systematically incorporates the testimonies of those witnesses.

In [23] the authors identify three goals for reputation systems:

1. To provide information to distinguish between a trustworthy principal and an untrustworthy principal;
2. To encourage principals to act in a trustworthy manner;
3. To discourage untrustworthy principals from participating in the service.

Some properties are common to most reputation-based trust models:

- The computational model, that is, whether the trust mechanism is centralized (a central certification authority exists) or decentralized;
- Metrics for trust and reputation: it refers to a metric which offers a value in order to express the reputation (and trust) of an entity as provided by the reputation mechanism. We have to differentiate between the reputation value of an agent and the feedback one is required to provide at the end of a transaction. Continuous metrics are usually considered (values are scaled between -1 and 1, or between 0 and 1);
- Type of reputation feedback: reputation feedback might be positive, negative, or both. Regarding an accomplished transaction, the reviewer can supply with binary, discrete or continuous values;
- Reliability: the trust model should help the users defend themselves against malicious information, including trust values propagated by other users into the system;
- Local control: in decentralized applications, data is stored by various entities in the system. These entities should not be able to change the trust and reputation values;
- Scalability: the trust model should scale well with an increase in the number of nodes. Bandwidth and storage costs increase with new nodes added;
- SLA or QoS negotiation: some reputation models are directly applied for negotiation of service level agreement (SLA) or quality of service (QoS). SLA o QoS agreements are usually incorporated in the direct trust component.
Following a classification of reputations methods is offered:

- The game-theoretical approach for reputation: It supposes that agents are continuously playing the same game and the rest of the players will learn this reputation;

- Reputation in Internet sites: Internet sites mainly use summation-based reputation systems, based on counting all votes or grades an entity receives. The most widely known reputation systems of this kind are eBay, Amazon, BizRate, etc;

- Reputation models based on referrals network: Reputation is based not only on the past interactions between entities but also considering the referrals the entity can obtain using a social network;

- Belief-oriented trust: Similarly to the referral networks, they introduce a more sophisticated technique for computing the trust/reputation value to consider the Dempster-Shafer theory and the theory of Marsh Agent-based approaches;

- Agent-based approaches: Considered as a good formalization for a wide variety of distributed systems. They are based on the openness propriety of multi-agent systems, the fact that the agents are self-interested, proactive, know only a local part of the acting world and no central authority restricts the behaviours of all agents;

- P2P approaches: In P2P systems, one main concern is the identification of malicious peers. The aim is how to aggregate the local trust values without a centralized storage management, what trust metric should be considered and how to store reliable and securely the trust values across the network;

- Incentive compatible approaches: As we have seen above, trust can be obtained both from direct interactions and via a third party source. After a transaction is finished, agents have to report about the result. Such information might be false or incorrect so it is possible to offer incentives in order to reduce or eliminate these risks.

3.5 Reputation methods in TAS³

WP5 (Trust Policy Management) proposes a Trust Policy Management architecture which interfaces with the remaining TAS³ architecture by means of a Trust Policy Decision Point (Trust PDP) which adheres to a standard PDP interface. The Trust PDP makes its decision with the aid of three different trust services: (1) Behavioural Trust Service, (2) Structural Trust Service and (3) KPI-Based Trust Service.

KPI-Based Trust Service might be considered as an evolution of others less objective and summation-based reputation methods (e.g. eBay’s ratings). TAS³ proposes a more objective approach in trust valuation taking into account particular criteria. Thus a TAS³’s user can consider as KPI a certain SLA. After the interaction between the users, the trustee gets these quantifiable values and compares them to the objectives in order to obtain trust indicator values. These indicator values are then aggregated and normalized in order to obtain a unified trust level value. These processes are described in D.5.1 in a more detailed way.

It would be possible that some of the issues analyzed in WP10 can integrate as KPI in the future platform, but this is a question which must be studied in-depth.
4 Security Testing

Target applications of TAS³, e.g. in the employment or e-Health sectors, involve management of personal data, and hence security is certainly a major concern. In this chapter we overview security challenges and survey existing approaches to security testing.

4.1 Web Services Security Challenges

Service-based technology promises to easily integrate software components deployed across distributed networks by different providers. Its great appeal derives from the announced flexibility and interoperability among heterogeneous platforms and operating systems, that is achieved by means of loose coupling, implementation neutrality and flexible configurability. However, as always happens in dealing with highly distributed applications that can evolve in an open-world, security aspects are highly critical in designing and developing Web Services. Thus, architects and developers would refer to specific techniques in order to protect their solutions against unauthorized accesses.

Basic requirements for secure interactions match with a composition of the main information security concepts: availability, confidentiality, integrity, authenticity and non-repudiation [24]. In other words, security assets require the concurrent existence of

- readiness for correct service for authorized actions only;
- absence of unauthorized disclosure of information;
- absence of improper system alterations;
- assurance that a message is created or sent by the source it claims to be from.

Cryptography is the core asset in most of the scenarios dealing with security aspects. Traditionally, cryptographic approaches are partitioned in two major classes: symmetric or asymmetric key. In the former case, the actors of a system use the same key in order to both encrypt or decrypt the messages. In such scenarios, the key is kept secret, known to some authorized users. Historically, this approach based on this class of encryption has been extensively employed as a countermeasure to those attacks that cause information release (a.k.a. passive attacks).

In contrast, in the case of asymmetric cryptography, the ability to encrypt messages under a given key is separated from the ability to decrypt it. Each actor of the scenario has a pair of keys (K1,K2). Messages encrypted under one key (e.g. K1) can be only decrypted by means of the other (e.g. K2), and vice-versa. In other words, these keys define a pair of mathematical transformations, each of which is the inverse of the other, and neither of which is derivable from the other. In the most common scenario, each actor reveals as public one of those keys and keeps private the other. In this manner:

- the sender can “sign” the messages encrypting them with its private key. All the receivers can prove the actual source of the message decrypting it by means of the sender’s public key;
the sender can “sign” the messages encrypting them with the public key of a given receiver. Only the receivers having the “appropriate” private key can receive and decrypt the messages.

Asymmetric encryption can also serve as a foundation on which to construct countermeasures to those attacks performing some kind of modification or processing on the messages (a.k.a. active attacks).

Cryptographic methods which permit to verify the origin of data and prevent an individual or entity from denying of having performed a particular action related to data, are used to achieve the authenticity and non-repudiation property. The most common method of asserting the digital origin of data is through digital certificates, that can also be used for encryption and are a form of public key infrastructure. The ways in which a party may attempt to repudiate a signature present a challenge to the trustworthiness of the signatures themselves. The standard approach for mitigating these risks is to involve a Trusted Third Party (TTP). One of the primary roles of the TTP is to establish a repository of digital certificates that embody the public keys that correspond to private keys used for digital signatures. These certificates are used for verifying that the digital signatures, effected using private keys, correspond to the public keys embodied in the digital certificates. In the legal sense the term “non-repudiation” can assume a different mean. Legal issues are addressed in TAS^3 WP6.

The conceptual model underlying the communication among Web Services provides the possibility that messages may pass via a number of intermediate nodes on their way from sender to ultimate receiver. Thus, it is a common understanding to distinguish at least two kinds of strategies for addressing protective measures with cryptographic procedures: security at the transport level, security at the message level.

Enforcing the security at transport level means that the authenticity, the integrity, and the confidentiality of the message (e.g. the SOAP message) are completely delegated to the lower-level protocols that transport the message itself (e.g. HTTP + TLS/SSL). In this case, the message is protected end-to-end from the sender to the receiver (see Figure 4.1). For example, such protocols may use public key techniques to authenticate both the end points and agree to a symmetric key, which is then used to encrypt packets over the (transport) connection [25]. This solution may prevent that the communication contents can be processed by intermediary third parties that are able to terminate the transport protocol.

On the other hand, security approaches at the message level specify that the content of the messages exchanged between two parties should be digitally signed or encrypted or both. In other words, as described in Figure 4.2^1, only the contents of the SOAP envelop (i.e. the header, and the body) are encrypted, not the whole envelop. Web Services protect their resources against unauthorized access with dedicated message-level encryption with

---

^1 Figure 4.1 and Figure 4.2 are taken from [26]

Version 1.5
suitable means for

- identify the sender of each message;
- verify the claimed identity of the sender of a message;
- apply access control policies to determine whether the sender (i.e. the client) has the right to use the required resources.

In this sense, many standards such as WS-Security [27], XML Encryption [28], XACML [29] and XML Signature [30], address message level security within SOAP and Web Services.

Another challenging issue of the service oriented approach is how to orchestrate existing services into more complex ones, by guaranteeing that their composition complies with some desirable security properties. In [31], the authors describe a semantics-based framework to model and orchestrate services in the presence of both functional and non-functional constraints, with a special concern for security properties. An abstract way to express security constraints in service orchestration is presented in [32]. The authors of this work provide a tool that relies on code generation and meta-models to make mapping between abstract specifications and actual code, including security mechanisms, such as X.509 certificate, username and password, encrypted data with XML Signature and XML Encryption.

4.1.1 Attack Patterns

In [33], Bertino and al. survey the most common threats facing with the security of a software system. In the following, we recall a no exhaustive list of traditional security attacks:

**Message alteration:** message information is altered by inserting, removing, or modifying information created by the originator of the information and mistaken by the receiver for the intention of the originator;

**Confidentiality:** the information within the message is visible to unauthorized participants;

**Falsified messages:** the attacker constructs counterfeit messages and sends them to a receiver who believes them to have originated from a party other than the sender;

**Principal spoofing:** the attacker alters the sender of the message appearing as someone/something else;

**Man in the middle:** in this context, this type of attack occurs when a third party poses as the other participant to the real sender and receiver in order to fool both participants. In other words, the attacker applies techniques of principal spoofing in parallel on the sender and the receiver interfaces;
Forged claims: the security claims of the messages are created “ad-hoc” in order to gain access to unauthorized resources/information;

Denial of service: the attacker forces the system under attack to do a large amount of work. The system under attack degrades its services or can even fail completely.

Many of the security attack patterns described above can have an instantiation within the Web Services domain.

In [34], attacks on Web Services are broadly classified into:

- Attacks leveraging UDDI vulnerabilities;
- Attacks using weaknesses in WSDL;
- Attacks on the SOAP messages;
- Other general attacks such as recursive payloads, schema poisoning and routing detours.

By carefully analyzing the interfaces within a WSDL (Web Services Description Language) file or UDDI (Universal Discovery Description and Integration) specification, attackers can get useful information about the Web Service in order to generate faults. Attacks on the SOAP protocol include parameter tampering and replay attacks. Recursive payloads, and schema poisoning are attacks aimed to XML parser. In the recursive payloads attack, malicious attackers can easily create documents that attempt to stress and break an XML parser by creating an XML document that is 10000 or 100000 elements deep. This could lead the common DOM and SAX XML parsers to recursively parse the elements and cause memory related issues such as buffer overflow. Schema poisoning is an XML Schema based attack. XML Schema provides formatting instructions for parsers when interpreting XML documents. Since most Web Services implementations validate XML instances against a schema before processing, an attacker might replace an existing schema with a malicious one to cause the Web Service to fail. The attacker might change the data type to cause severe denial of service attacks by using schema poisoning techniques.

