Project Acronym	**Fed4FIRE**
Project Title	**Federation for FIRE**
Instrument	**Large scale integrating project (IP)**
Call identifier	**FP7-ICT-2011-8**
Project number	**318389**
Project website	**www.fed4fire.eu**

D7.5 – Report on second cycle developments regarding trustworthiness

Work package	WP7
Task	Task 7.2, T7.3, T7.4
Due date	31/03/2015
Submission date	27/03/2015
Deliverable lead	Steve Taylor (IT Innovation)
Version	1.4
Authors	Steve Taylor, Vadim Krivcov, Michael Boniface (IT Innovation) Elena Garrido Ostermann, Javier García Lloreda (ATOS) Chrysa Papagianni, Aggelos Kapoukakis , Costas Yiotis, George Androulidakis (NTUA)
Reviewers	David Margery (INRIA)

Abstract	This deliverable has reported on experiences and developments in cycle 2 in three main areas, corresponding to the development tasks in WP7. • In T7.2, we have deployed the access control Policy Decision Point (PDP) in a number of testbeds and demonstrated that it addresses the major requirements that guided its design. • In T7.3, we have developed the SLA management framework further in a number of areas. • In T7.4, we have further developed and implemented the Reputation Service, and designed and implemented a reputation-based trust framework for federated testbeds, named the "Federated Trust and User Experience" (FTUE) Framework.
Keywords	Trust, Reputation, SLA, Security, Certificate, Credential, Access Control

Nature of the deliverable	R	Report	X
	P	Prototype	
	D	Demonstrator	
	O	Other	
Dissemination level	PU	Public	X
	PP	Restricted to other programme participants (including the Commission)	
	RE	Restricted to a group specified by the consortium (including the Commission)	
	CO	Confidential, only for members of the consortium (including the Commission)	

Disclaimer

The information, documentation and figures available in this deliverable, is written by the Fed4FIRE (Federation for FIRE) *– project consortium under EC co-financing contract FP7-ICT-318389 and does not necessarily reflect the views of the European Commission. The European Commission is not liable for any use that may be made of the information contained herein.*

Executive Summary

This deliverable has reported on experiences and developments in cycle 2 in three main areas, corresponding to the development tasks in WP7.

In T7.2, we have deployed the access control Policy Decision Point (PDP) in a number of testbeds and demonstrated that it addresses the major requirements that guided its design. For each deployment, varying degrees of customisation were required in order to fit in with the existing resource allocation components at the testbed concerned. Initially the customisations were incorporated into the main body of the PDP code, because they added new functionality. However, it was quickly realised that the deployment-specific customisations needed to be separated from the main body of the PDP code. In addition, the PDP was complex to install and configure. As a result of these experiences, it was decided to conduct an analysis of usability, generality and scalability and derived requirements for cycle 3, and these requirements are described in this deliverable. How to address the technical requirements in this set is discussed in D7.4 (specifications for cycle 3), which is issued concurrently with this deliverable.

In T7.3, we have developed the SLA management framework further in three areas. The SLA Management module was improved during cycle 2 to include the SLA creation and evaluation functionality, as well as performance and security related enhancements. The SLA Collector has also been updated. Previously in cycle 1, it had a basic proxy function, and this has been enhanced to allow all SLAs for an experiment to be retrieved securely. The SLA plugin for the Fed4FIRE portal has been updated to allow experimenters to accept SLAs offered by testbeds and visualise SLA evaluation results at the end of the experiment.

In T7.4, we have further developed and implemented the Reputation Service, and designed and implemented a reputation-based trust framework for federated testbeds, named the “Federated Trust and User Experience” (FTUE) Framework. The FTUE algorithm bases its principles on the ROCQ scheme. ROCQ was originally designed for peer-to-peer systems, and therefore several modifications were implemented in order to adapt it for a federated testbed environment, resulting in the FTUE. The FTUE framework combines the perceived user experience, as provided by user feedback, with the monitoring data retrieved for the respective experiments.

Some sections of this deliverable provide requirements for the work described in D7.4, so ideally this deliverable should be read before D7.4.

Acronyms and Abbreviations

AM	Aggregate Manager
Authn	Authentication
Authz	Authorisation
CA	Certificate Authority
CMS	Content Management System
CRUD	Create, Read, Update, Delete
CSR	Certificate Signing Request
DN	Distinguished Name
FRCP	Federated Resource Control Protocol
GID	Globally Unique Identifier
GUI	Graphical User Interface
HRN	Human-Readable Name
IdP	Identity Provider
MOS	Mean Opinion Score
OMF	Orbit Management Framework – a reference implementation of FRCP
OML	ORBIT Measurements framework and Library
PDP	Policy Decision Point
PEP	Policy Enforcement Point
QoE	Quality of Experience
QoS	Quality of Service
RAG	"Red-Amber-Green" – status indicators for components operational state for first level support
RC	Resource Controller (in OMF)
REST	Representational State Transfer
ROCQ	Reputation, Opinion, Credibility, Quality
SA	Slice Authority
SFA	Slice Federation Architecture
SLA	Service Level Agreement
URN	Uniform Resource Name
UUID	Universally Unique Identifier

Table of Contents

1 Introduction

This deliverable reports on experiences and developments in cycle 2 in three main areas, corresponding to the development tasks in WP7. These concern the access control Policy Decision Point (PDP) in T7.2, the SLA management framework in T7.3, and the Reputation Service in T7.4. Each task is discussed in a separate section of this deliverable.

This deliverable is delivered early according to the most recent deliverable due date schedule. The main reason for this is that in many cases this deliverable gives context and reasons for developments in cycle 3, and these reasons are based on experiences from cycle 2. The specification for developments in cycle 3 is in deliverable D7.4, which originally was due before this deliverable (D7.5), but since cycle 3's specification is consequent on this deliverable's contents, it was decided that D7.5 needed to be delivered concurrently with, rather than after, D7.4. Consequently it is recommended that this deliverable is read before D7.4.

There are a number of appendices to this deliverable, mainly describing REST interfaces and APIs to the components reported in the main body of the deliverable.

2 PDP Development, Deployment Experiences and New Requirements

During cycle 2, the PDP was implemented as described by the specification in D7.2, and deployed at a number of sites. The major work in cycle 2 was performing deployments, which in some cases required customising the basic PDP as necessary to suit each deployment. This section describes these deployments, together with the experiences deriving from them. The major conclusion is that the PDP, along with the use of the SFA slice credential adopted by the project, enables federation authorisation, which was the main requirement for the project identified in D7.1 and described in D7.2. The deployments are described in section 2.1, and a description of the federation authorisation is in section 2.2.

As a result of the developments and deployments, some significant lessons were learned, and these are discussed in section 2.3. The main conclusion from these lessons was that there was need for a review of the PDP codebase. This was an analysis of its usability, scalability and general applicability. The conclusions from this analysis are presented in section 2.3 as a set of requirements, which will be addressed in cycle 3.

2.1 Deployments

2.1.1 BonFIRE

The PDP is deployed in BonFIRE's integration platform, and specialist scripts have been created to accommodate the differences between BonFIRE's resource management approach and those of the SFA. BonFIRE uses the concept of experiments, and in order to map the experiments to slices special scripts were created for Authorised Asserters to program the PDP. The deployment plan for this is included in D7.2, and the deployment has been implemented as described.

2.1.2 Virtual Wall & Fuseco

The PDP is deployed at both of these testbeds, and uses a standard configuration. No customisation was required for the PDP because both of these deployments have or plan to adopt SFA for resource reservation. Therefore the deployment of the PDP in these two testbeds was straightforward. The reservation of resources is via the slice owner requesting resources from the testbed and presenting their slice credential. If the testbed owner agrees, they program the PDP by instructing it that the resources they have allowed are allocated to the slice.

2.2 Federation Authorisation

Federation authorisation is illustrated below using the example of two testbeds, BonFIRE and the Virtual Wall.

