D45.1 – Requirements Report

Server Testbed

Document Identification	
Date	27/05/2013
Status	Final
Version	0.6

Related SP / WP	SP4 / WP45	**Document Reference**	https://dms-prext.fraunhofer.de/livelink/livelink.exe/overview/3012742
Related Deliverable(s)		**Dissemination Level**	PU
Lead Participant	USTUTT	**Lead Author**	Eray Özmü
Contributors	Christopher Ruff, Dr. Lothar Fritsch, Monika Drabik	**Reviewers**	AGETO, ATOS

This document is issued within the frame and for the purpose of the *FutureID* project. This project has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318424

Document Information

Contributors

Name	Partner
Eray Özmü	USTUTT
Christopher Ruff	USTUTT
Dr. Lothar Fritsch	NRS
Monika Drabik	CA
Tomasz Grabowski	CA

History

Version	Date	Author	Changes
0.1	17.01.2013	Eray Özmü	Created Outline
0.2	20.05.2013	Eray Özmü, Christopher Ruff, Monika Drabik, Lotar Fritsch	Contribution on the wiki
0.3-NRS	23.05.2013	Lothar Fritsch	Additions, esp. in test requirements
0.5	27.05.2013	Eray Özmü	Added Abstract and Conclusion
0.6	27.05.2013	Eray Özmü, Christopher Ruff	Finalized document.

Table of Contents

Abstract

The server infrastructure of *FutureID* needs to satisfy different requirements which are defined in the deliverables D41.1, D42.1, D43.1, D43.2, D44.1 and D44.2. This deliverable will ensure that the right requirements for the server testbed will be defined to ensure thorough testing and a clear test process for all participants.

Subject of this deliverable is only the server side of the *FutureID* infrastructure. The client side will be subject of WP3.7. As the client testbed already covers system and acceptance tests, these test levels are not subject of this deliverable either. It is intended to set up a test environment for all server components. Based on this test environment different kinds of tests should be performed and integrated within the continuous integration platform Jenkins, which is part of deliverable D37.2.

Test Assertion Markup Language (TAML) has been considered as a way to define test assertions, but was found to have several disadvantages. After the evaluation of alternative strategies, it is intended to use an informal and light-weight way to define test assertions and test cases on a central wiki and connect these test cases to certain tests by defining a certain ID for each test case.

1. Introduction

The *FutureID* project aims to build a sophisticated server infrastructure to integrate with the *FutureID* client in order to ensure secure, privacy-friendly and usable authentication with all different kinds of tokens. The development team works in a decentralized and geographically separate manner. To be able to ensure a high quality of the server infrastructure a server testbed needs to be created. This server testbed will help the developers and testers of the project to evaluate the developed software and ensure its quality.

This document will lead to the foundation of the server testbed, as it will define the underlying requirements and also will give recommendation on the tools for the server testbed. Furthermore Test Assertion Markup Language will be evaluated as a way to define test assertions.

As it is also important to coordinate testing activities, the w-model will be introduced and the testing approach will be derived from this model. In this case, the testing will be bottom-up, where the small units of software get tested first. All requirements will be summarized in chapter 5.

1.1 Scope and Limitations

Figure 1 shows the high level architecture of the *FutureID* client-server infrastructure. As this deliverable is only concerned with the server testbed, only the part within the red rectangle will be discussed. The testing of the *FutureID* client is discussed in work package 37.