Routing detours is another known attack that could be launched against a Web Service-based system: the WS-Routing specification provides a way to direct XML traffic through a complex environment. It operates by allowing an interim way station in an XML path to assign routing instructions to an XML document. XML gateways can perform encryption and decryption of WS-Security documents. A compromised or malicious gateway may participate in a man-in-the-middle attack by inserting bogus routing instructions to point a confidential document to a malicious location. SOAP intermediaries that are extensively used in a Web Services environment could also participate in a malicious activity by causing routing detours.

There are many attacks concerning service routing. A lot of them exploit the vulnerabilities of BGP (Border Gateway Protocol) protocol that is the de facto protocol enabling inter-domain routing in the Internet. A Survey of BGP security issues and solutions is presented in [35].
4.1.2 XML security verification

Message exchange is one of the core services required for system integration in SOA environments. In SOA Web Services, communications are made through XML-based messages called Simple Object Access Protocol (SOAP) [36] messages. Since messages may carry vital business information, their integrity and confidentiality needs to be preserved and SOAP message exchange in a meaningful and secured manner remains a challenging part of systems integration. XML security focuses on message security. Unfortunately, those messages are prone to attacks that can lead to several consequences such as unauthorized access, disclosure of information, identity theft. These attacks are all based on an on-the-fly modification of SOAP messages.

WS-Security [27] provides mechanisms to ensure end-to-end security and allows to protect some sensitive parts of a SOAP message by the mean of XML Digital Signature [30] and XML encryption [28]. However, XML Signature weaknesses could let an attacker modify SOAP messages without altering the signature. In the setting of SOAP security, we refer to these as XML rewriting attacks, to differentiate them from other classes of attacks concerning XML messages, such as XML injection. In the following we will overview possible XML-attacks and existing solutions to detect them.

4.1.2.1 XML rewriting attacks

XML rewriting attack is a general name for a distinct class of attacks based on the malicious interception, manipulation, and transmission of SOAP messages in a network of communication systems. We review existing solutions to avoid these attacks and highlight their limitations to detect all range of rewriting attacks.

Parts of a SOAP message can be encrypted and/or signed using XML Digital Signature [30]. The signature is used by the receiver to check integrity of the message and authenticity of the sender.

XML Signature defines XML syntax and processing rules for signing and verifying digital signatures over one or more data objects. XML Signature uses a set of indirect references to each signed data object, allowing for the signing of several potentially noncontiguous and/or overlapping data objects. For each signed data object a Reference element, which points to the object via a URI, contains a digest value computed over that object. This indirect referencing does not give any information regarding the actual location of the signed object. Therefore, the signed object can easily be relocated and the signature value still remains valid. In cases where the location of the data object is important in the interpretation of the semantics associated with the data, this can be exploited by an adversary to gain unauthorized access to protected resources. This is the main limitation of XML Digital Signature. McIntosh and Austel [37] have shown that the content of a SOAP message protected by an XML Signature as specified in WS-Security can be altered without invalidating the signature.

In [38] the authors proposed an inline approach that adds to the SOAP message a new header element called SOAP Account before sending it to the legitimate receiver. The general objective is to protect the integrity features of SOAP messages while they are in transit from hop to hop so that the ultimate recipient of the message can be sure that no malicious intermediary modified the message. The SOAP Account contains information about the structure of the SOAP message, such as the number of header elements in the SOAP message, the number of signed object, the successor and predecessor relationship of each
signed object. This **SOAP Account** element must be signed by the creator using its X.509 certificate. Each successive SOAP node must sign its own SOAP account concatenated with the signature of the previous node. Although using SOAP Account can detect XML rewriting attacks very early in the validation process by a legitimate receiver of a SOAP message, a SOAP Account itself is vulnerable to XML rewriting attacks. The authors of [39] point out the weakness of **SOAP Account** by constructing realistic counterexamples. In [40], the authors proposed a rule-based tool to help detect incorrect uses of WS-Security in SOAP. Web Services Enhancements (WSE) [41] is a SOAP library, implementing WS-Security and other specifications, that uses WS-Security Policy as part of its configuration data. This tool, called Advisor, is able to detect typical errors in WSE configuration and policy by running several static queries and generating security reports stating the threats that might occur due to the missing syntactic conditions. It also generates a remedial action that could be used by the author of the policy files to fix the flaw.

### 4.1.2.2 XML injection attacks

XML injection is the process of inserting elements or attributes into an XML document as a result of improper input validation. The insertion of elements or attributes may allow an attacker to sufficiently alter the structure of the XML document such that subsequent processing is affected and application behaviour is modified. XML injection may allow an attacker to effectively overwrite previous nodes in the document or re-initiate processing with arbitrary supplied values. Depending on the implementation and subsequent processing, this may lead to tampering issues that can result in privilege escalation or spoofing attacks. An important test value is the `<![CDATA[...]]>` tag that directs the XML parser to ignore its contents. This allows any character string to be contained within an element and may provide an avenue for XML injection as special characters will be left effectively untouched and may be inserted into a resulting XML document. A particular type of XML injection is the XPath injection. **X**Path **P**ath **L**anguage (XPath) is a language for accessing elements of an XML document. XPath queries may be dynamically constructed from user inputs and if these inputs are not properly validated, the resulting query may be vulnerable to injection and allow access to unauthorized parts of the XML document or database. Specifically, XPath special characters may be used to manipulate the nodes or attributes that are referenced by the application. Similar to SQL injection, XPath injection is most easily detected by supplying the single quote and double quote character to each web service parameter one at a time, and assessing the responses obtained. Fields vulnerable to XPath injection will cause the application to construct a syntactically invalid XPath query, due to the addition of the quote character, and an error will occur when the query is processed.

### 4.2 Approaches for Security Testing

An important aspect of (web) service security is the activity of fault detection. Indeed most breaches are caused when a system component is used in an unexpected manner. Improperly tested code, executed in a way that the developer did not intend, is often the primary culprit for security vulnerability.

The situation that major concerns is the lack of security testing standards and specifications [33]. Many points of vulnerability such as, functional level, input manipulation, informa-
tion disclosure, can be managed by:

- organized testing phase and strategies based on integrity and confidentiality [42, 43];
- the adoption of a fault model covering all interaction aspects of the service and spanning all the Web Services layers that include UDDI, WSDL, and SOAP [44];
- use of security patterns and/or problem frames for defining issues and solutions and problems related to security aspects on a fairly detailed level of abstraction [45, 46].

Indeed testing can be used to assess the quality attributes of Web services and hence increase the requesters’ trustworthiness. Robustness and other related attributes of Web services can be assessed through test case generation and are designed by, firstly, analyzing WSDL document to know what faults could affect the robustness quality attribute of Web services, and secondly, using the fault-based testing techniques to detect such faults [47]. Some indications on how minimize the effort of testing and reduce the factors that need to be tested for increasing the security of the developed applications are in [33].

Focusing in particular on testing aspect, the different strategies and approaches that have been developed over the years can be be divided into passive or active mechanisms. Passive consists of observing and analyzing messages that the component under test exchanges with its environment [48]. In this approach, which has been specially used for fault management in networks [49], the observer can be either on-line or off-line: an on-line observer collects and checks in real time the exchanged input/output while an off-line observer uses the log files generated by the component itself. Recently passive testing has been proposed as a good approach for checking whether a system respects its security policy as in [50, 51]. In this case, formal languages have been used in order to give a verdict about the system conformity with its security requirements. Active testing is based on the generation and the application of specific test cases in order to detect faults. Recently researches on active testing of web services [52, 53] have been focused on the problem of verifying that a system is in conformity with its security policy such as [54, 55].

All the above mentioned techniques have the purpose of providing evidence in security aspects, i.e. that an application faces its requirements in presence of hostile and malicious inputs [56]. Like functional testing, security testing relies on what is assumed to be a correct behavior of the system and on non-functional requirements. However the complexity of (web) security testing is bigger than functional testing and the variety of different aspects that should be taken in consideration during a testing phase implies the use of a variety of techniques and tools. Among the different approaches the most common adopted methodologies and techniques are [42]:

- Negative testing;
- Fuzz testing;
- Information Gathering;
- Injection;
- Confidentiality & Integrity;
- Web Services Security Extensions;
4.2.1 Negative testing

Beizer defines Negative Testing as *Testing aimed at showing software does not work* [57]. Currently the concept associated to negative testing concerns the use of a methodology which attempts to show that the application does something that it is not supposed to do [58]. Negative testing has an important role in software security because it can discover significant failures, produce strategic information about the model adopted for test case derivation, and allow overall confidence in the quality and security level of the system. Although negative testing is a powerful and effective approach, it is also a hard-to-manage task which may produce unwelcome information or force security error handling.

Common best practices of negative testing are usually grouped into 6 different subsets [59]:

1. Boundary Value Analysis and Equivalence Class Partitioning: These two techniques are widespread used in testing software application and are based on input and output data, and an expectation of the system's behaviour [60]. In particular, they examine data elements that can take a continuous range of values, using the requirements and design to predict boundaries where the system's behaviour changes. The idea is to have a set of values (at least three values), one on the right boundary (if exists), one on the left boundary (if exists), one in the middle of the range. If the boundary is between valid and invalid ranges, the test case that uses the invalid value will be a negative test. Equivalence Class Partitioning (ECP) looks at the range between the boundaries. Each member of a given equivalence class should, in the context of a known test, make the system do the same thing so the tester does not have to test every value in an equivalence class;

2. State Transition testing: this method is based on state transition diagrams, or semantically equivalent diagrams, and derives test cases to examine whether unreachable states are indeed unreachable. A variation on this works in the same way as n-switch testing;

3. Test against known constraints: this technique is based on the explicit and implicit restrictions and constraints. These constraints are considered additional requirements and exploited for generating a variety of negative tests. Typically, these tests involve measurement and observation of the system behaviour, rather than direct tests against expectations;

4. Failure Mode and Effects analysis: in this case a fault model of the system is used as basis for tests generation. In some cases the fault/failure mode can be derived by a Root-cause analysis on existing faults, system design documentation, knowledge of characteristic problems of the infrastructure. Is it also possible that the analysis of system behavior requires to monitor the systems in use;

5. Concurrency: The purpose of testing is to stress the concurrent use of resources such as data, database entities, files, connections and hardware;
6. Use cases: Use cases that exercise non conventional paths can help to improve the
design by illustrating user activities that are outside the normal range of expectation,
and allow a formal approach to test selection and coverage.

As a guideline in [61] the top 10 negative test cases that should be developed during a
negative testing phase are listed. In summary they are:

1. Embedded Single Quote - Use test that contains one or more single quotes;
2. Required Data Entry - Use each field that has been indicated as being required;
3. Field Type Test - Use specific data entry requirements (date fields, numeric fields,
   phone numbers, zip codes, etc);
4. Field Size Test - ensure that only the specified number of characters can be entered;
5. Numeric Bounds Test - For numeric fields, test for lower and upper bounds;
6. Numeric Limits Test - In database systems and programming languages which allow
   numeric items use the bound limits to ensure correct behavior. For numeric data en-
   try that do not have specified bounds limits, work with these limits to avoid numeric
   overflow error;
7. Date Bounds Test - For date fields, use lower and upper bounds;
8. Date Validity - For date fields, ensure that invalid dates are not allowed;
9. Web Session Testing - Force to launch web pages without first logging in;
10. Performance Changes - Compare the prior release performance statistics with those
    of the current release.