Figure 1 shows the request phase of resources at BonFIRE and the Virtual Wall using the same slice credential. Even though at the time of deployment, BonFIRE has not implemented SFA as the method of requesting and allocating resources, it is still possible to achieve interoperation via the use of the slice credential by mapping the slice credential to BonFIRE's concept of an "experiment".The Virtual Wall does implement SFA, and so no mapping was required in this case.

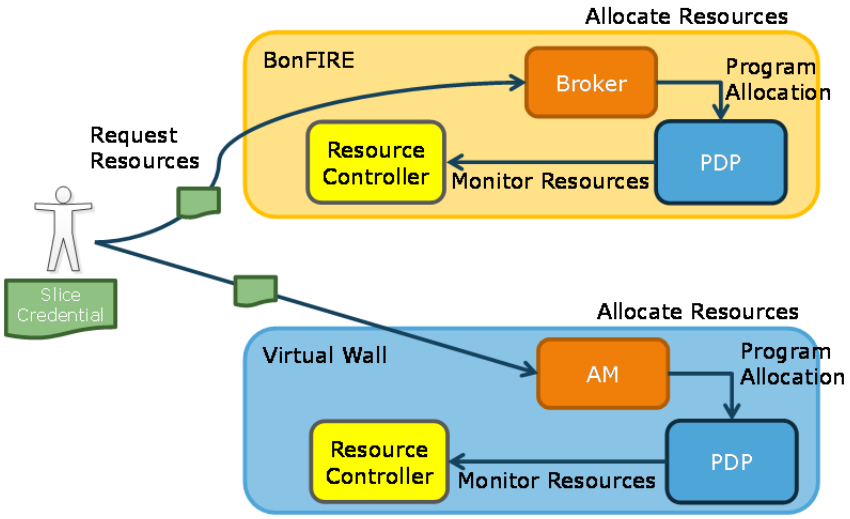


Figure 1: Federation Authorisation - Request Phase

When the request for resources is received at the testbeds, the respective decision making component (the Broker in the case of BonFIRE and the Aggregate Manager in the case of the Virtual Wall) makes a decision whether the request is permitted and if so, programs the allocation of resources into the PDP. This programming maps the credential presented by the user with the allocation of resource decided upon by the Broker or the AM.

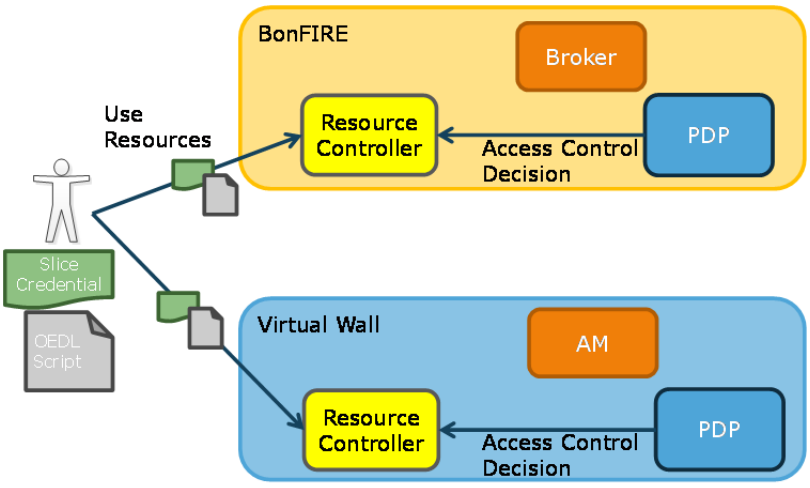


Figure 2: Federation Authorisation - Usage Phase

Once the resources are allocated, they can be used in a single experiment, as shown in Figure 2. The actual function of the experiment the user wants to execute is represented by the OEDL (OMF Experiment Description Language) script in the figure. Along with the OEDL script, the user submits their slice credential used in the request phase. Because the PDP has already been "programmed" to indicate the resources are allocated to the slice credential, the PDP can make a positive access control decision.

This pattern not only satisfies the major requirement of federation authorisation, it also satisfies the other major requirement that the testbeds are in control of their own resources: the testbeds' decision making components are the ones that decide whether a resource request is granted or not.

2.3 Lessons Learned from Deployments

The deployments illustrated some issues that needed to be addressed in the next development cycle, and these are summarised below.

- Currently the deployments at each site have been typically performed by IT Innovation, mainly because the installation of the PDP and associated components is complex. Multiple components need to be installed and configured in order for the PDP to be incorporated into its environment. The conclusion is that the PDP is too difficult to install, and installation needs to be made easier.
- The PDP was customised for each deployment. Initially the customisations were incorporated into the body of the PDP code, mainly because the customisations added new functionality. However, this also meant that the code base was becoming difficult to manage and maintain, due to all the deployment-specific customisations included in the main code body. It was identified quickly that there was therefore a need for standardisation - making as much of the PDP as generally applicable as possible, and clearly separating between generic code and any deployment-specific code.
- It is difficult to trace failed access requests in the PDP system logs – typically the log files are very large and it is difficult to search through them.
- When an access request fails, users are not told anything – the system simply does not respond. This is not very helpful as users are left wondering whether their request has got through or not.

The points discussed above gave rise to the need to conduct a review of the PDP code base, to address these issues. This review is discussed in the next section, which provides requirements that need to be addressed in the next development cycle. The review goes beyond these lessons learned, but the lessons learned from the deployments illustrated the necessity for a comprehensive review of the PDP's code structure, ease of installation and generality.

2.4 FRCP-PDP Usability, Scalability and General Applicability Review

A usability and scalability review of the PDP was conducted internally at IT Innovation based on experiences in deployments of the PDP at different testbeds. The purpose of this review was to determine the areas that needed improvement for aspects such as ease of use, ease of maintenance, scalability and applicability to a wider range of deployments than currently imagined.

This section contains the results from this analysis in the form of requirements for developments in cycle 3. Many of these requirements require updates to the design and implementation of the PDP, and plans for technical developments to address these requirements are reported in D7.4. Other requirements reported here pertain to non-technical aspects (such as documentation), and these will be addressed during cycle 3 but not necessarily discussed in D7.4.

2.4.1 Documentation

- *Requirement 1: D*ocumentation should be hyperlinked and automatically generated using mark-up tools such as Sphinx.
- *Requirement 2:* Documentation should include a motivating overview and architecture diagram that conveys why a consumer of the technology should use it and give some structure to how it will be integrated into their systems.
- *Requirement 3:* Documentation should provide guidance to PDP administrators on how to create Authorised Asserter scripts and policy rules.
- *Requirement 4:* The documentation should guide PDP administrators to create static policy rules for authorisation. The documentation should include example policies and / or policy templates that can be adapted and customised by the user to suit their own policy requirements.
- *Requirement 5:* Provide documentation that will allow testbed administrators to let their clients know how to run experiments on OMF Resource Controllers that will be protected by the PDP. Running experiments on protected OMF Resource Controllers will be slightly different from running experiments on non-protected OMF Resource Controllers and will require extra steps from clients such as including their Slice Credential documents to the requests, and therefore the necessary guidance should be provided to the clients. Documentation should include information on how to configure the OMF experiment tool, as well as how to use slice credentials when running experiments.
- *Requirement 6:* PDP installation and user manual documentation should include standard or template authorised asserter scripts and examples of rules for illustration purposes and to help the testbed administrator create their own assertion scripts and rules.
- *Requirement 7:* The commands are complex and the examples given are not what would be expected by a developer as they do not describe the command line options. Authorised Asserter command line API scripts should be described in accordance with standards such as UNIX MAN pages.
- *Requirement 8:* Trust anchors (e.g. X509 certificates) are stored on the file system, which is acceptable although there is no reference to concepts such as “Trust Store” and “Trust Manager”. There are no guidelines provided on how to ensure the security of the directory to avoid the threat caused by malicious actors. The PDP documentation should contain precise guidelines on best practices for protecting PDP trust stores (e.g. encrypting and storing private keys etc.). The current PDP documentation provides only core information on how to configure and manage PDP trust stores using UNIX file permission system.
- *Requirement 9:* The documentation should also clearly explain that each Certification Authority (e.g. FED4FIRE) should provide operating procedures for registration and certification, as well as each Certification Authority should provide their Certification Policy.