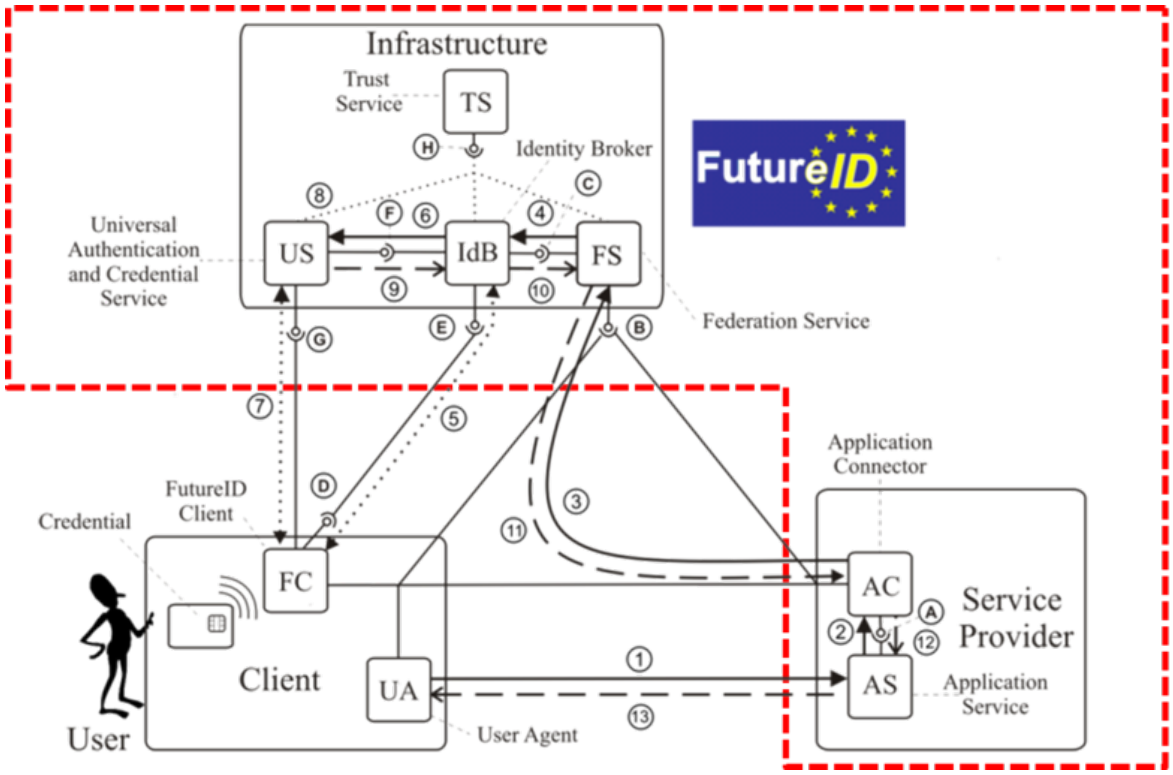

Figure 1 - FutureID Architecture

The high level goal of the server testbed is to provide the right toolset for developers and testers of this project to fulfill thorough testing and to evaluate if the software meets the requirements defined by the different stakeholders. It is not intended to provide testing of the whole infrastructure within this work package or to define all in-depth requirements for the testing activities. Instead this task will provide general requirements for the server testbed to be able to choose the right toolset and define the right processes for testing. Furthermore recommendations will be created to simplify testing for all stakeholders.

1.2 Challenges

There are certain challenges involved when testing a server-oriented, distributed system such as the *FutureID* infrastructure in contrast to software solely installed and tested locally. For one, setting up such a complex infrastructure for testing is more challenging than installing local software and running it. Another challenge is introduced by the network communication which can be unpredictable, as network packets can arrive in different order, be delayed for some time or not arrive at all. So in this scenario, we have to deal with asynchronous network queries and responses which make it harder to track the progress of tests and verify assertions. *FutureID* will also be designed for different use-cases and thus has to be as flexible as possible. While for most use-cases all parts of the infrastructure will be used, there are some degenerate use-cases, where only parts are required. So testing all the functionality needed for the many possible use-cases and various usage of the infrastructure will be another challenge.

2. Testing Approach and Strategy

A major part of the testing approach and testing strategy are the tools and methods that will be used. This chapter is divided into two parts. The first part is concerned with the evaluation of a formal language for test assertion specification and definition. The second part describes the general approach and model as well as the testing process, including recommendations for documentation to be produced.

2.1 Evaluation of TAML

Test assertion markup language (TAML) was introduced by the former partner "trustable" that has since been replaced by the University of Stuttgart. This partner proposed the use of TAML for the specification of test cases. TAML is a XML based notation for specifying test cases created by the OASIS consortium[1].

Our evaluation of TAML brought up some difficulties, which will be explained within this chapter more deeply.

The first thing one will notice while evaluating TAML is, that there is almost no literature concerning the use and existence of TAML. There is just the specification available online. No papers which are evaluating the standard or business cases are available.

Furthermore, there is almost no toolset for the use of TAML. There is only one tool available which hasn't been updated for a long time. "TAMElizer", a tool for verifying structural and semantic constraints in TAML files, was last updated in 2011.[2] Other than that, there are no additional tools available for TAML.