4.2.2 Fuzz Testing

The word fuzzing is conventionally used to refer to a black-box software testing method
for identifying vulnerabilities in data handling. It involves generating semi
valid data and
submitting them in defined input fields or parameters (files, network protocols, API calls,
and other targets) in an attempt to fail the program and find bugs. Usually the term “fuzz”
means to give a set of inputs taken from random data to a program, and then systematically
identify the failures that arise [62]. Semi valid data are correct enough to keep parsers from
immediately dismissing it, but still invalid enough to cause problems. Fuzzing is useful in
identifying the presence of common vulnerabilities in data handling and the results of fuzzing
can be exploited by an attacker to crash or hijack the program using a vulnerable data
field. Yet, fuzzing covers a significant portion of negative test cases without forcing the
tester to deal with each specific test case for a given boundary condition. Sutton et al.
present a survey of fuzzing techniques and tools [62]. It is out of the scope of this section a
complete overview of the numerous proposals for fuzz testing. We propose here a general
classification of the different proposals and some of the most recent proposals for each
subset. In particular fuzzing approaches can be divided into: data generation; environment
variable and argument fuzzing; web application and server fuzzing; file format fuzzing [63];
network protocol fuzzing [64]; web browser fuzzing; in-Memory fuzzing. In the specific area

Version 1.5
of web service security, fuzzing inputs can often be generated by programatically analysing
the WSDL or sample SOAP requests and making modifications to the structure and content
of valid requests. Some example of useful test objectives provided in [42] are listed below:

- Numerical Values: identify mishandling of numerical fields;
- Base64 Encoded Values: Modify base64 values so that to contain characters other
  than digits, lowercase and uppercase letters, plus and minus etc;
- Handling of values that have an unspecified format by means of supplying unexpected
  inputs. Usually this will include excessive length, meta-characters etc;
- Identify possible injection flaws in sub-systems by pinpoint any sub-systems being
  utilized by the web service;
- Assess the security of the data output mechanism: if a custom display format is used,
  any special character used by the format should be tested;
- Identify simple session token weaknesses by means of modification of valid session
  token checking if it presents a valid connection;
- Assess the logging mechanisms handling of simple attacks.

File fuzzing strategy is also used for detecting XML data vulnerability. In particular, two main
testing methods, generation and mutation are adopted [65]. XML-based fuzzing techniques
using the first approach parse the XML schema and the XML documents to generate the
semivalid data. The second way to get malformed data is to start with a known set of good
data and mutate it in specific places. The authors of [66], propose a methodology and an
automated tool performing efficient fuzz testing for text files, such as XML or HTML files, by
considering types of values in tags. This tool named TAFT (Tag-Aware text file Fuzz testing
Tool) generates text-files, extracts tags and analyzes them, makes semivalid data by random
data returned by different functions according to types of values and inserts them into values
of tags, then executes automatically a target software system using fault-inserted files and
observes fault states.

4.2.3 Information Gathering

In general the information gathering process has the purpose of identify available web ser-
VICES, so that to obtain information about their operating environment and infrastructure.
To enforce security aspects the testing techniques usually tend to mimic methods used by
potential attackers. The methodologies adopted are mainly based on scanning techniques
such as [42]:

- WSDL Scanning: the detailed information present in a WSDL document can be an
  extremely useful source of information for a web service attacker as it provides infor-
  mation about the services themselves as well as required parameter types. When
  assessing web service security, it is essential to determine the degree of exposure
  provided by the associated WSDL documents and to ensure that sensitive services or
  methods are not disclosed. A possible testing method is to variate the naming of the
  WSDL conventions so that to attempt to retrieve WSDL documents from URLs;
• SOAP Fault Error Messages: error messages within SOAP Faults can contain detailed platform information and implementation details such as code fragments or stack traces. These error messages can provide useful feedbacks for attack or reveal sensitive information. A possible testing approach is to induce errors in the web service application by supplying invalid inputs and observe the web service response;

• Web Method Enumeration: it is focused on the methods that may be published in the WSDL document. The testing target is to guess web method names that may have been deliberately or inadvertently omitted from the WSDL. A list of likely candidates can be used for this purpose.

4.2.4 Injection

In general web services can interact with a variety of systems and for this must be resistant to injection attacks when external systems are accessed or invoked. The most prevalent injection vulnerabilities include SQL injection, command injection, LDAP injection, XPath injection, and code injection. There are undoubtedly many other forms of injection and the tester should be aware of these for testing of different subsystems. Most injection vulnerabilities can be easily and quickly identified by the process of fuzzing due to the presence of meta-characters in various language syntaxes. Recently, SQL Injection Attack has become a major threat to Web applications. SQL injection occurs when a database is queried with an SQL statement which contains some user influenced inputs that are outside the intended parameters range. If this occurs, an attacker may be able to gain control of the database and execute malicious SQL or scripts. In the following we will review some solutions to mitigate the risk posed by SQL injection vulnerabilities. Huang et al. [67] secure potential vulnerabilities by combining static analysis with runtime monitoring. Their solution, WebSSARI, statically analyzes source code, finds potential vulnerabilities, and inserts runtime guards into the source code. The authors of [68], present a static analysis framework (called SAFELI) for discovering SQL injection vulnerabilities at compile-time. SAFELI analyzes bytecode and relies on string analysis. It employs a new string analysis technique able to handle hybrid constraints that involve boolean, integer, and string variables. Most popular string operations can be handled. Halfond and Orso [69] secure vulnerable SQL statements by combining static analysis with statement generation and runtime monitoring. Their solution, AMNESIA, uses a model-based approach to detect illegal queries before they are executed on the database. It analyzes a vulnerable SQL statement, generates a generalized statement structure model for the statement, and allows or denies each statement based on a runtime check against the statically-built model.

4.2.5 Confidentiality & Integrity

Confidentiality in a web services context provides the ability for client-intermediary-server communications to occur without eavesdropping by unauthorized parties. Integrity provides the ability for the communications to be protected against tampering by unauthorized parties [42]. Considering confidentiality, the security aspects rely on the properties of the secret key used for decryption and the key management process. Thus testing activity should be focused on:
• assess the suitability of the cipher adopted so to avoid that an attacker successfully crack the encryption and recover plain text data;

• the items within web service communications that are encrypted. Testing should assure that all sensitive portions of messages are protected against unauthorized eavesdropping. Significant items to look for include message integrity check fields, signatures, timestamps and so on.

Testing security aspects of message integrity involve digital signatures or cryptographic hash functions or both. In web services it is also important to check that message integrity aspects meet the requirements of the application. Thus testing activity should:

• Check malicious use of a valid message or set of messages that has already been accepted by the web service previously. A possible method is to use capture and replay mechanism so to collect previous messages useful to test whether or not the application accepts them;

• Check whether failure may allow messages to be sufficiently manipulated by an attacker such that the meaning of the message is changed. The analysis of the elements contained within integrity mechanisms and the remaining unprotected elements within the SOAP body should be used for this purpose.

4.2.6 Web Services Security Extensions

The Web service security is an important aspect of the services interaction and communication. To rule the web services security the SOAP Message Security (WSSSMS) standard [27] has been recently proposed. The WSSSMS specification, which is focused mainly on services communications, provides three main features: the transferring of security tokens, the ensuring of message integrity, and message confidentiality.

The implementations based on WSSSMS are tested for verifying their compliance with the standard. Compliance testing involves two main areas: XML validation, that means validating the SOAP message output against the appropriate XML schemas, and application behavior, that involves testing the communication component with various messages and assessing the applications behavior.

However it is out of the scope of WSSSMS, problems such as authentication and non-repudiation. Thus, testing effort should be focused to assess that the cryptographic elements utilized provide the confidentiality and integrity required by the application.

Other standards have been developed to complete the WSSSMS view of security such as the WS Secure Conversation [70], WS-Security Policy [71] and WS-Trust [72], which focus security constructs, security policies and secure sessions for web service communications. However, implementation following the above mentioned standards should in any case be tested for security weaknesses [73]. A web security test plan should therefore include test cases for:

• Assess the response of the web service to malformed XML input by means of valid SOAP requests that do not comply with the various security schemas;

• Determine the effectiveness of XML canonicalization in the web service by generating XML documents that are logically equivalent and verify that they are treated identically by the web service;
• Assess the web service response to SOAP messages that do not meet security policy requirements.

4.2.7 Logging

The logging functionality is one of the most tested area of web services. Testing approach are quite similar to any other form of application, and focus mainly on the injection of special inputs, overflowing logs, and enumerating and analyzing actions logged [72]. Due to the wide variety of logging methods available, it is not possible to list all the features that should be tested. Here below we provide just a small subset [42]:

• Verify the ability of the web service to log messages that contain special separator characters;
• Determine if the logging mechanisms can be influenced by white space injection;
• Assess the possibility of inserting XML elements and/or attributes into an XML based log (the same with HTML tags into a HTML based log);
• Verify the vulnerability of the logging mechanism in case of arbitrary entry creation via carriage return and line feed injection;
• Assess the handling of log data after reaching the upper log size limit;
• Verify if there is the possibility of retrieving or otherwise revealing information contained within log entries;
• Check if the level of logging in place is sufficient;
• Focus on logic flaws that could cause some actions to be logged incompletely or not at all.

4.2.8 Authentication and authorization

Authentication and authorization are two important functions of web services. Usually authentication is performed using a custom implementation or a standards-based implementation based on OASIS and XML standards [42], thus testing an authentication and authorization solution may involve a detailed cryptographic analysis of the authentication protocol and access control system as well as an assessment of the implementation.

Considering in particular authentication, even if each testing approach could depend on the system under test, it is important to check the procedure used to determine user identity and the authentication protocol used. For this, test cases usually are either focused on forged or modified credentials, missing credentials and other inputs to induce error handling in the application or to capture a legitimate exchange and attempt to extract the credentials from it necessary for accessing the application. A non exhaustive list of possible testing features could be [42]:

• Assess the susceptibility of the authentication exchange to brute-force or dictionary attacks;
• Verify the system response to forged or modified credentials, such as self-generated certificates or arbitrary tokens, or missing credentials;

• Assess the susceptibility of the authentication exchange to replay attacks or to tampering by a malicious user;

• Determine the ability of the web service to resist man-in-the-middle attacks;

• Determine the factors of authentication used to identify users to the system;

• Verify the ability to tamper with or subvert authentication session mechanisms;

• Assess the strength of authentication credential stores;

• Assess the ability to eavesdrop on the authentication exchange;

• Identify if the web service sufficiently verifies digital certificates presented to it.

Concerning instead authorization issue, the implementations should be tested for verifying that the correct access or role is assigned to an authenticated user and that its access or role cannot be tampered to enable access to unauthorized information or functionality. The approaches for testing this functionality can be white-box or black-box. In the former, testing methodology uses the design and mechanisms available from documentation or discussions with stakeholders, in the latter this is derived by interactions with the web service.