2.4.2 Installation

- *Requirement 10:* Currently the PDP installation is a complex and manual process, which is prone to errors and a barrier for adoption. In order to ease the PDP installation and initial PDP configuration procedure appropriate automated tools (such as Vagrant or Docker) should ideally be used to create fully automated PDP installation and initial configuration script(s).

2.4.3 PDP Configuration

- *Requirement 11:* The current PDP installations are hard coded to each deployment. This means that there are multiple versions of the PDP to maintain, which is not scalable. Any deployment-specific code should be separated out from the main PDP code base, and the main code base should be generally applicable.
- *Requirement 12:* Message handling and policy rule configuration should be separated. Deployments are named by testbed (e.g. “vwall” or “bonfire”) in the configuration file, and the deployment name is used to determine the message handing (how to get facts, attributes, etc from message) and policies (how message outcomes relate to decision making). Coupling these two functions together in collections specific to one deployment is firstly architecturally problematic as the two functions are different aspects and secondly not scalable to further deployments.
- *Requirement 13:* The policies that apply to an installation should not be distributed with the PDP – instead, they should be set by the systems administrator on installation. Example policies as templates should be included in the distribution so as to guide the testbed administrator in creating the policies for their testbed.

2.4.4 Distribution & Code Management

- *Requirement 14:* The PDP requires a patched version of OMF, and these changes to OMF should be handled by a version control system and release procedure. IT Innovation should discuss possible options with NICTA of how the patched OMF can be version-controlled and release-managed. Possible options could be incorporation into the main trunk of OMF, a branch, or a separate release.
- *Requirement 15:* The generic PDP distribution should not include specific testbed deployment scripts (i.e. Authorised Asserter CLI scripts etc.) – these should be distributed separately.

2.4.5 Administration of the PDP

- *Requirement 16:* Investigate tools to support visualisation of policies and logs for systems administrators. Commands are provided to get the working memory state by listing asserted facts but this is a complex low level view on the data and for any large deployment would be unreadable by humans. This is problematic when trying to investigate the current state of access to resources and the facts contributing to a security decision. Readability of policies and facts is important for trust in the system itself and knowing that the correct decisions are being made.
- *Requirement 17:* Investigate tools to support analysis of message authorisation decision outcomes for system administrators. Currently all message authorisation requests are logged to one common log file which makes if difficult for system administrators to analyse message authorisation requests (i.e. system administrators will need to analyse a large common PDP log file if there are errors and problems need to be fixed).

2.4.6 PDP Error Handling

- *Requirement 18:* Mechanisms for ensuring the consistency of the fact base (i.e. handle duplicate facts etc.) need to be investigated. In the current version of the PDP, it is possible to assert mutually inconsistent facts (for example, a fact saying a situation is true and another saying the same situation is false), leading to indeterminate decisions.
- *Requirement 19:* When a request is denied (e.g. not authorised) no error message or any feedback is sent to the client. This decreases the usability of the system as the user has no idea what is going on. A mechanism for reporting the outcome of security decisions to the client should be implemented.

2.4.7 PDP Performance

- *Requirement 20:* The primary performance of the PDP can be measured in terms of transactions a second (e.g. security decisions are second). The performance will be influenced by factors such as the size of the fact base, complexity of policy rules, arrival rates and available compute resources. Performance tests should be conducted to understand how the PDP will scale for expected deployment conditions.

2.4.8 Generalisation

Currently the PDP contains code that is specific to certain deployments. These were created as a means of helping early adopters of the PDP to get up and running quickly, but it is recognised that this is not desirable as the number of deployments increases, as it will require extensive code maintenance. Hence, this section contains a series of recommendations based on a review of the PDP code and deployments, with a view to making the PDP more generally applicable to other deployments without code changes.

- *Requirement 21:* Remove any deployment-specific assertion message parsing. The existing PDP contains special bean objects corresponding to specific deployments, and these should be removed where they are not generally applicable to other deployments. Standardised assertions should be investigated to replace these specific objects.
- *Requirement 22:* Identity standardised mappings for standard types of credentials. Parsing different credentials such as X.509 Certificates or Slice Credentials has revealed same patterns therefore providing standardised mappings will help to control what facts should be generated from different credential types and consequently used in rules.
- *Requirement 23:* Provide a REST interface for the Authorised Asserter. This will be used by the Authorised Asserter to assert or retract facts to and from the PDP without the need to use OMF tools.

3 SLA Management

Within the second cycle of the project the work has been focused on developing and improving the different modules for the SLA management:

- **SLA Management module:** the key component for SLAs was introduced and improved during cycle 2. The work includes the SLA creation and evaluation functionality and performance and security related enhancements.
- **SLA Collector:** the centralized communication point has also been developed during cycle 2.
- **SLA plugin for Fed4FIRE portal:** a tool for experimenters to accept SLAs offered by testbeds and visualize SLA evaluation results at the end of the experiment has been developed in cycle 2.

During cycle 2, three testbed providers have shown their interest on offering SLAs for their resources in the federation: iMinds, Fraunhofer Fokus and NTUA. Therefore, the SLA Management module has been deployed and maintained in those facilities to allow SLAs on their respective testbeds.

The integration of SLA Management with the Monitoring Systems has also been done for FUSECO and NETMODE testbeds, retrieving monitoring data (i.e. resource availability) from a temporal database located in FUSECO where both testbeds push their monitoring streams.

SLA Collector has also been deployed in an iMinds' instance as a central component of the federation. The SLA plugin for the portal has been improved and adapted to make use of the SLA Collector upon its deployment. In this way, the SLA Collector acts as a proxy between the Fed4FIRE portal and the SLA Management module of each testbed.

A more detailed description of the developments of each SLA component is provided next.

3.1 SLA Management Module

The SLA Management was introduced in the project in cycle 2. This component is located at testbed level and is the main element in the SLA management process. It is in charge of creating, storing, evaluating and destroying SLAs defined by testbeds. It also allows retrieving information about SLA templates[1], SLA[2] themselves and SLA violations so that both experimenters and testbeds can know at any time which SLAs are provisioned, active or finished together with the evaluation result.

The development work has been done following the guidelines described in "D7.2 - Detailed specifications regarding trustworthiness for cycle 2". During this cycle, some functional enhancements had to be implemented as well (e.g. an enforcement job is automatically created when an agreement is created). Furthermore, both XML and JSON format support have been added to the SLA Management tool as information input and output format. Considering the different locations of testbeds in Fed4FIRE, the time zone can also be specified in the date field of SLAs.

All access to the SLA Management module is done through REST calls. Basic operations, such as SLA creation and SLA evaluation start and stop can be done. Furthermore, more complex calls have also

[1] An SLA template is an offer proposed by a testbed provider.
[2] SLA or SLA agreement is an accepted SLA template by a testbed user.

been implemented, allowing filtering results using different GET call parameters. For example, it is possible to retrieve SLA violations by SLA identifier, testbed or date period. Details of the SLA Management API are described in Appendix A (Section 6) of this document.

Regarding security, another change that has been tested has been enabling the use of certificates as inter component authentication, making possible to force the use of client certificates when calling any of the REST interfaces from the SLA Manager. By doing this, the access to the SLA Management API is restricted to authorized components, i.e. SLA Collector.