Because of the low adoption rate of TAML in daily business, the lack of resources about TAML and the few available tools we decided to use a more practical approach for the definition of test cases and test assertions. The test cases will be documented on the wiki and the test assertions will be defined within the tool which will be used for automatic testing. A good example on documentation of test cases can be found here.[3]

A light-weight way of documenting of test cases must be found. The test strategy (deliverable D37.2) has a first approach of documenting test cases and assertions. This approach should be considered in the following tasks of test assertions and test case documentation.

---

[1] https://www.oasis-open.org
[2] https://code.google.com/p/tamelizer/downloads/list
[3] http://users.csc.calpoly.edu/~jdalbey/206/Assign/testcasePizza.html

2.2 Testing approach

At the beginning of every software project, the right development approach and testing approach needs to be defined. This will lead to a better understanding on what tasks are to be worked on and when, having a clear process to follow. In deliverable 37.1 the general testing approach for *FutureID* has been evaluated and defined. Deliverable 23.2 was concerned with the development guidelines and therefore worked on different development models. In this deliverable the use of the W-Model is recommended.

The W-Model is a common development model which is illustrated in Figure 2. As you can see in the illustration, the development model is structured in different levels. The execution of the tests, which is located on the right side of the figure, follows the bottom-up approach. First the lowest level of testing will be executed. If there is an error on this level, the code gets debugged and changed. If the tests on the certain level get passed, the next level of testing will be executed and so on.

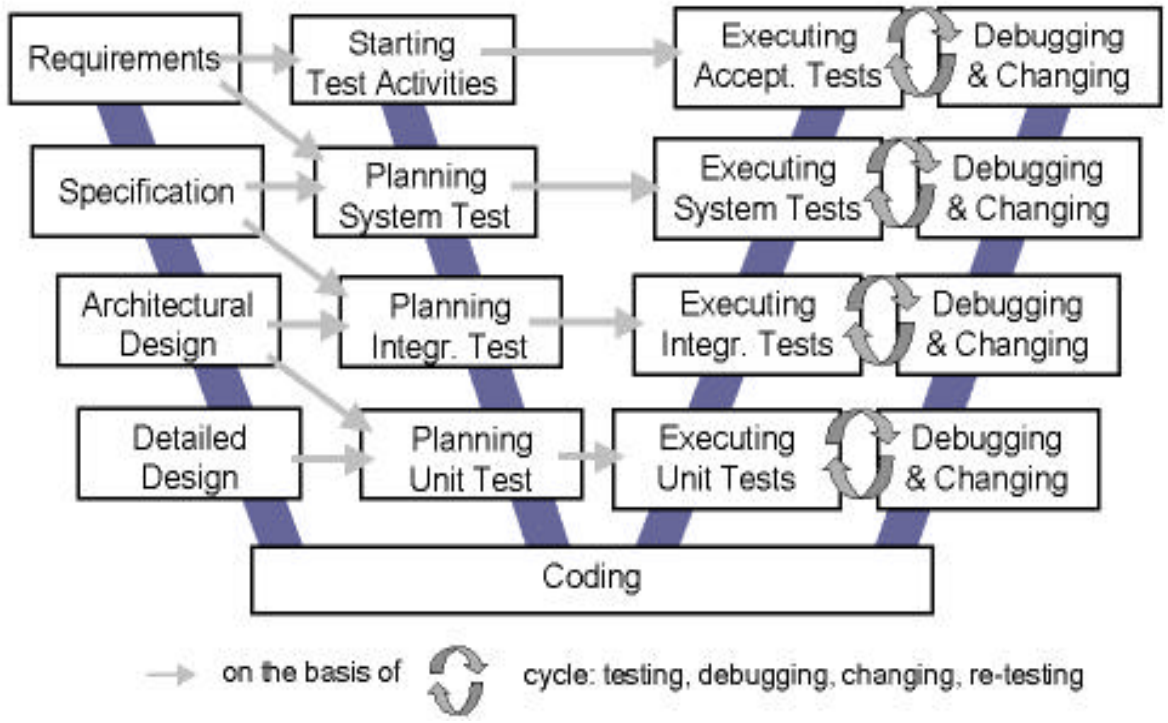

Figure 2 - W-Model

This approach will be followed for the server testbed as well as the client testbed, which is subject of work package 37.