In general, several standards can be used during the development of web services such as Security Assertion Markup Language (SAML) [74] and eXtensible Access Control Markup Language (XACML) [29]. In this case the security testing phase should also verify the token generation and distribution processes and attempt to find weaknesses that may allow forgery or unauthorized modification of tokens. However because neither SAML nor XACML explicitly define protocols for the authorization control, testing approaches should also verify that tokens are generated from a suitably pseudo-random source. Moreover the tokens themselves should be analyzed for predictability and opacity, and the presence of integrity and authenticity mechanisms to protect tokens during transit should be verified.

A further role of the testing activity is the evaluation and assessment of specific methods for establishing trust and exchanging authentication and authorization information between different domains such that proposed into WS-Federation and WS-Trust [72] standards. Without the intention to be exhaustive, a possible list of testing features for authorization could be:

• Assess the consistency and role definitions to business rules and policy;

• Determine the extent to which tokens can be manipulated or forged;

• Assess the resilience to session hijacking attacks;

• Determine the ability to manipulate trust and authorization between different trust domains;

• Identify susceptibility to address spoofing attacks.
4.3 Security Testing in TAS\textsuperscript{3}

WP10 does not centrally target the development of innovative approaches to security testing. However, existing techniques and tools can be adopted in development and testing of the TAS\textsuperscript{3} platform. Here we have surveyed the literature. All outlined approaches are relevant and could be adopted. In particular, within WP10 we will investigate the opportunity of including negative testing and fuzz testing, by means of the XML-based technology described in Chapter 8.
5 Functional Testing

Development of service based systems is still a discipline undergoing intense research by academia and industry. In this chapter we give a brief overview of related work and existing tools in the area of testing of Service Oriented Software focusing in particular on Web Services that by far are the most used technology to make real the SOA vision. Nevertheless most of the approaches we report here can be easily adapted to other SOA technologies.

The resemblance of the SOA domain to the Component-based (CB) domain makes it possible to reuse some of the results and approaches proposed in testing CB. Nevertheless, as happened for Component-based (CB) domain, software testing of service oriented systems requires a revision of the traditional testing phases. In particular in the TAS\textsuperscript{3} vision, service verification spans also to the run-time phase and cannot be considered fulfilled after service deployment. Therefore three different phases can be identified in the testing of services: in a first phase, called off-line, the service is tested by its developer trying to anticipate possible usage scenarios, in a second phase, referred to in this document as Audition, the service is tested when it asks for registration to a directory service, in a third testing stage, that is monitoring, the behavior of a service oriented system is continuously checked also at run-time, during live-usage. Before explaining in detail these phases in the next sections we provide a classification of the involved actors in testing of services, as presented in [75]:

1. Developer: an organization that implements the service;
2. Provider: an organization that deploys the service and makes it accessible to possible users;
3. Integrator: an organization that provides services integrating (i.e., orchestrating) services provided by other parties;
4. Third-party: an organization that certifies the quality of a service in different contexts;
5. User: an organization that is interested in using a service.

In testing a service, each of these actors pursues different objectives and can apply different strategies.

5.1 Approaches to Off-line WS Testing

As stated above, off-line testing is carried on by the developer, or possibly by the provider of the service, in order to assess that the implementation of the services actually behaves as expected. In this phase services required by the service under test are “simulated” by the testing environment. In the service oriented domain the specification of the behavior could have been defined by the developers themselves or even by some external bodies, as in the case of choreographies. In the latter case the developer has to retrieve the specification of the service s/he wants to implement from the Net. Also the specification of the services that will be used by the service under development is usually retrieved from the Net using in general some kind of discovery service. Depending on the used specification, testing can be carried on during the off-line phase in different ways. In particular if the specification
is defined by the developer itself, the testing phase probably does not show many differences with respect to traditional testing. In this case the developer has to create stubs that will simulate the behavior of the expected accessed services. More interesting seems the possibility of using already developed specifications. For instance in [76] the authors show how the specification of a choreography can be used to test a service implementation. In their approach they automatically derive test cases from a specification, given as a Symbolic Transition System. Another approach targeting a similar objective, using however a different notation, is discussed in [77].

An interesting approach to the off-line unit testing of orchestration description, defined using BPEL, is described in [78]. In the paper the authors provide a testing architecture that permits to recreate an environment useful for testing the BPEL process. In particular the BPEL4WS service (called Process Under Test (PUT) in the document) will interact with a set of Test Processes (TPs) simulating the behavior of the composed services. The authors discuss advantages and drawbacks of having centralized or distributed implementation of the TPs. Finally the architecture proposed foresees the presence of a Control Process that in some way allows the TP to coordinate with each other in order to have a global control on the invocation sequences.

Other approaches have been proposed posing the focus on the use of orchestration and/or choreographies in order to check before run-time the possible emergence of failures. In particular the use of such information as main input for analysis activities is considered in [79]. The objective of the authors, in this case, is to formally verify, rather than testing, that some undesired situations are not allowed by the collaboration rules defined by a service orchestration definition. To do this, the authors, after having translated the BPEL specifications into Promela (a language that can be accepted by the SPIN model checker), apply model checking techniques to verify if specific properties are satisfied. Another approach to model-based analysis of WS composition is discussed in [80]. From the integration and cooperation of WSs, the authors synthesize Finite State Machines and then verify if the obtained result and allowable traces in the model are compatible with that defined by BPEL-based specification.

5.2 Testing by Enhancing the Directory and Discovery Service

The above list of five actors concerned with testing of services does not include the directory and discovery services. The very general idea is that the service can be tested at the time it asks for registration. Moreover, following such an idea, only “high quality” certified services will be guaranteed to pass the registration. In a semi-open environment this will provide some extra guarantees to the registered services. Nevertheless, in a completely open environment the discovery service could be cheated by a malicious service provider that after registration modifies the code associated to the end point.

Approaches belonging to this class can be differentiated mainly on the base of the information used to carry on the testing session. The possibility of enhancing the functionality of a UDDI service broker (for the case of a Web Service based approach) with logic that permits to perform a testing step before registering a service has been firstly proposed in [81] and [82], and subsequently in [83]. This idea is also the basis for the Audition framework [84] [85], (see Section 5.2.1) which permits to test and monitor a service as it interacts with other services. However, the information the Audition approach uses, and the correspond-
ing derived tests, are very different from those proposed in the cited papers. In particular, while in the cited works testing is used as a means to evaluate the input/output behavior of the WS that is under registration, the Audition framework grants the registration also on the base of the interactions occurred between the WS under registration and the providers of services already registered. For doing this the framework uses monitoring mechanisms to be inserted in the run-time infrastructure and to be used also for run-time checking. With reference to the information that must be provided with the service description, the authors of [81] foresee that the service developer provides precise test suites that can be run by the enhanced directory and discovery service. In [83], instead, the authors propose to include Graph Transformation Rules that will enable the automatic derivation of meaningful test cases that can be used to assess the behavior of the WS when running in the “real world”. To apply the approach they require that a service specifically implements interfaces that increase the testability of the service and that also permit to bring the service in a specific state from which it is possible to apply a specified sequence of tests. Moreover, in literature some approaches for run-time testing have been proposed, for instance in order to select the best service among the registered services providing the same functionality [86].

5.2.1 The Audition Framework

This section presents the Audition framework. More details on the approach underlying Audition can be found in [85]. The basic idea behind the Audition framework is to test a service when it asks for registration within a directory service. Then in case the service fails to show the required behavior the registration in the directory is not granted. In this sense we called the framework “Audition”, as if the service undergoes a monitored trial before being put “on stage”. Figure 5.1 shows the main elements of the framework. The figure intends to provide just a logical view, i.e., the arrows have not to be directly interpreted as invocations on methods provided by one of the elements. Instead they generally represent a logical step in the process and point to the elements that will take the responsibility of carrying on the associated task.

The process subsumed by the framework is activated by a request made by a service asking for being listed within the registry and is structured in eight main steps (the numbers in the list below correspond to the numbers in Figure 5.1):

1. a service S1 asks the registry (directory service in the figure) to be published among the services available to accept invocations. Contextually, S1 provides information concerning both the syntax (WSDL in the framework of the Web Service related technology) and a behavioral description of the offered service (expressing the protocol that a possible client should follow to correctly interact with the service);

2. the registry service stores S1 provided information marking them as “pending registration”. At the same time S1 related information are sent to a Testing Driver Service (TDS);

3. the TDS starts to make invocations on S1, acting as the driver of the test session, checking if the service behaves accordingly to the specification;

4. during the Audition, S1 will ask to the registry for references to other services necessary to complete the provision of the service. Indeed S1 could use other services
without asking to the registry since the references are hard coded in S1 definition. From the point of view of the framework in this case S1 is not different from a “basic service” since also at run-time it will continue to use the statically bound services;

5. the registry checks if the service asking for external references is in a pending state or not. If not references for the required service description file and its relative access point are provided. Instead in case the service is in a pending state the registry provides the information, such as the interface and the behavioral description for the requested service to a Proxy/Stub Service factory. This Service starting from the syntactic and behavioral description is able to derive proxy or stubs for the requested service. In case of a proxy generation the generated service will implement the same interface of the “proxyed” service but at the same time it will check if the invocation made by S1 is in accordance to the ones defined in the specification and then expected by the invoked service. In case no error is discovered the invocation is redirected to the real implementation of the service;

6. for each inquiry request made by S1 the registry service returns a binding reference to a Proxy/Stub version of the requested service;

7. on the base of the reference provided by the registry, S1 starts to make invocations on the Proxy/Stub versions of the required services in order to fulfill a request made by the test driver service. As a consequence the Proxy/Stub version of the service checks the content and the order of any invocation made by S1; in case a violation to
The specification for the invoked service is detected, the Proxy/Stub informs the registry service that S1 is not suitable for being registered. As a consequence the directory service removes from the pending entries the service currently under test, and denies the registration;

8. finally in case one of the invocation made by the TDS results in the detection of an error the registry is informed. As for the previous case the registration will be denied.

The availability of a registry enhanced with testing capabilities, granting the registration only to “good” services, should reduce the risk of run-time failures and run-time interoperability mismatches. A service asking for registration will undergo two different kinds of check before being registered. The first concerns the ability of the service of behaving according to its specification and the second of being able to correctly interact with required services.

Nevertheless some issues have to be considered in particular to derive a real implementation of the service and to better understand the applicability of the framework itself.

In order to automatically derive test cases for services asking for registration the framework requires the use of an increased service information model that should provide some description in a computer readable format of the service expected behavior. Such information model has to be provided to the service registry when a service asks for being included in the registry, and according to the framework the behavioral description has to be suitable for automatic test case derivation. Slightly different configuration of the framework can be derived for instance relaxing the part on automatic derivation of test cases with the usage of predefined and static test suites stored in the registry. We assume that in the framework an automatic test generator is available.

An important note concerns the reduced control over a service implementation by a third party such as the tester. In a SOA setting each organization has full control over the implementation of exposed services. This would mean that a service implementation can be changed by the organization to which it belongs and after that the registration has been granted by the registry, and obviously without informing the registry. As a result a non tested service will be considered as registered. Main consequence of this lack of control is that the framework can be fruitfully applied only within an environment in which the participating organizations are known and interested in collaborating with the registry in order to guarantee that no “bad” services will enter the stage.

Another relevant request posed by the framework concerns the fact that each interaction with the registry has to permit the identification of the sender. This constraint directly derives from the fact that the registry has to recognize the status of the registration for the invoking service when it asks for references to external services.