During cycle 2, the SLA Management module has been deployed at iMinds, Fraunhofer FOKUS and NTUA testbed providers. Therefore, VirtualWall, Wi-Lab2, Fuseco Playground and Netmode testbeds currently support SLAs. The SLA that has been defined as a first approach corresponds to the "X% availability for Y% of resources" policy described in deliverable D7.2. However, the possibility of offering other kinds of SLA policies will be studied for cycle 3. No modifications of the SLA Management module would be required to extend the possible SLA offerings of next cycle since it is designed to support other types of policies.

3.1.1 Integration with Monitoring Systems

In order to evaluate SLAs, the SLA Management tool retrieves monitoring metrics from the PostgreSQL monitoring database located at each testbed. However, this is expected to change for next cycle, where metrics would be located at a central database component of the federation.

The first SLA evaluation approach developed in cycle 2 was to cover the set of resources under a single sliver per testbed. The identifiers of the reserved resources were reflected in the SLA accepted by the experimenter and later the SLA Management module would ask the testbed monitoring database for the corresponding metrics. However, due to the different implementations per testbed regarding the mapping of resources to slivers, it has been decided to cover the set of slivers from resulting from a reservation in one SLA per testbed. More details on this change can be found in the deliverable "D7.4 - Detailed specifications regarding trustworthiness for the third cycle".

Resource availability is the only metric currently retrieved from the monitoring database, indicating with a 1 or a 0 the status of the resource. Using sliver identifiers to obtain those metrics instead of individual resource identifiers will ease the process of creating and evaluating SLAs. The requirements for this change involve testbeds to make the sliver-resource mapping information available, a topic which will be tackled during cycle 3.

.

3.2 SLA Collector

This module has been developed for cycle 2 and is being used for different purposes within Fed4FIRE. It is a common element for the federation whose purpose is to act as a central communication point for the client tools and the SLA Management module located in each testbed. The initial functionality of the SLA Collector was just to act as a proxy to redirect the calls done from the common components to the specific SLA Management module installed in a testbed.

The proxy functionality has been implemented such a way can it can detect any call being done to the SLA Management module and redirect the call. It has been implemented in a generic way, this is, changes in the parameters or possible calls in the SLA Management module don't imply changes in the SLA Collector. Moreover, besides the proxy functionality, the SLA Collector also allows to ease the process of creating a new SLA: an SLA can be created taking as input arguments the active slice, the experimenter id, the expiration date of the reservation and the list of slivers that are selected. By

providing those parameters, instead of the complete WSAG definition, thus reduces the complexity for client tools to create an SLA. Furthermore, the SLA Collector can as well be requested to retrieve the SLAs that exist under an experiment slice.

Using the SLA template defined by the testbed, the SLA Collector takes the previous parameters and builds the final SLA that will be redirected to and stored in the proper SLA Management tool database.

The SLA Collector has its own database to be able to associate the name of a testbed with its IP. The management of this database can be also done through the SLA Collector.

The access to the SLA Collector API is described in Appendix B (Section 7) and access to it has been controlled using HTTP Basic Authentication. The SLA Collector has been tested to use client certificates to access the SLA Management module API. The client certificate must have been issued with the same CA as the server certificate from the SLA Management module. For Cycle 2, tests have been done with fake certificates (the ones that provide Spring Framework for testing purposes).

3.3 SLA plugin for Fed4FIRE portal

During cycle 2, access to SLAs has only been allowed through the SLA plugin developed for the Fed4FIRE portal. By means of this plugin, experimenters can visualize SLAs offered by testbeds and accept them, being able to see their evaluation at the end of the experimenter.

The workflow for creating a new SLA has suffered some modifications throughout this cycle due to the change of SLA coverage from resource to sliver identifiers. This, together with the late deployment of the SLA Management module in testbeds that wanted to offer SLAs, has caused the access SLAs to be unavailable to experimenters during cycle 2. Thus the functionality of the SLA plugin has been hidden in the portal, waiting for the SLA core components to be ready.

The first version of the SLA plugin has been developed to use the resource identifiers listed in the “Resource” under the slice view of the portal. Once the desired resources are selected, a window appears on the screen indicating that a SLA is being offered by the testbed owner with those resources, giving the chance to the experimenter to accept or decline it. When the SLA is accepted, the process for the resource reservation is done by the underlying MySlice manifold. Otherwise, the resources of that specific testbed could not be reserved. After that, under the portal SLA tab, the experimenter can visualize the current status of the accepted SLAs. A table indicating the testbed, the SLA identifier, the creation and expiration dates and the status of the SLA evaluation is presented.

At the end of cycle 2, the decision of changing the use of sliver identifiers instead of resource identifiers in SLAs produced some modifications in the SLA creation procedure. Nevertheless, the overall process of accepting and visualizing SLAs through the SLA plugin, as well as the features presented in the SLA visualization tab, will remain the same. In order to get the sliver identifiers of the reserved resources, the actual reservation has to occur prior to the SLA creation. The plugin will then obtain the sliver Ids from the response of the testbed’s AM and then will create the SLA agreement.

The changes in the architecture related to how the components would communicate with SLAs for cycle 3 imply that more modifications have to be done to the SLA plugin. Thus, the development of the new version, able to satisfy defined requirements regarding sliver identifiers and the future requirements for cycle 3, was started by the end of cycle 2. Further details on those requirements can be found in “D7.4 - Detailed specifications regarding trustworthiness for third cycle”.

4 Trust and Reputation

The *Reputation Service* in Fed4FIRE implements the mechanisms for building trustworthy services based on the combination of Quality of Experience (QoE) and monitoring data. The developed mechanisms reflect the end users' (experimenters') perspective with the objective of empowering the users/experimenters to select testbeds based on dynamic performance metrics. These metrics eventually offer a "smart" user support service that provides a unified and quantitative view of the trustworthiness of a facility. The *Reputation Service* is complemented with the *Reputation Service Frontend*, allowing F4F users to submit QoE feedback with regards to the conducted experiment. The aforementioned components are described hereafter along with their interactions with other F4F components.

4.1 Reputation Service

During cycle 2, a reputation-based trust framework for federated testbeds, termed as Federated Trust and User Experience (FTUE) Framework has been designed and implemented as the basis for the Reputation Service in Fef4FIRE.

FTUE Algorithm[3]

As documented in D7.1 and D7.2, the FTUE algorithm bases its principles on the ROCQ scheme, proposed by Garg and Battiti [1]. ROCQ was originally designed for peer-to-peer systems, and therefore several modifications were implemented in order to adapt it for a federated testbed environment. The FTUE framework [2] combines the perceived user experience, as provided by user feedback, with the monitoring data retrieved for the respective experiments.

The system model consists of a set of *K* testbeds advertising *S* aggregated services and *N* users, while each user *i* is conducting M_i experiments. Each testbed may support one or more of these aggregated services, i.e. a service pertaining one or more testbeds. Services are distinguished to technical and non-technical services. A technical service is a technical aspect stemming from the nature of each testbed and its resources. Some examples of services in testbeds are: node availability (up/down), link quality (e.g., Packet Delivery Ratio, Service Loss Ratio, Packet Loss - over a predefined threshold), system resources performance[4] (e.g., CPU, memory, disk - average utilization over a predefined threshold). A non-technical service corresponds to the non-quantifiable experience of the user on conducting an experiment. An example of such a service is the *Overall Experience* of the user conducting the experiment. Detailed description of technical and non-technical services is provided in Deliverable 7.2 [3].

Each user, also called experimenter, upon completing an experiment $j \in M$, provides feedback (an Opinion) denoted as O_{ij}^{S} regarding the quality of experience in using the aggregated service $s \in S$. The ratings of the experimenters are given in terms of a Mean Opinion Score (MOS), as MOS is a typical user-related metric for measuring QoE, ranging from 1 (worst) to 5 (best). Users are asked to

[3] The FTUE framework has been presented in the 2014 IEEE 19th International Workshop on Computer Aided Modelling and Design of Communication Links and Networks (CAMAD) [2]

[4] Applies only for shared computing resources.

evaluate each aggregated service and then the framework delegates each service opinion to the respective testbeds (O_{ij}^{sk}).