As described in Figure 2, there are four levels of testing. The lowest level describes unit tests, where small modules get tested on code level. The next level is described as integration testing, where the integration of different modules gets tested. System tests and acceptance tests are the highest level of testing and also the highest level of abstraction. Here, the underlying code won't be inspected. The developed software will be executed under real conditions and tested.

When looking at deliverable D37.1 one can see, that system tests are already covered by the client testbed. These tests include the server connection as well. The same is for the highest test level, which is acceptance test. Therefore this document will focus on the interfaces and certain functions of the server. There won't be any system tests, or acceptance tests within this work package.

The system to be tested should be measured by its compliance to the requirements and the user acceptance criteria. Each requirement and acceptance criteria must be mapped to specific test plans that validate and measure the expected results for each test being performed. The objectives should be listed in order of importance and weighted by risk.

2.3 Test process and documentation

Creating and maintaining a proper documentation is crucial for managing a reliable and standardized testing process. The standard IEEE 829 (Standard for Software and System Test Documentation) proposes to produce a certain set of documents for each phase of testing. Depending on resources, tasks and other variables not all of them have to be produced. Some documents can be omitted or combined with other documentation to fit the requirements of a specific testing environment. Test documentation and details on how to comply with IEEE 829 is also described in *FutureID* deliverable 37.1 (Client Testbed - Requirements) and deliverable 22.1 (Documentation Guidelines) which are a useful reference. As the test documentation according to IEEE 829 is generic and not domain specific, it can be applied to the client testing process as well as to the server testing process.

In IEEE 829 also compliance levels have been introduced. These compliance levels classify the test cases in regard of importance for the software. Test cases which evaluate functions with high failure impact are grouped into compliance level 4. Minor effects are grouped within the lowest compliance level 1. This classification of test cases into different compliance levels should be available for all testing activities.

The responsible persons who are concerned with definition of test cases should always consider the compliance levels and group the developed test case accordingly.

Description	Level
Software must execute correctly or grave consequences (loss of life, loss of system, economic or social loss) will occur. No mitigation is possible	4
Software must execute correctly or the intended use (mission) of system/software will not be realized causing serious consequences (permanent injury, major system degradation, economic or social impact). Partial to complete mitigation is possible.	3
Software must execute correctly or an intended function will not be realized causing minor consequences. Complete mitigation possible	2
Software must execute correctly or intended function will not be realized causing negligible consequences. Mitigation not required.	1

Table 1 - Integrity levels

For information security and information privacy testing, concrete objectives and test criteria must be defined. They will be gathered from requirements and specification documents in *FutureID*, however to create tool-based tests, some more detailed information from the developers and their cooperation

concerning test interfaces will be necessary. Security-critical functions will need a good specification of test cases and input/output criteria for a successful test.

3. Requirements

3.1 General Requirements

ID	Derived from	Description	Importance (MUST, SHOULD; MAY)
R1.1		The server testbed must provide tools to test basic request/response automatically.	MUST
R1.2		The server testbed must be integrated to the continuous integration platform provided by WP37.	MUST
R1.3	D41.1 – 2.2	The server testbed should provide the possibility to test the integrity and confidentiality of messages exchanged by the Identity Broker.	SHOULD
R1.4	D41.1 – 2.3	The server testbed should provide different scenarios of preferred authentication services in order to test the dynamic selection of authentication servers of the identity broker.	SHOULD
R1.5		The server development team must provide a list of software and hardware that is to be tested	MUST
R1.6		The software and platform configurations to be tested must be provided	MUST
R1.7		Documentation to design, features relevant for testing, and documentation must be provided	MUST
R1.8		If necessary, all relevant hardware involved in testing must be provided.	MUST

3.2 General Security Requirements

ID	Derived from	Description	Importance (MUST, SHOULD; MAY)
R2.1		Functions/Classes shall implement log levels with logging of calls, parameters and results in log files used for test analysis. Calls of own code and calls into external modules shall get logged. Log messages shall contain information that is easy relatable to specification documents. Log files shall consist of tabular text file lines to accommodate automatic processing. They shall be timestamped.	MUST
R2.2		Access to data base entries. For verification of operations, the test activities shall be able to access database entries for all data used in *FutureID* server transactions (e.g. in the TrustService).	MUST
R2.3		Specific test criteria, input and output parameters and pass/fail criteria for the security functions shall be provided by specification and documentation	MUST