A final interesting note concerns the automatic generation of stubs and proxy. Stubs intend to simulate the behavior of an invoked service. However the automatic generation of a service stub asks for the storing in the registry of a complex service model such as for instance State Machines or other specification suitable for automatic test generation. Indeed in case the registered service model does not permit the automatic derivation of a suitable stub the framework foresees the generation of proxy services instead of stub services. A proxy service will check and log incoming invocation with respect to the model and then it will redirect the invocation to a real implementation of the service in order to generate a meaningful answer.

A real implementation of an enhanced version of a registry showing a standard UDDI Web
Service interface called WS-GUARD, able to apply the “audition phases”, is provided in [87]. WS-GUARD uses Apache related technologies and in particular the Axis2 SOAP container. Concerning the registry, an open source version of a UDDI registry that is provided by the Apache foundation under the project called jUDDI [88] is provided.

5.3 Runtime Monitoring

Monitoring is a runtime activity which collects data and transfers them to external functions for elaboration. In this section, we survey related work on runtime monitoring approaches directed to Web Services technologies. Nevertheless they do not make assumptions that logically prevent their applicability in a more general SOA setting.

Baresi et al. [89] propose an approach to monitor dynamic Web Service compositions, described in BPEL, with respect contracts expressed as assertions. These are checked at runtime, by invoking an external monitor service. In [90] they extend this work by introducing the concept of monitoring rules, a language (WS-CoL) to specify constrains on execution, the capability of setting the degree of monitoring at run-time and a proxy-based solution to enact the monitoring rules. A similar approach is proposed in [91] where the BPEL process is made fault tolerant by automatically identifying and monitoring desired services and replacing them upon failure, using a proxy service for discovering and binding equivalent Web Services that can substitute monitored services.

A framework for the runtime verification of requirements of service-based software systems is described in [92]. Requirements can be behavioral properties of a service composition, or assumptions on the behavior of the different services composing the system. The first can be automatically extracted from the composite process, expressed in BPEL; the latter are specified by system providers using the event calculus. System events are collected at runtime and stored in an event database; checking is done by an algorithm based on integrity constraint checking in temporal deductive databases. In [93], requirements are expressed in KAOS [94] and analyzed to identify conditions under which they can be violated. If such conditions correspond to a pattern of events observable at runtime, each of them is assigned to an agent for monitoring. At runtime, an event adaptor translates SOAP messages into events and forwards them to the corresponding monitoring agent.

Li et al. [95] present a framework for monitoring the runtime interaction behavior of Web Services and validating the behavior against their predefined interaction constraints. Monitor intercepts and analyses messages exchanged between a service and its clients, and validates the message sequence against the interaction constraints.

For validation purposes, the functional interface of a service can be coupled with the specification of the exposed QoS, which constitutes its Service Level Specification (SLS). Monitoring techniques are also largely used to monitor QoS levels: this chapter only considers functional monitoring, while QoS monitoring is surveyed in the next chapter.

5.3.1 Logical Architectures for Monitoring

Many of the approaches presented above have a monitoring component, e.g. a proxy, which represents a potential bottleneck or requires the modification of the workflow process, which could be error-prone and cumbersome.
An alternative architecture for integrating monitoring, property checking, and possible reactions may be based on an Aspect-Oriented Programming (AOP) approach. Instead of modifying the workflow process, it is possible to dynamically weave the needed actions into the code of the run-time workflow engine. This approach has been explored by Courbis and Finkelstein [96], who solve the problem by using non-standard and proprietary workflow engine and AOP tool.

Another use of AOP for monitoring is described in [97]: a selection mechanism for Web Services is proposed that allows dynamic switching between services, based on business driven requirements that can change over time. The selection procedure is triggered by a monitoring mechanism which observes criteria such as cost, presence on approved partner list, binding support, quality of service classifications, historical performance and proximity. AOP techniques, in the form of monitoring aspects, allow to dynamically insert measurement points in the system and to execute the monitoring logic tailored for a desired property.

Finally, in the PLASTIC project, a framework for monitoring has been released, called Dynamo-AOP. The framework allows a tester to monitor the functional properties of external services which a BPEL process interacts with, to realize a composite service. It is based on the conceptual model proposed in [90], but, with respect to the original design, its architecture is based on the dynamic expectation of the BPEL engine executing the monitored service compositions, achieved by using AspectJ [98] language. The tool can be downloaded freely from [99].

### 5.3.2 Some Industrial Proposals

Cremona [100] is a tool from IBM devised to help clients and providers in the negotiation and life-cycle management of WS-Agreements. It provides a component, the “Status Monitor”, which helps in deciding whether a negotiation proposal should be accepted or refused, on the basis of system’s available resources and the terms of an agreement. Once an agreement has been accepted by the client and the provider, its validity is checked at runtime by a “Compliance Monitor”, which can check for violations as they occur, predict violations that still have to occur, and take corrective actions.

Colombo [101] is a lightweight, optimized middleware for service oriented architectures supporting BPEL, also proposed by IBM. One of its features is the support of declarative service descriptions, such as those expressed using WS-Policy. It intercepts messages before they leave the system or before they are processed, and uses a pipe of dedicated policy-specific verifiers to validate messages with respect to a certain policy.

GlassFish [102] is an open-source community implementation of a server for JavaEE 5 applications. It allows for collecting data on response times, throughputs, numbers of request, and message tracing, of the deployed services.

soapUI [103] is a tool developed by Eviware Software AB, available both in free and improved commercial versions. It assists programmers in developing SOAP-based web services. In particular, it allows the developer to generate stubs of SOAP calls for the operations declared in a WSDL file. Additionally, it is possible to use soapUI to send SOAP messages to the web service and display the outputs. The tool can be used for black-box testing purposes.
5.4 Functional Testing in TAS³

In this chapter we have surveyed briefly main challenges and related works of testing and monitoring of Service Oriented Software, focusing in particular on Web Services technology. Most of the approaches we reported are general and can be adapted in order to be suitable methodologies to TAS³ systems. We have also provided a brief overview of existing tools in the area. In particular, we presented the Audition framework that had been originally conceived to certify services when they ask of being registered within the infrastructure. The characteristics of such a framework make it very suitable for exploitation within the TAS³ architecture. At the time of writing, the specification of the components to be included in the architecture is under development. To prevent confusion with the traditional “Audit” component, the name Audition has been dropped and the approach is being referred to in future deliverables as “On-line Compliance Testing” (OCT). Lot of investigation work remains to be done to understand how OCT can be seamlessly integrated within TAS³ architecture, and how adequate means to improve automation can be incorporated. There are also many integration challenges to address, as presented in Deliverable D12.1. We plan to derive the test suite based on the service exposed policies, and automation of the latter concern is partially addressed in Chapter 8 by means of the tool TAXI.
6 Quality of Service testing and assessment

The TAS$^3$ project recognises that both the factual/technical and the social/human trust angles need to be represented in a trustworthiness vision.

Factual trust in a system or organisation, or even a person, involves technical dependability that can be built by testing and monitoring. Social trust or gut feeling is harder to measure, as it is inherently not facts-based and comes from the psychological and social parts of human beings. In this chapter we survey these different aspects, in particular we will provide the state-of-art in QoS testing and monitoring in Section 6.1 and we will focus on User-Perceived Service Quality in Section 6.2.

6.1 QoS Testing

One of the key aspects of the Service Oriented Architecture (SOA) paradigm is to enable the dynamic integration between applications belonging to different, globally distributed enterprises.

Flexibility is given into the service-oriented systems by using late-binding mechanisms. In other words, let us consider the typical case of a service provider who is to offer a certain service ($S^*$) to their clients. In order to do so, the service provider could orchestrate a number of other services ($S_1 \ldots S_n$), which may be either under their direct control, or may be provided by third parties. Furthermore, for each service $S_i$, various implementations may be available. The binding between $S^*$ and one of the implementations of the $S_i$ it could be defined at run-time only when $S^*$ actually invokes the functionality provided by $S_i$.

Since services are intended to be discovered and used by other applications (e.g., by the Web), they make available declarative interfaces describing functional properties and Quality of Service (QoS) attributes. When the services candidated to bind are functionally equivalent, the choice between them can be dictated by QoS properties.

In literature, QoS is referred as combination of several qualities or properties of a service. In [104] some of the most used definitions of QoS attributes are given. In particular, the availability of a service is defined as percentage of time that a service is operating when invoked by a client. Reliability states the maximal admissible number of failures a service can raise in a given time window. Security and Trust properties include the existence and type of authentication mechanisms the service offers (e.g., resilience to attacks), confidentiality and data integrity (e.g., exchanged data are valid and usable). Response time is the time a service takes to respond to various types of requests. Throughput is defined as the rate at which a service can process requests.

In [105], the authors defined and discussed several QoS metrics for Web Services (i.e. Provider-advertised metrics, Consumer-rated metrics, Observable metrics). Focusing on the observable metrics, the authors defined an observation model, which allows users to define the metric types and formulas.

Within the Services Oriented technologies, the definition of QoS is strictly related with the notion of Service Level Agreements (SLAs). SLAs aim at ensuring a consistent cooperation for business-critical services defining contracts between provider and client of a service and the terms governing their individual and mutual responsibilities with respect to these
qualities [106].

Usually a SLA contains a technical QoS description with the associated metrics. Such technical information inside the SLA are also referred as Service Level Specifications (SLSs). When a formal SLS is specified, not only it establishes a contractual basis for negotiation between a service provider and a user of that service, but it can be also usefully exploited as a reference model for testing and monitoring purposes. In the rest of the section we will refer to the literature of SLA-based testing and monitoring and in Section 6.1.1 we will present in detail Puppet, a tool supporting the QoS testing in service oriented architectures.

IBM researchers have developed Cremona [100], a framework that implements the WS-Agreement specification and also facilitates the establishment of agreement between client and provider.

Menascé [107] proposed an automatic QoS controller that adopts traditional performance predictors to best tune the load of requests at design time, while more recent studies propose to make the system capable of self-adapting at run time: for instance Menascé and Bennani [108] reconfigure a web server so to optimize its performance while Abrahaao and coauthors [109], referring to a cost model of SLA (in terms of penalties and rewards), optimize financial return and propose to adopt different levels of operation. The above works aim to make the subject of monitoring, the workload itself, adaptive (opportunistic) depending on the measured QoS parameters. A proposal that also is aimed at making the monitoring less invasive comes from Bertolino et al. [110]; in contrast with the previously cited work, [110] focuses on the instrument employed for monitoring the QoS level, and adapts it to the workload. Therefore, this approach could be also usefully introduced to alleviate the task of scaling up SLA management to large grid environments, as also investigated in the Gruber system [111].

Keller and Ludwig [112] present the Web Service Level Agreement (WSLA) framework to define and monitor SLAs, focusing on QoS properties such as performance and costs. The monitoring component is made up of two services: the first, the Measurement Service, measures parameters defined in the SLA, by probing client invocations or retrieving metrics from internal resources; the second, the Condition Evaluation Service, tests the measured values against the thresholds defined in the SLA and, in case of a violation, triggers corrective management actions.

An automated and distributed SLA monitoring engine is proposed in [113]. The monitor acquires data from instrumented processes and by analyzing the execution of activities and messages passing. Collected data are then stored in a high performance database and a data warehouse, to allow the verification of SLAs.