After collecting the provided feedback, each allocated opinion is compared with the corresponding monitoring data (i.e. D_j^{sk} represents the monitoring data for a service *s* of testbed *k* for the *jth* experiment), as retrieved from the monitoring infrastructure. Depending on the outcome and whether there is a match between the opinion and the monitoring data, the credibility of the user is accordingly altered. Opinions from users are weighted with the Credibility C_i and Quality Q_{ij} values and then aggregated to form the trust scores T_{ks} for each service s of testbed *k* (Figure 3).

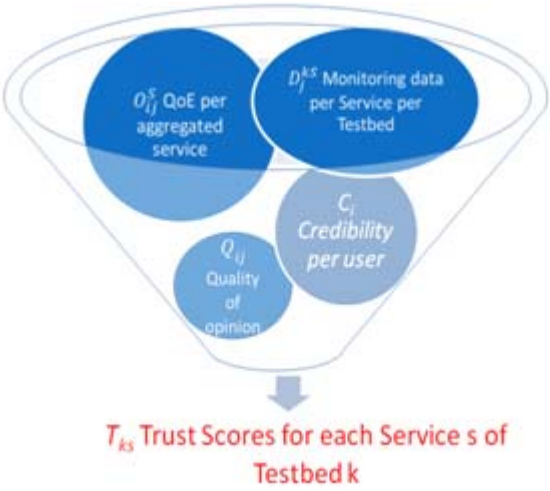


Figure 3: FTUE (high level description)

The description of the respective variables used in our reputation system follows. For the sake of simplicity we provide the description for a service denoted as *S*, while *t* denotes the iteration number.

Opinion: expresses the amount of satisfaction for a service *S* of the user *i* for his j^{th} experiment and is represented as follows:

$$O_{ij}^{kS}, \{1 \leq i \leq N, 1 \leq i \leq M_i, k \epsilon K\}$$

where *N* is the number of users in the system and M_i is the number of the experiments conducted by user *i*. Opinion O_{ij}^{kS} takes a value in [0, 1].

Quality: represents the confidence of the user providing feedback. If a user is not sure about his opinion, his credibility and the final reputation should be influenced proportionally. Quality reported from the user *i* for his j^{th} experiment is represented as Q_{ij} and lies within [0, 1].

Credibility is used to express whether a user provides true or false judgements for testbed services. Each user has a credibility value assigned, which is adjusted according to the honesty of his ratings. If credibility is low - namely the user is not trustful - his opinion plays a minor role in the system evaluation and vice versa. The credibility value of each user, which lies in the [0,1] area, is updated after each evaluation that provides according to different scenarios, as will be further explained.

Since, several testbeds can be involved in a single experiment, different situations may arise. Therefore, each potential scenario has been identified and a corresponding formula for updating the

Credibility of an experimenter has been determined. For every case the Opinion O_{ij}^{kS} provided by the user *i* is compared with the retrieved monitoring data D_j^{kS} with regards to a threshold *e* that is properly configured depending on the rated service.

Scenario I

Consider an experiment conducted by user *I* spanning a set of testbeds in the federation and service *S*, which is affiliated with one or more of these testbeds. If the absolute difference of the given opinion O_{ij}^{kS} from the retrieved (normalized) monitoring data value D_j^{kS} is not greater than the threshold *e* for every testbed *k* involved in the experiment, then the credibility of the user is increased as follows:

$$C_i^{t+1} = C_i^t + \frac{\left(1 - C_i^t\right) Q_{ij}}{2}, \quad if \left| D_j^{kS} - O_{ij}^{kS} \right| \le e \, \forall \, k \exists K$$

This scenario corresponds to the case where the ratings of the experimenters match the retrieved monitoring data for every testbed.

Scenario II

In this scenario, the absolute difference of the given opinion O_{ij}^{kS} from the monitoring data D_j^{kS} is greater than the threshold *e* for every testbed *k* involved in the experiment. In addition, the retrieved monitoring data for all involved testbeds is either above the opinion ($D_j^{kS} - O_{ij}^{kS} > 0$) or bellow the opinion $D_j^{kS} - O_{ij}^{kS} < 0$. Then, the credibility of the user is decreased as follows:

$$C_i^{t+1} = C_i^t - \frac{C_i^t Q_{ij}}{2}, \quad if \left| D_j^{kS} - O_{ij}^{kS} \right| > e \ \forall \, k \exists K,$$
$$if \left| D_j^{kS} - O_{ij}^{kS} \right| > 0 \vee \left| D_j^{kS} - O_{ij}^{kS} \right| < 0 \forall \, k \exists K$$

This scenario corresponds to the case where the ratings of the experimenters differ from the monitoring data retrieved for every testbed.

Scenario III

In this scenario, the rating provided by the experimenter lies in the middle of the rating range (3 out of 5) and the absolute difference of the given opinion O_{ij}^{kS} from the monitoring data D_j^{kS} is greater than the threshold *e* for every testbed *k* involved in the experiment ($D_j^{kS} - O_{ij}^{kS} > e$) . In addition, the retrieved monitoring data of at least one testbed is above the opinion $D_j^{nS} - O_{ij}^{nS} > 0, n \in K$ and the monitoring data of at least one testbed is below the opinion $D_j^{mS} - O_{ij}^{mS} > 0, m \in K$. Then, the credibility of the user is computed as follows:

$$C_i^{t+1} = C_i^t + \frac{(1 - C_i^t) Q_{ij}}{2}, \quad if \left| D_j^{kS} - O_{ij}^{kS} \right| > e \ \forall \, k \in K \wedge O_{ij}^{kS} = 3 \wedge$$
$$\exists n \in K : \left| D_j^{nS} - O_{ij}^{nS} \right| > 0 \wedge \exists m \in K : \left| D_j^{mS} - O_{ij}^{mS} \right| < 0$$

This scenario corresponds to the case where experimenters evaluate in a rather conservative way giving median ratings, while the services are either of high or low quality.

Scenario IV

In this scenario, the absolute difference of the given opinion O_{ij}^{nS} from the monitoring data D_j^{nS} is greater than the threshold *e* for at least one testbed $n \in K$ of the experiment. At the same time, there exists at least one testbed $m \in K$ where the absolute difference of the given opinion O_{ij}^{mS} from the monitoring data D_j^{mS} is not greater than the threshold *e*. Then, the credibility of the user is increased, while for the testbeds $L \subset K$ that their monitoring data deviate from user opinion, we acquire a modified opinion as follows:

$$\begin{cases} C_i^{t+1} = C_i^t + \dfrac{\left(1 - C_i^t\right) Q_{ij}}{2} \\ O_{ij}^{'lS} = O_{ij}^{lS} + \left(D_j^{lS} - O_{ij}^{lS}\right) Mu \end{cases}$$
$$\exists n \in K : \left| D_j^{nS} - O_{ij}^{nS} \right| > e \wedge \exists m \in K : \left| D_j^{mS} - O_{ij}^{mS} \right| < e$$

The modified opinion lies between the original opinion and the monitoring data depending on the multiplier variable *Mu*, which indicates the isolation level of malicious users and takes a value in (0,1]. This modified opinion is used to compute the reputation of the given service.

This scenario corresponds to the case where at least one testbed has different behaviour from the others involved in the experiment.

Reputation

R^{kS} for the service *S* of testbed *k* is the result of aggregating user Opinions, weighted by Credibility and Quality values.

$$R^{kS} = \frac{\sum_i^N \sum_j^{M_i} O_{ij}^{kS} \cdot C_{ij} \cdot Q_{ij}}{\sum_i^N \sum_j^{M_i} C_{ij} \cdot Q_{ij}}, \forall k \in K$$

The aggregation process is carried out by the means of a double summation, where the outer sum evaluates all users and the inner sum evaluates all the experiments M_i of user *i* that utilized resources from testbed *k*.