3.3 Automated vulnerability testing requirements

ID	Derived from	Description	Importance (MUST, SHOULD; MAY)
R3.1		The testbed should use automated vulnerability scanning against the *FutureID* server platform.	SHOULD
R3.2		The testbed should deploy various distributed denial-of-service attacks on *FutureID* services.	MAY
R3.3		The testbed should facilitate IP spoofing, DNS spoofing, session capture and other attack patterns to inject false web sites, redirect traffic, and fool user clients or other communication partners.	SHOULD
R3.4		The testbed should be open to include new attacks that get known under the testbed operation phase, e.g. from CERT vulnerability reports.	MAY

3.4 Server communication testing requirements

ID	Derived from	Description	Importance (MUST, SHOULD; MAY)
R4.1		Complementary to the D37.1 specification for the client testbed, the server API and communications protocols must be ready for client interaction testing. The server should be prepared for simulated or real client interaction from automated testing tools.	MUST
R4.2		The testbed shall be able to simulate business applications / relying party protocols to perform transactions.	MUST
R4.3		The testbed needs access to all access credentials necessary to connect to and run the *FutureID* components, e.g. client certificates, smart cards, key material, authentication keys against the applications and IdPs).	MUST

3.5 Security protocols testing

ID	Derived from	Description	Importance (MUST, SHOULD; MAY)
R5.1		The testbed needs access to all access credentials necessary to connect to and run the *FutureID* components, e.g. client certificates, smart cards, key material, authentication keys against the applications and IdPs).	MUST
R5.2		Repeatability of tests. The test activity needs possibilities to repeatedly make the same intentional mistake to provoke blocking of certificates, termination of sessions or authentication efforts without getting permanently locked out of services. Possibly, a function to re-activate test credentials, accounts etc. would be a feasible solution.	MUST

3.6 Test environment requirements

ID	Derived from	Description	Importance (MUST, SHOULD; MAY)
R6.1		Test Environment may allow for easy versioning of the Test Scenarios configuration	MAY
R6.2		Testing Environment should allow for defining dependencies between Test Scenarios. E.g.: Test Scenarios can be executed in parallel, Test Scenarios can be executed as cascade, Test Scenarios should wait for result of other Test Scenarios, etc.	SHOULD
R6.3		Testing Environment should allow for easy running of selected Test Scenarios	SHOULD
R6.4		Test Environment should allow for easy scheduling of selected Test Scenarios	SHOULD
R6.5		Test Environment should have GUI interface	SHOULD

R6.6	Testing Environment should have the possibility to read test data from XML files	SHOULD
R6.7	Testing Environment may have the possibility to read test data from xls files	MAY
R6.8	Testing Environment may have the possibility to read test data from a database	MAY
R6.9	Testing Environment should have allow to store all test data (source data and expected results) in one/common data source	SHOULD
R6.10	Test Environment should allow for implementation of Comparator which will be responsible for checking if results of Web Service execution are consistent with expectations	SHOULD
R6.11	Test Environment should have possibility of parallel processing/execution of Data Processing Elements	SHOULD
R6.12	Test Environment should allow for creation of reusable "processing blocks"	SHOULD
R6.13	Test Environment should have possibility of reports creation containing results of each Test Scenario	SHOULD
R6.14	Test Report must contain list of errors/exceptions encountered during test execution	MUST
R6.15	Test Environment may allow for measuring of the execution time for each Data Processing Element	MAY
R6.16	Test Environment should allow for creation of Data Processing Elements using script language (Groovy, Python, JavaScript, Ruby)	SHOULD
R6.17	Test Environment has to be able to parse of XML format (serialization/deserialization)	MUST
R6.18	Test Environment may have a possibility to publish Test Results on a web page	MAY
R6.19	Test Environment should have possibility to send emails containing Test Reports	SHOULD

4. Conclusion

In this document different types of requirements for the server testbed have been defined and listed. There are general requirements which ensure the right approach for testing. Furthermore there are specific requirements like security requirements, automated vulnerability testing requirements, server communication testing requirements and test environment requirements.

In the following tasks, these requirements will be used as foundation for the decision of tools and types of tests which need to be created as well as for the general test process. As a first approach, derived from the requirements stated above, the SoapUI testing tool in combination with the existing Jenkins platform will be used. However, this tendency must be evaluated furthermore in the following tasks within this work package.