In [114], the authors proposed a methodology for on-line monitoring of SLAs. In particular, the methodology requires that the SLAs are expressed in terms of an extended version of Timed Computation Tree Logic (TCTL) [115]. TCTL is a logic to reason about time and time intervals. Thus, the problem of online monitoring SLAs is reduced to the problem of checking whether or not the actual execution of the system is a (timed) word accepted by the automaton. The methodology is also supported by the SLAngMon tool [116]. SLAngMon is an Eclipse plugin to generate Java monitors for Service Level Agreements.
6.1.1 Puppet

PUPPET (Pick UP Performance Evaluation Test-bed) is an approach for the automatic generation of test-beds to empirically evaluate the QoS features of services under development in SOAs [117]. Specifically, the generation exploits the information about the coordinating scenario, a description of the interface the services export and the specification of the agreements.

The approach proposed by PUPPET is to automatically derive stubs for the externally accessed services $S_i$ from published functional and extra-functional specifications of the external services. PUPPET generates an environment (the services stubs) within which the composite service can be run and tested (see Figure 6.1).

While various kinds of testbed can be generated according to the purposes of the validation activities, PUPPET aims specifically at providing a testbed for reliable estimation of the exposed QoS properties of the System Under Test (SUT). Concerning the externally accessed services, PUPPET is able to automatically derive stub services that expose a QoS behavior conforming to the extra-functional specifications such as agreements among the interacting services [118].

Once the QoS tested is generated, the service provider may test the SUT by deploying it on the real machine used at run-time. This would help in providing realistic QoS measures preventing the problem of recreating a fake deployment platform; in particular, the QoS evaluations will also take into account the other applications running on the same machine that compete for resources with the service under test (it is worth noting that handling this case would be extremely difficult using analytical techniques).

The stubs developed include the set of operations they export and the emulation code for the extra-functional behaviors as specified in the SLA. The levels of Quality of Service can be expressed by means one of the many languages proposed in SOA such as WS-Agreement or WSLA.

Moreover, PUPPET includes a module to link each stub with code emulating the supposed functional behavior [119]. This module is optional, in the sense that is anyhow possible to
skip it and still generate working stubs that only emulate the extra-functional behavior of the real services.

The functional behavior of a service is modeled by means of the STS models as described in [119].

In the end, PUPPET generates service stubs that can be used by the testers in order to mock-up the deploying environment they would emulate. The architecture of PUPPET is structured in layered modules, whereby each module plays a specific role within the stub generation process. The detailed description of the architecture is reported in [119].

The current implementation of PUPPET is based on the Web Services technologies [120]. In particular, in [121], the description of the interface exported by services is given in WSDL [122], while the specification of the agreements are expressed by means of WS-Agreement [123]. The functional behavior of a service is modeled by means of a symbolic transition systems called Service State Machine (SSM) [124]. PUPPET includes an interface to an STS simulator [125] able to execute the SSM models.

The tool is available at http://plastic.isti.cnr.it/wiki/doku.php.

The automatic generation of a testbed environment that emulates both the functional and the extra-functional characteristics of a third-part service in a given choreography can be used in TAS³ in order to support the auditing of the service in a controlled environment as described in Chapter 5.

6.2 End-User Perceived Service Quality

Any information system architecture that aims to deliver a certain level of Quality of Service (QoS) must be based, not only on technology schemes, standards and protocols, but also on users’ perceptions.

The quality, whether in tangible products or services, is what customers perceive [126] or judge in relation to what they expected [127]. There are, however, some characteristics that differentiate services from products [128] and some specific issues in the services that make the assessment of quality by customers more problematic [129].

In the context of services market, quality deserves a treatment and conceptualization different from the quality of tangible goods. Unlike quality of products, that can be objectively measured through indicators such as length or number of defects, quality in services is something fleeting that can be difficult to measure. In fact, the intangibility of services causes that they are perceived in a subjective way.

In recent years numerous studies have worked on identifying the key attributes that consumers value to define the quality of services they buy. Three dimensions of quality have been identified: a) physical quality, including the physical aspects of the service (equipment, buildings, etc.); b) corporate quality, which affects the company’s image, and c) interactive quality, which derives from the interaction between staff and customers or customers with other customers. Grönroos [130] describes the quality of service, as a multidimensional variable formed from two major components: a technical dimension or outcome and a functional dimension or related to the process. Grönroos relates these two dimensions of quality of service with the corporate image, because image can determine what is expected of the service provided by an organization. Parasuraman et al. [131, 132] began from the conceptualization of Grönroos to develop an instrument for measuring the quality of ser-
vice: SERVQUAL. Based on an extensive group of interviews, they identified some determinants of the quality of service not necessarily independent [133]: reliability, responsiveness, competence, access, courtesy, communication, credibility, security, understanding/knowing the customer. Cronin and Taylor [134] or Babakus and Boller [135], proposed the scale SERVPERF. This scale is prepared in accordance with the same items that SERVQUAL scale, but does not incorporate the expectations measurement construct. Two alternative models are: the Evaluated Performance (EP) model and a Norman Quality (NQ) model [136]. Bolton and Drew [137] suggest that different scales of measurement must be adapted to different contexts or industries.

About measuring service quality in information systems, different studies have been carried out for developing measurement scales to appraise digital services. Similarly to earlier research on service quality in conventional distribution channels, Zeithaml et al. [138] and Parasuraman et al. [139] carried out a study on Internet service quality, from which they developed the e-SQ scale. This scale is defined as the degree to which a website facilitates effective and efficient purchasing. Cox and Dale [140] showed that traditional dimensions of service quality (e.g. competence, courtesy, clarity, comfort and friendliness) were not relevant to digital service distribution. However, other factors (e.g. accessibility, communication, credibility and appearance) were very important to being successful in these environments. Lastly, the authors of [141] identify 15 dimensions: performance, features, structure, aesthetics, reliability, storage capacity, serviceability, security and system integrity, trust, responsiveness, product/service differentiation and customization, web store policies, reputation, assurance and empathy. Yoo and Donthu [142] developed the SITEQUAL scale to measure the perceived quality of an online shop, composed by four dimensions: ease of use, aesthetic design, processing speed and security. For their part, Barnes and Vidgen [143] developed the WebQual 4.0 scale, made up of twenty-two items divided into five dimensions: usability, design, information, trust, and empathy. And lastly, Loiacono et al. [144] created the WebQualTM scale, composed by thirty-six items and twelve dimensions: informational fit to task, interactivity, trust, response time, design appeal, intuitiveness, visual appeal, innovativeness, flow (emotional appeal), integrated communication, business process, substitutability.

Despite efforts made to measure the perceived service quality in digital service distribution, it is considered that research in this area is still at an early phase [145].

In the TAS³ context will be used a modification of the scale PeSQ, proposed by Cristobal, Flavián and Guinaliu [146]. This scale will constitute the basic tool for measuring a broader conception of QoS inside of TAS³ needs. We will present this tool in the next section.

6.2.1 Perceived Service Quality Measurement

The proposed measurement instrument of perceived service quality is an extension of the so-called Perceived e-Service Quality (PeSQ) [146] originally developed by e-commerce interactions. This instrument is composed of four dimensions: interface design, support services, assurance, order management.

The design of the interface (e.g. contents layout, contents updating and user-friendliness) coincides with the proposals of previous studies [128, 147]. In the design of an interface, offering enough information is very important. Benefits perceived by IS users are the reduction in search costs, especially in information-related services [131, 148], and a large amount
of freely available information well organized and easily accessed. Other important issues of interface design are: the services offered by the system and its features are correctly presented, the information is regularly updated, the pages load and tasks are developed quickly.

Support services or customer services are key elements for achieving good results in an IS platform [149]. Users expect to be able to complete tasks correctly, to receive personalized attention, to have tailor-made services, to have the items delivered on time, to have their emails answered quickly, to have services that perform properly first time and to have access to information. IS management should ensure these expectations are met in the best way possible. The majority of the scales previously developed consider attributes related to support services [145, 137].

The assurance attribute is the term given in the services world to describe the sensation that a supplier transmits in terms of security and credibility [136]. In a digital environment, security is probably better defined when it is contemplated alongside the notion of privacy. The lack of confidence motivated by the absence of security and privacy in the digital environments is one of the main obstacles to their development. This assurance dimension includes incorporating security elements and communicating them to customers, transmits an image of reliability and trustworthiness, ensures the confidentiality of user data, confirms the successful operations, gives clear information on how to make the tasks.

Order management relates to the possibility of modifying and/or postponing the services order process at any given moment and with no obligation, and of obtaining information on service availability at the moment of order. Order management effectiveness holds considerable weight in the assessment of a service, whether conventional or online [150, 151].

Focus groups techniques or in-depth interviews are advisable approaches to determine how contextual factors influence users’ perception of QoS. Both techniques can be used with experts in TAS3 or potential users.

6.3 QoS in TAS3

QoS is today an irremissible asset of Service-oriented applications. Several approaches and tools exist for monitoring the attributes of QoS from a technical point of view. The automatic generation of a testbed environment that emulates both the functional and the QoS characteristics of a third-party service in a given choreography could be used in TAS3 in order to support the validation (via Audition) of the service in a controlled environment as described in Chapter 5. Let us consider the case of service provider who is registering a new TAS3-service \( TS^* \) in a given choreography of TAS3 services. In such a choreography, \( TS^* \) would interact with a number of other TAS3-services \( (TS_1, \ldots, TS_n) \), which may be either under direct control of the service provider, or may be are provided by third parties. Thus the binding between \( S^* \) and \( S_i \) during the auditing of the new TAS3-service \( TS^* \) might be not possible implying undesired side-effects (e.g., usage costs, undesired writes on a DB). For the sake of completeness, we admit that at the time of this writing, the project is addressing other priorities, in terms of security and privacy, than the verification of the QoS properties of a TAS3-service. However, with further investigation on the opportunities in combining the SLAs with the TAS3 architecture, the emulation of services with QoS offered by PUPPET could usefully be exploited in the next phases of the project.
From a marketing and business management perspective the study of the concept “service quality” is broader than the traditionally observed in technical disciplines. In this sense the analysis of QoS should be expanded in order to get a broader view of the different elements that make up the perceived quality, beyond the purely technical aspects. The use of measuring scales will allow to contemplate all components of the perceived service quality.

Due to the need of a broader conception of the term “service quality” we propose the use of a double perspective in the evaluation of quality in the context of TAS$^3$. So we firstly recommend a survey methodology based on the Perceived e-Service Quality (PeSQ) scale. Secondly, focus groups techniques and in-depth interviews offer us an interesting evaluation in the early stages of development.
7 Usability Testing

The TAS$^3$ system is for the largest part a set of infrastructural modules that have zero human interaction. As such, usability from the TAS$^3$ point of view is restricted to how organizations perceive the ease or difficulty to adapt their processes to TAS$^3$, and the suitability of TAS$^3$ security to their business. Industry acceptance is an important project goal.

Traditional usability research focuses on the human end user of a computerized application. This is largely outside the scope of the TAS$^3$ project. However, TAS$^3$ will need to provide suggestions to partners on how to ask security-related questions to their customers. Best practices on this topic are relevant for the project.