FTUE Performance Evaluation

In order to assess the effectiveness of the framework, we carried out a series of experiments in a simulated environment. We developed a Ruby-based simulation platform that provides an API for creating new experiments and managing the trust mechanism, in order to conduct extensive behaviour analysis. It consists of three distinguished processes: i) scenario generation, ii) input modelling and iii) output analysis. The simulation tool is designed so that new trust/reputation management algorithms can be added via a simple plugin.

User ratings (Opinions) as well as the retrieved monitoring data were simulated by different uniformly distributed functions. With the purpose of testing the proposed algorithm under diverse

conditions, we modelled the behaviour of four user types that correspond to the scenarios presented above. Users that behave according to Scenario I are considered *truthful* users, while those of Scenario II are considered *malicious*. Scenario III generates moderate users, though are considered *truthful* whereas users generated from Scenario IV are either *truthful* or *malicious in disguise.* The Credibility value of each user is initiated to 0.5.

Apart from evaluating the technical services, a rating for a non-technical service, the *Overall Experience* of using the testbed, has been included in the simulations. Due to the subjective nature of the attribute, two amendments are necessary. Firstly, all credibility equations need proper modification because of the absence of a ground truth when updating a user's credibility. Therefore monitoring data has been substituted with the reputation calculated from the current opinions for the Overall Experience. Moreover, user Credibility in this case is a different variable, corresponding to non-technical services.

The performance of the framework in dynamically changing conditions is presented in the following subsection. Its robustness is also tested against malicious attacks with a simple algorithm that calculates reputation as a mean score.

Simulation I

In the first experiment, we assess the adaptability of our framework in changing conditions. 100 users conduct 10 experiments each, utilizing resources from 2 to 4 federated testbeds, according to a uniform distribution. Each testbed is considered to have (i) one technical service, with corresponding monitoring data, and (ii) one non-technical service *Overall Experience* calculated as described above. For the first 500 experiments the service in all testbeds (namely Testbeds A-D) operates smoothly (monitoring data range in [0.8, 1.0]) and users are 80% generated according to Scenario I and 20% according to Scenario II. Thereafter, for the next 500 experiments, the service of Testbed B is experiencing technical issues (monitoring data range in [0, 0.2]). Users are now rating the experiments either based on Scenario III (20%) or Scenario IV (80%) while the value of *Mu* is set to 0.75.

Figure 4 illustrates the reputation values for the Service and the *Overall Experience* attribute over experiments for Testbed A and Testbed B. The results of Testbeds C and D are similar to those of Testbed A and are omitted for clarity reasons. As it can be seen, the framework converges quickly after a few experiments and for the first 500 experiments, the reputation score for both testbeds is high, successfully constraining any malicious users. Moreover, not long after the degradation of the service in Testbed B, our framework reacts by lowering the respective reputation score. There is, also, a minor drop in the reputation of Testbed A which can be explained by the value of the multiplier variable Mu as it will be demonstrated in the next experiment. Finally, as far as the *Overall Experience* service is concerned, each user bases his opinion on different factors with different weights and this subjectivity justifies the fluctuations depicted in the graph.

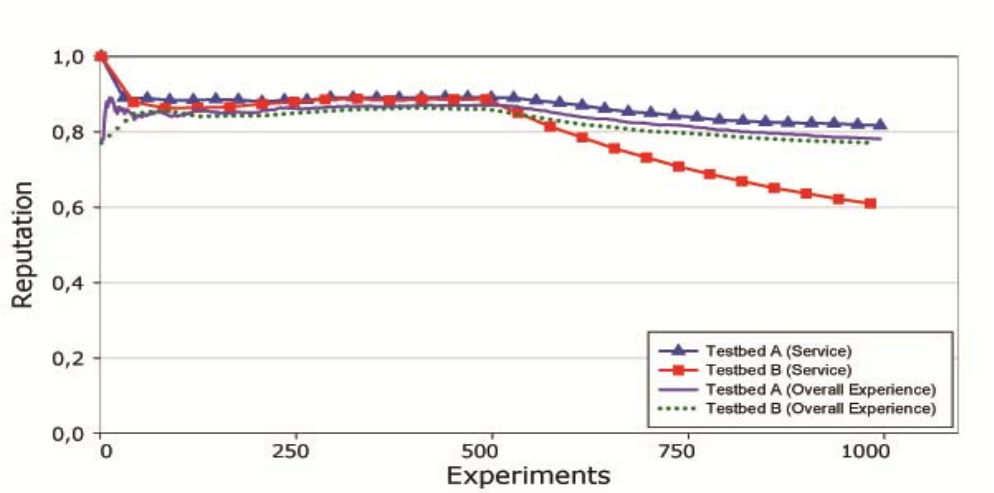


Figure 4 Simulation I: FTUE algorithm under changing conditions

Simulation II

In order to study the impact of the isolation factor *Mu* of Scenario IV in our proposed framework, we conducted a series of simulation experiments for different *Mu* attribute values. Mu multiplier is used for the calculation of the modified opinion. Values close to 1 adjust the Opinion towards the value of the monitoring data, thus isolating the majority of malicious users. The problem that arises with the use of multiplier values close to 1 is that not all testbed services are objectively monitored. For example, consider an experiment with a time span of 1 hour, during which the resources of a particular testbed become unavailable for 5 minutes. Monitoring data will yield a value of 92%, but experimenter-wise this time might be crucial and thus the reason for a bad rating. In those cases, monitoring data are indicative of the service status, but they do not necessarily reflect the view of the experimenter. As a result, the isolation factor must be properly leveraged depending on the service.

Continuing on, for the current experiment we have used the same set of users as those of the previous experiment for comparison reasons. Evidently, as the multiplier's value increases (Figure 5), the reputation values asymptotically converge to the corresponding monitoring data. In addition, even for smaller values of *Mu*, the trust mechanism successfully adapts to the new conditions. Nevertheless, the observed small drop in the reputation of the testbed that still operates smoothly is attributed to the presence of moderate users in the system (Scenario III).

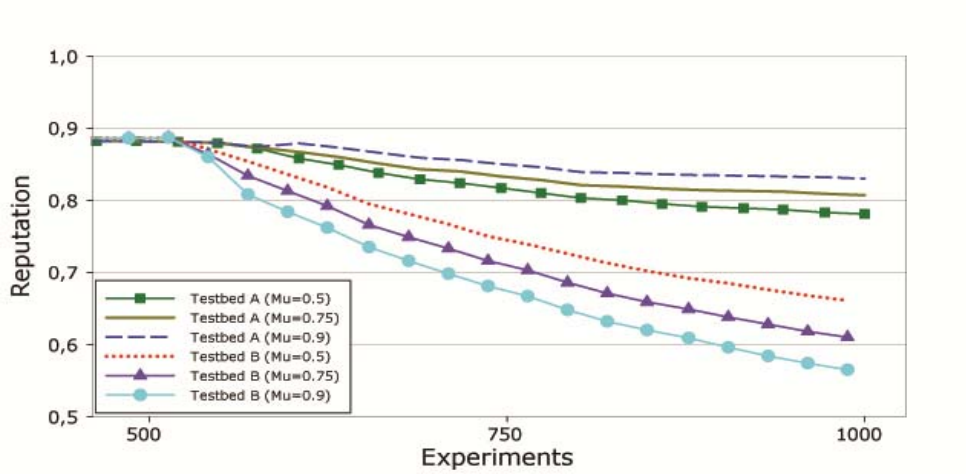


Figure 5: Simulation II: *Mu* impact on FTUE

Simulation III

Finally, we evaluate the resilience of our framework against an intense attack of malicious users, and compare it with an algorithm that calculates reputation as a simple mean score. According to the simulation scenario, 100 users conduct 1000 experiments in total, utilizing 1 to 4 testbeds that have 1 technical service. The reputation scores for a randomly picked testbed service of the federation are depicted in Figure 6, where an increasing percentage of malicious attacks is injected in the simulated environment. It is evident that our trust mechanism can intrinsically mitigate malicious attacks, as opinions of malicious users are isolated and play a minor role, since their Credibility value is low, compared to those of the mean score, where all opinions are considered and valued as equal.