On the other hand, TAS$^3$’s development process must be guided by human-computer-interaction principles, that is, we must consider that our architecture will be designed, evaluated and implemented in an interactive computing framework, not only for human use but also considering the automatic or semi-automatic interactions between the computers. Thus it is critical to understand how people will interact with TAS$^3$ as well as how direct computer interactions will affect final users of the systems. Final users won’t perceive the electronic interactions between TAS$^3$’s serves (this will be a black box for them) but these interactions must not affect users’ tasks. So TAS$^3$ must be an integrated system which facilitates interactions for users and computers.

Properly applied, Human Computer Interaction (HCI) methods and techniques must identify the useful information in each stage of the project life cycle in order to facilitate a more accurate project definition and an improved project design during the design phase.

The benefits of HCI and usability activities continue after release, including increased user productivity and satisfaction, reduced training, support, and error recovery costs, increased sales and revenue, and reduced maintenance costs [152]. However, these benefits require that HCI and usability working together in order to constitute a TAS$^3$ process design centered on the user and the user tasks, rather than on technical and programming issues. Following a user-centered design process is the only way to reliably create practical, useful, usable, and satisfying technology products. In a user-centered process, you will design the user experience first, and then let that drive the technology. This design process must consider several steps questions:

- Requirements (to define the architecture and TAS$^3$ goals clearly and early on). From a usability perspective, the product definition should address who the TAS$^3$’s users are, the types of tasks for which users will use the system, and the real-world situations (the “context”) in which the product will be used;
- Design and Development: During the design and development phases, the project team needs to create a clear, high-level design for the architecture that will be implemented during development;
- Post-release: During prerelease beta-testing and after release of the final version of the TAS$^3$, the research team needs to be informed of feedback from the users who are using the architecture.
7.1 Unique Aspects of HCI and Usability in the Privacy and Security Domain

There are unique aspects in the HCI design of privacy and security systems that represent challenges and opportunities for the organizations.

Firstly, we have to consider that security and privacy are rarely the user’s main goal. Security and privacy are only a secondary element necessary to completing primary tasks like ordering medications. Users want to be in control of the situation and understand what is happening. They need that the information and the interaction methods related to security and privacy are accessible.

Secondly, due to the more intensive use of computing technology for the transmission of sensitive information, disparate types of users must be accommodated. However, security solutions have historically been designed for highly formed technical users although the functionality provided by the system for people with different roles must accommodate the skill sets of each one.

Thirdly, usability is a more important issue for security and privacy applications than for many other types of systems due to the complexity associated with this type of applications. Sophisticated systems that are badly designed, without following the HCI rules, may actually put users at more risk than if less sophisticated solutions were used. Thus, there is a great incentive to include all aspects related with interaction between users and computers, their needs and technology in systems research and development.

Finally, security and privacy solutions must easily update to accommodate frequent changes in legislation and regulations. This is especially important for very dynamic domains such as health care, banking or government.

7.2 Usability Testing Tools

This section offers an overview of the main tools usually considered by usability professionals and that will be used in the TAS context.

Depending on circumstances, testing facilities can range from a simple office setting or a hotel room to a full-blown usability testing lab. Thus, we may face a significant trade-off between the quality of the information obtained and the costs in terms of equipment, placement, etc. However, highly-skilled professionals may diminish this gap as they have the knowledge and the experience for obtaining relevant responses with no more than a conference room, few chairs and a computer. But owning a usability testing facility shows commitment to testing within the organization and gives insights about the goal, mission, etc. of the organization. This facility must provide a relaxed atmosphere, avoiding noises and disturbances to the participants. It is highly recommendable to have a one-way mirror to observe the individuals and the technological equipment, that should not be forgotten would be voice recording, video recording and, if budget available, an eye tracker mechanism that permits gaining a lot of information.

The Walqa laboratory of usability will be used in developing the usability tests in the context of TAS. This laboratory has the configuration and appropriate tools for the development of analysis at the highest level. Its location in the Walqa Technology Center facilitates communication with major cities and business centers which have potential users or experts.
necessary for the analyses.

In the laboratory works a multidisciplinary team consisting of engineers, economists and jurists. This team evaluates the usability and accessibility of applications with different methods (e.g. heuristic, analysis of logs, user-centered evaluation, etc.) depending on the circumstances of each case. Laboratory facilities have an advanced computing environment, with local network, high quality access to the Internet and servers and other resources. Besides the specific equipment Usability Lab are as follows:

1. Heuristic analyses: The laboratory has a semiautomatic tool to this kind of evaluations. It is aimed at evaluators of websites to facilitate the process and shorten the period of assessment. The tool combines two methods of assessment, automatic evaluation and heuristics. It consists of a database that contains information provided by the evaluator, completing a series of forms based on the following rules: guidance on usability; reporting; user guide; dialogue through menus; orders through dialogue; dialogue through dialogue and direct manipulation by filling in forms. The inspection of each and every one of the items for each standard guarantees adjustment. The process is based on the evaluation by a group of experts, working individually and then integrates its findings. Also, specific guidelines can be used for specific software;

2. Testing room or interview room: it is used to perform tests such as focus group and includes: 12 working places (table, chair, computer anf webcam), 1 seat for the moderator with computer; installation of PA system in the courtroom and micros to record the sound of the meetings; 3 IP cameras located on the roof and 1 located on a tripod;

3. Observation room: where members of the analyzed company and developers of the application can see (unseen) as users manage the program, noting the strengths and shortcomings in the design: 2 working-places for Laboratory technicians (table, chair, computer personnel), 4 screens to monitor the meetings, installation of PA system for tracking meetings; table meetings with capacity for 8 people;

4. Card Sorting: it is a technique used to optimize the menus in response to perceptions of users. The laboratory has a specific program (ACASO), development in-house that allows that the classification takes place directly on the computer and automatically extracts the relevant statistics;

5. TAW: TAW is an acronym for Test de Accesibilidad Web (Web Accessibility Test). It is a tool for analyzing the accessibility of websites. The system analyzes one page or website, based on the Web Content Accessibility Guidelines 1.0, and generates a report based on HTML page analysis with information on the outcome of the analysis. The report is divided into three parts: headline, web analysis (the page is analyzed by inserting warning icons on accessibility issues found) and summary;

6. Modeling Tools: most of the important usability work can be completed with a simple office suite (a flowcharting package, a graphics work, a word processor to document meetings and descriptions) or specialized utilities (e.g. Micro Saint; to work with very large task-flows). Nevertheless, we must be careful when using these mechanisms. Many companies have purchased tools to track Web sites and provide feedback and statistics on usage. They, the tools, claim to provide usability information for performing quick checks and validation but they are not relevant when trying to identify the reasons
for spending a short period of time to click on a link: good design that allows to find what we want straightforward or bad design that makes us click on the wrong place.

7.3 Usability Testing in TAS\textsuperscript{3}

The usability testing process for privacy and security technological solutions -like the TAS\textsuperscript{3} platform- can be differentiated according to the following stages:

- **Requirements Phase:** it will be especially useful analyses based on in-depth interviews and focus groups with experts and potential users of the TAS\textsuperscript{3} Platform;

- **Usability in Design and Development:** heuristic tests, surveys about perceived usability, eye-tracker analyses and users' tests are the most adequate methodologies to use in this stage;

- **Usability in Postrelease:** TAS\textsuperscript{3} users' surveys about perceived usability and satisfaction and focus groups are recommended.

This planning will vary if whatever previous elements (needs, resources) are modified. Moreover, several aspects such as the types of system's users must be considered. This usability framework is adapted to TAS\textsuperscript{3} platform in order to achieve the highest levels of satisfaction and global performance.
8 XML-based Testing

Due to its flexibility and efficiency in transmission of data, XML (eXtensible Markup Language) [153] has become the emerging standard of data transfer and data exchange across the Internet. In this chapter, we will address the main issues dealing with testing and security of XML applications. In particular, after providing in Section 8.1 a survey of the general testing strategies of XML-based applications, we will present XML-based security policy testing strategies in Section 8.2. Mechanisms dealing with XML data encryption and XML security verification are addressed in Section 4.1.2.

8.1 Testing of XML-based applications

To be able to exchange XML documents, communicating parties need to agree on a reference structure of the XML documents. This is pursued by the creation of “control documents”, which follow standardized specifications, such as Document Type Definition (DTD) [154] and XML Schema (XSD) [155].

We overview testing approaches for XML-based applications distinguishing between testing of XML documents and schema-based testing.

With the aim of developing successful applications, which can correctly cooperate, verifying the correctness and adequacy of XML documents becomes extremely important. A first essential test that must be assessed on an XML document is the well-formedness testing. If this property is not satisfied the tested document cannot be classified as an XML file. Another important test is the compliance to specified requirements, coming from different domains. The requirements can derive from a standard or consist of some specific rules defined by the developers community, which can represent the specification of input domain or other requirements of the system. Tools for the automatic verification and validation of XML documents have been implemented, such as [156] which enables unit testing of XML and [157] which develops strategies for the automated production of XML documents to facilitate the testing of web services. When a vast amount of information must be manipulated and organized, having a common reference structure is extremely important. Research on how to derive XML schema from XML instances is going on actively, and interesting results have been produced [158, 159, 160, 161]. Recent applications of these technologies are [162, 163], which extract schema using some graphs according to the frequency of element occurrence in XML documents and [164] in which the authors represent, by using the XML markup, a text type schema definition of the structure of the scientific paper. In parallel with these approaches, another field of research, also called instance-level matching [165], tries to derive a common structure by dynamically analyzing the diverse XML elements and extracting from time to time the proper schema structure. For implementing such kind of analysis different proposals have been adopted such as rules, neural networks, and machine learning techniques, [166, 167, 168].

The approaches dealing with XML schema based testing can be distinguished in

- Well-formedness of XML Schema;
- Verifying XML documents against the schema structure;
Several XML schema validators for checking the syntax and the structure of the W3C XML Schema document are available. Among them, there are SQC (Schema Quality Checker) [169], XSV (XML Schema Validator) [170], and XML Spy5 [171]. Recently an interesting approach has been proposed by [172], which detects semantic errors in XML schema by using mutation analysis. Other tools verify the XML documents against its DTD or XML Schema [173, 174, 175, 176].

Many tools are developed for the automated XML instance generation, based on DTD or XML schema. Most of them generate the instances randomly. The disadvantage of random generation tools is that instances cannot cover all possibilities of the schema. An emerging and innovative research field is thus the application of traditional testing strategies for generating suites of XML instances from the XML Schema structure. In this area two proposal can be mentioned:

- **Specification Based Testing**: it relies on the structural properties derived from the program specifications and different techniques can be used for guiding the selection of test data [177]. In Section 8.1.1 we will present the TAXI tool where the application of equivalence partitioning is adopted for the generation of XML instances from XML schema. This testing strategy relies on the partitioning of the input domain into subdomains so that any input within a subdomain can be taken as a representative for the whole subset.

- **Adopting Perturbation Testing**: in the direction of using commonly adopted testing strategies for guiding the XML based testing, another interesting work is [178] that presents a new approach to testing Web services. The authors, taking into account the approach in [179] which presents a technique for using mutation analysis to test the semantic correctness for XML based component interactions, consider communication infrastructure of web services, typically XML and SOAP, and develop a new approach for testing them based on data perturbation.

### 8.1.1 TAXI tool

TAXI (Testing by Automatically generated XML Instances), is a proof-of-concept tool for the systematic generation of XML instances. Such tool inputs an XML schema and automatically generates a set of XML instances for the black-box testing of a component whose expected input conforms to the taken schema. TAXI implements the XML-based Partition Testing (XPT) methodology.

The XPT methodology is composed by two different phases: XML Schema Analysis (XSA) and Test Strategy Selection (TSS). The former implements a methodology for analyzing the constructs of the XML schema and automatically generating instances. The latter implements different test strategies useful for both selecting the parts of the XML schema to be tested and opportunely distributing the instances on the schema [180]. These phases work in agreement, to realize the application of the Category Partition (CP) testing technique [181], which constitutes the overall basis on which the whole XPT approach relies. The XML schema leads quite naturally to the application of the Category Partition, since it provides an accurate representation of the input domain. In particular, the subdivision of the input domain into functional units and the identification of categories can be done by exploiting the
formalized representation of the XML schema. In order to reduce the number of combinations generated by CP, TAXI also integrates a set of weighted test strategies. The user can either fix the number of generated instances or distribute them according to defined importance factors (weights). After reading the input XML schema, there is the weight assignment phase where the user can modify the default weights assigned to the children of choice elements, expressing the relative “importance” of a child with respect to the other children of the same choice. Weights belong to the \([0,1]\) interval and the sum of the weights associated to all the children of the same choice element is equal to 1. Greater weight corresponds to more critical node. There is a database containing the values for the instances derivation. The user can populate the database by inserting the values by means of TAXI interface.

The XPT methodology proceeds then with the Preprocessor activity in which the other XML schema constructs, like all, simpleType, ComplexType, attributeGroup, ref, and so on are analyzed and manipulated. The next activity is then Choice Analysis which extracts for any alternative within a choice construct, a separate sub-XML schema containing it. In case of several choices within one schema, as many subschemas as number of the possible combinations of the children of the choice nodes are produced. Once completed this operation and derived the set of possible substructures, using the final weights of the leaves in each substructure, it is possible to derive unique values, called subtree weights, useful for test strategy selection activity, for properly deriving the set of final instances. Specifically three different test strategies are implemented: applying TAXI with a fixed number of instances, applying TAXI with a fixed functional coverage, applying TAXI with a fixed functional coverage and a number of instances. As example, Figure 8.1 shows the interface of TAXI for the test strategy selection.

![Figure 8.1: TAXI interface for test strategy selection.](image)

The implementation of the Category Partition methodology proceeds with the activity called Occurrences Analysis which analyzes the occurrences declared for each element in the subschema and, applying a boundary condition strategy, derives the border values (minOccurrences and maxOccurrences) to be considered for the final instances generation. The last activity of the XPT methodology is the Final Instance Derivation. Accordingly with the test strategy selected and using the values in the database or randomly generated, the final set of instances, which is by construction conforming to the original schema, is gener-
Within TAS³ such tool can be used for automatically generating a set of XML instances for the black box testing of a component, whose expected input conforms to the taken XML schema. Finally, TAXI could be improved with testing strategies for the derivation of invalid instances useful for the verification of robustness of the TAS³ system.

8.2 Web Services security policy testing

Access control is one of the most fundamental and widely used security mechanisms, especially in web applications. Access control mechanisms verify which users or processes have access to which resources in a system. To facilitate managing and maintaining access control, access control policies are increasingly written in specification languages such as XACML (eXtensible Access Control Markup Language)\[29\], a platform-independent XML-based policy specification language, or RBAC (Role-Based Access Control) \[182\]. Whenever a user requests access to a resource, that request is passed to a software component called Policy Decision Point (PDP). A PDP evaluates the request against the specified access control policies and permits or denies the request accordingly.

Policy-based testing is the testing process to assure the correctness of policy specifications and implementations. By observing the execution of a policy implementation with a test input (i.e., access request), the testers may identify any faults in policy specifications or implementations, and validate whether the corresponding output (i.e., access decision) is as intended. Although policy testing mechanisms vary because there is no single standard way to specify or implement access control policies, in general, the main goals to conduct policy testing are: assure the correctness of the policy specifications and assure the conformance between the policy specifications and implementations.

The authors of \[183\], classify recent approaches on XACML policy specification testing in the following main categories:

- Fault Models and Mutation Testing: there exist various basic fault models \[184, 185\] for different types of policies. Martin and Xie \[184\] propose a fault model to describe simple faults in XACML policies. They categorize faults broadly as syntactic and semantic faults. Syntactic faults are the result of simple typos. For example, in XACML, an XML schema definition (XSD) can be used to check for obvious syntactic flaws. Syntactic faults that do not violate the XSD can occur due to typos in attribute values. Semantic faults are involved with the logical constructs of the policy language. Based on the fault model, mutation operators are described in \[184\] to emulate syntactic and semantic faults in policies. Traon et al. \[185\] design mutation operators on a given Organization Based Access Control (OrBAC) model. They consider similar semantic and syntactic mutation operators as the preceding ones in addition to other mutation operators including rule deletion, rule addition, and role change based on a role hierarchy.

- Testing criteria: testing criteria are used to determine whether sufficient testing has been conducted and it can be stopped, and measure the degree of adequacy or sufficiency of a test suite. Among testing criteria for policy testing, there are structural coverage criteria and fault coverage criteria. The first ones are defined based on observing whether each individual policy element has been evaluated when a test suite
(set of requests) is evaluated by a PDP [186]. The second ones are defined based on observing whether each (seeded) potential fault is detected by the test suite. Fault coverage is commonly measured with mutation testing techniques.

- Test generation: To test access control policies, policy testers can manually generate test inputs (i.e., requests) to achieve high structural policy coverage and fault coverage (i.e., fault-detection capability). To reduce manual effort, automated test generation approaches can be used. Three main types of automated test generation approaches for policy testing can be adopted: random test generation, test generation based on change-impact analysis, and test generation for model-based policies. For XACML policies, Martin [187] develops an approach for random test generation. The approach analyzes the policy under test and generates requests by randomly selecting requests from the set of all possible requests. Fisler et al. propose Magrave, a tool for analyzing XACML policies and performing change-impact analysis [188]. The tool can be used for regression testing to identify the differences between two versions of the policies and test the PDP. Xie et al. propose the tool Cirg that automatically generates test for XACML policies using change impact analysis [189]. Test generation for model-based policies derives abstract test cases directly from models such as FSMs that specify model-based policies. Traon et al. [185] propose test generation techniques to cover rules specified in policies based on the OrBAC [190] model. The same authors in [191] present a new automated approach for selecting and adapting existing functional test cases for security policy testing. The method includes a three-step technique based on mutation applied to security policies and AOP for transforming automatically functional test cases into security policy test cases.

For the sake of completeness, we also address in this section some works dealing with policy implementation testing. In particular, the authors of [192] propose an approach to detect defects in XACML implementations via observing the behaviors of different XACML implementations (policy evaluation engine or PDP) for the same test inputs, whereas [193] concerns the performance of XACML request processing and presents a scheme for efficient XACML policy implementation.

### 8.3 XML-based approaches in TAS³

XML has become the emerging standard of data exchange in SOA Web Services. We have provided a survey of the different testing approaches for XML applications with a particular emphasis for that ones concerning security issues in web services. In particular, we addressed strategies for testing access control mechanisms in web applications.

We have introduced the tool TAXI, that systematically derives XML instances from a XML Schema, based on the Category Partition method. Within TAS³ the applications of TAXI may be manifold. Currently our efforts are focused on the verification and evaluation of the correct interoperability of the demonstrators using common repositories for data exchange.

The formal data representation (object and personal data) is currently under specification within TAS³. Some of the partners are currently selecting and specifying XML Schemas to be adopted for this purpose. These schemas will define relations between data objects. These relations, which might also include specific annotations, should be respected in the final
instances. Within TAS\textsuperscript{3}, TAXI could be specified for creating particular instances conforming to the relations expressed by ad hoc in line comments in XML Schemas.

Finally, TAXI could be improved with testing strategies for the derivation of invalid instances useful for security testing purposes of the TAS\textsuperscript{3} database. With access policies specified in XACML we can investigate the adaptation of TAXI to automatically derive the test suite used by OCT (see Chapter 5).
## Document Control

### Amendment History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Description/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2008-05-14</td>
<td>Guglielmo De Angelis</td>
<td>Initial document structure</td>
</tr>
<tr>
<td>0.2</td>
<td>2008-05-22</td>
<td>Antonia Bertolino</td>
<td>Revised structure and assigned responsibility</td>
</tr>
<tr>
<td>0.3</td>
<td>2008-07-02</td>
<td>Antonino Sabetta, Antonia Bertolino</td>
<td>Reformatted according to TAS(^3) template, included more details on due contents</td>
</tr>
<tr>
<td>0.4</td>
<td>2008-07-11</td>
<td>all</td>
<td>Definition of a preliminary version of Part 1 and a Draft of Part 2</td>
</tr>
<tr>
<td>0.5</td>
<td>2008-07-25</td>
<td>all</td>
<td>Finalization of Part 1 and preliminary version of Part 2</td>
</tr>
<tr>
<td>0.6</td>
<td>2008-08-25</td>
<td>all</td>
<td>Finalization of Part 2</td>
</tr>
<tr>
<td>0.7</td>
<td>2008-08-29</td>
<td>all</td>
<td>Initial review according to comments provided during the WP10 call of August 25th</td>
</tr>
<tr>
<td>0.8</td>
<td>2008-08-25</td>
<td>Antonia Bertolino, Carlos Flavian, Jeroen Hoppenbrouwers</td>
<td>Finalization of prefaces of each chapter, completion of executive summary, keywords list, Abstract and general revision</td>
</tr>
<tr>
<td>0.9</td>
<td>2008-09-12</td>
<td>all</td>
<td>Final version ready for review</td>
</tr>
<tr>
<td>1.0</td>
<td>2008-09-29</td>
<td>all</td>
<td>Final delivered version</td>
</tr>
<tr>
<td>1.1</td>
<td>2009-03-30</td>
<td>Antonia Bertolino, Francesca Lonetti</td>
<td>New Table of contents after review</td>
</tr>
<tr>
<td>1.2</td>
<td>2009-04-20</td>
<td>all</td>
<td>Complete draft for internal review</td>
</tr>
<tr>
<td>1.3</td>
<td>2008-05-15</td>
<td>all</td>
<td>Advanced complete version for discussion, revisions to Chapters 3 and 7 still pending</td>
</tr>
<tr>
<td>1.4</td>
<td>2008-05-18</td>
<td>all</td>
<td>Advanced complete version for discussion, including revisions to Chapters 3 and 7</td>
</tr>
<tr>
<td>1.5</td>
<td>2008-05-25</td>
<td>Antonia Bertolino, Guglielmo De Angelis, Francesca Lonetti</td>
<td>Final version for the review</td>
</tr>
</tbody>
</table>
Bibliography


[103] Eviware, “soapUI: the web services testing tool.”


[156] XMLUnit, “XMLUnit-JUnit and NUnit testing for XML.” http://xmlunit.sourceforge.net/.

Version 1.5