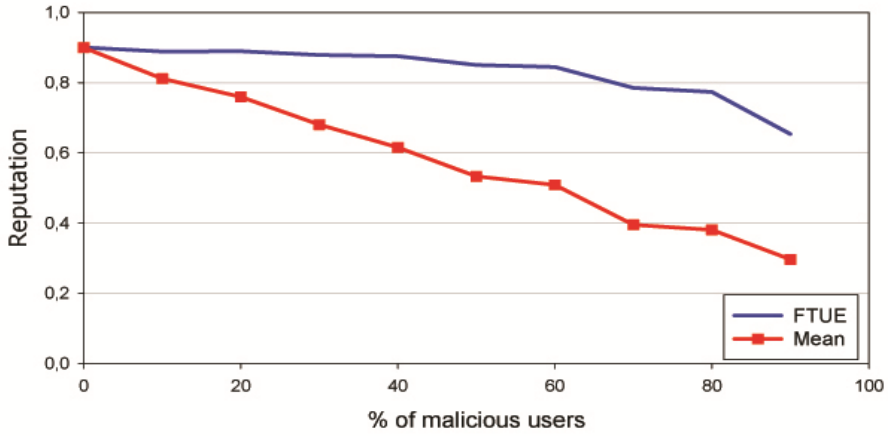


Figure 6: Simulation III: FTUE comparison to MOS

FTUE Implementation

The FTUE framework consists of the *Reputation Service Repository* and a ruby based implementation of the FTUE algorithm.

One of the primary concerns for adopting the FTUE algorithm in Fed4FIRE was the definition of the

"Experiment" since for the various testbeds the resource sharing mechanisms are different (e.g., exclusive use of resource over a constrained timeslot, shared use of resources for a longer period in time). From a researcher's perspective, a slice is a substrate-wide network of computing, communication, and measurement resources capable of running an experiment. An experiment is a researcher-defined use of a slice (an experiment runs in a slice) . With regards to the reputation service the assumption is made that an "Experiment " aggregates all resources that are reserved over a common timeslot/period. In other words the user is asked to provide feedback for the resources that were part of the slice for a particular period in time.

The *Reputation Service Repository* holds information regarding:

1. Testbeds and the technical services that these are under evaluation, i.e. all the services that will obtain trust scores from the proposed reputation mechanism, along with their reputation scores,

2. Users, their quality and their credibility values - regarding technical and non-technical services.

3. Information about the conducted experiments, including their description, experiment ID and associated scores.

The *Reputation Service Repository* data model is provided in Appendix C (Section 8). As presented in the earlier section, for the calculation of the services' reputation the monitoring data, corresponding to the ground truth, as well as the users' opinions, subjective view, are needed. The FTUE framework currently retrieves monitoring data on respective experiments directly from the *OML Server* instead of the Data Broker (Manifold). The Reputation Service frontend facilitates the submission of users' feedback regarding the perceived QoE per experiment. For this purpose the service exposes a REST API, used by the Reputation Service Frontend, which is documented in the Appendix D (Section 9). The REST API is implemented using Sinatra[5].

4.2 Reputation Service Front-End

The Reputation service is accessible to experimenters using the Django-based Fed4FIRE Portal. The frontend has been implemented as a plugin (reputation plugin) deployed on the Fed4FIRE Portal, in order to facilitate the submission of users' feedback regarding the perceived QoE. The reputation plugin was implemented following the architecture proposed by the Django Framework.

The main functionality of the Reputation plugin includes (i) listing user experiments that have not been rated yet and (ii) processing and passing on to the Reputation Service the evaluation feedback provided by the experimenter, as depicted in Figure 7 and (iii) providing the reputation scores for the various testbed services, as presented in Figure 8. Listing of unrated experiments is enabled by:

- Collecting the appropriate slices (and the related resources) based on the current logged-in user.
- Grouping the resources of a slice into experiments.

[5] http://www.sinatrarb.com/

- Specifying the testbeds involved in these experiments.
- Constructing the Experiment ID for each experiment, based on the experiment description (e.g., slice_hrn, user_hrn, timing information, testbed[resources]).
- Retrieving from the Reputation Service the list of experiments that have not been evaluated yet.

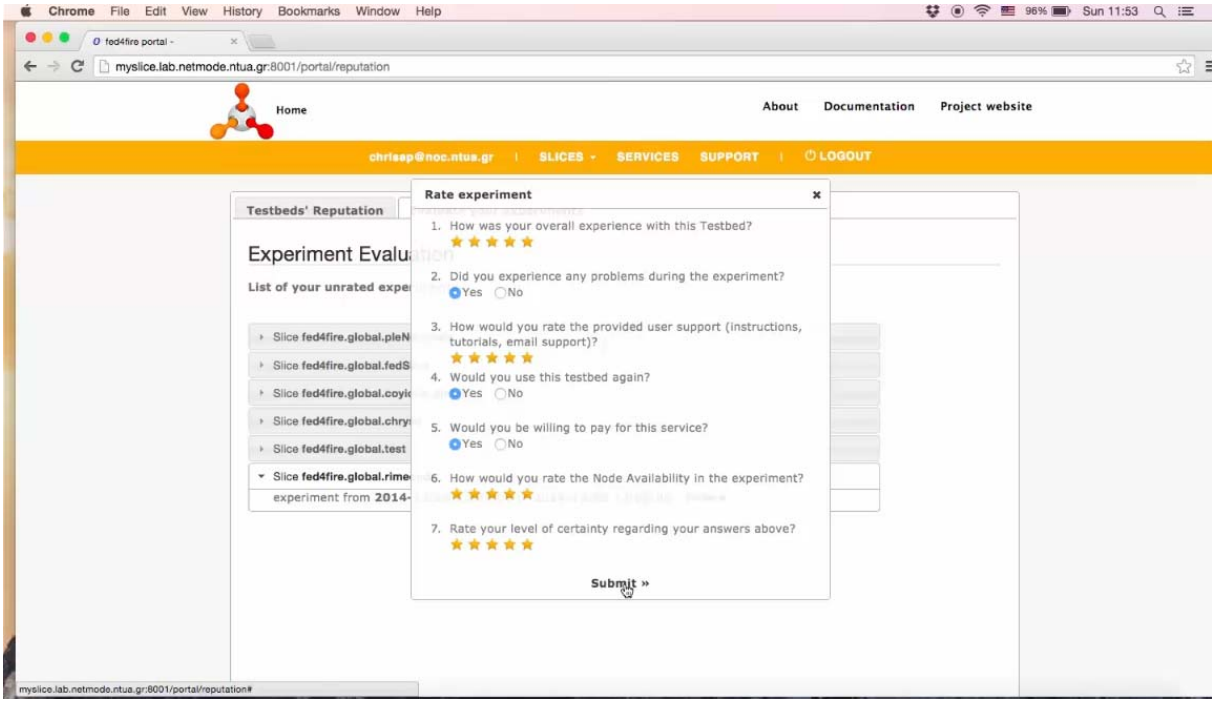


Figure 7: Reputation Service - Provide User Feedback

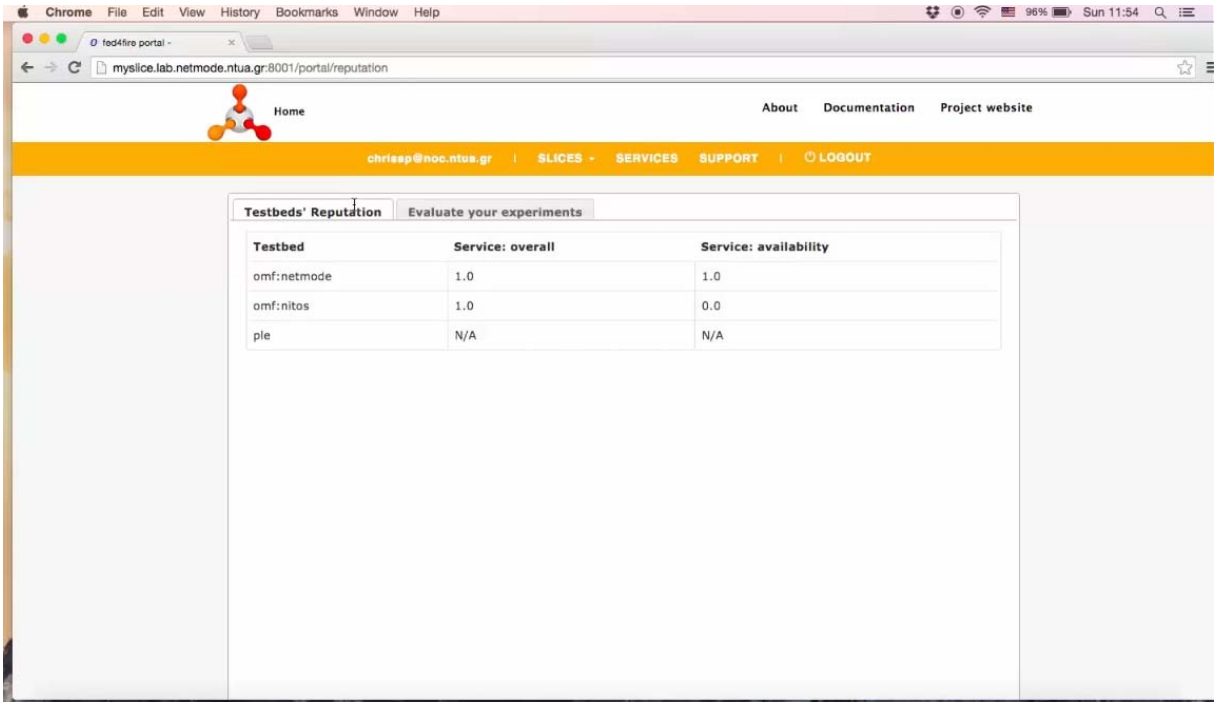
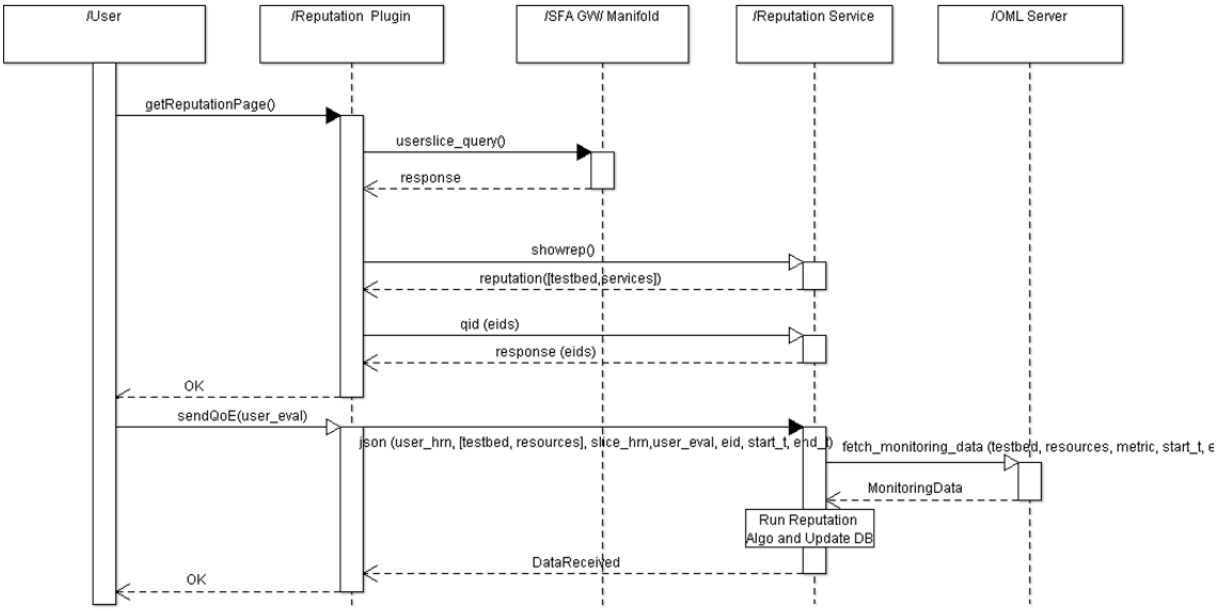


Figure 8: Reputation Service - View Reputation Scores

4.3 Reputation Service Functionality

The implemented process of submitting user feedback and retrieving evaluation scores is presented in the following sequence diagram.



The experimenter submits feedback on the perceived QoE for conducting an experiment as follows:

1. Logged in Fed4FIRE portal, the user clicks on the reputation tab where he can:
    a. Retrieve a list of the experiments he has/is conducted/ing via the SFA GW in Manifold (***userslice_query***). The experiment is essentially a slice bounded within a specific timeframe. The unrated experiments are identified, by retrieving the corresponding info from the Reputation Service (***qid***).
    b. View existing reputation scores for specific testbed services (e.g. node availability) on a per testbed basis (***showrep***)
2. For the experiments that have not been already rated by the user, a scoreboard is provided. The experimenter provides the corresponding score based on questions that are presented via the *Reputation Service Frontend (Reputation plugin)*. The score is sent to the *Reputation Service* (**json**), to evaluate the updated reputation values using the FTUE framework.
3. The *Reputation Service* retrieves from the *OML Server* the corresponding monitoring data, regarding slivers used in the particular experiment (***fetch_monitoring_data***).
4. The *Reputation service* applies the FTUE algorithm and updates the Reputation values for the services of each testbed involved. Experimenter may refresh the Reputation score page in Fed4FIRE portal to retrieve the updated Values.

References

[1] A. Garg, and R. Battiti, ”The Reputation, Opinion, Credibility and Quality (ROCQ) scheme”, University of Trento, IT, TR DIT-04-104, 2004.

5 Conclusions

This deliverable has reported on experiences and developments in cycle 2 in three main areas, corresponding to the development tasks in WP7.

In T7.2, we have deployed the access control Policy Decision Point (PDP) in a number of testbeds and demonstrated that it addresses the major requirements that guided its design. We have also conducted an analysis of usability, generality and scalability and derived requirements for cycle 3. Addressing some of the requirements will require further design and implementation, and the approaches for addressing these requirements are discussed in D7.4, which is issued concurrently with this deliverable.

In T7.3, we have developed the SLA management framework further in three areas. The SLA Management module was improved during cycle 2 to include the SLA creation and evaluation functionality, as well as performance and security related enhancements. The SLA Collector has also been updated: previously in cycle 1, it had a basic proxy function, and this has been enhanced to allow all SLAs for an experiment to be retrieved securely. The SLA plugin for the Fed4FIRE portal has been updated to allow experimenters to accept SLAs offered by testbeds and visualise SLA evaluation results at the end of the experiment.

In T7.4, we have further developed and implemented the Reputation Service, and designed and implemented a reputation-based trust framework for federated testbeds, named the “Federated Trust and User Experience” (FTUE) Framework. The FTUE algorithm bases its principles on the ROCQ scheme, but since ROCQ was originally designed for peer-to-peer systems, several modifications were implemented in order to adapt it for a federated testbed environment, resulting in the FTUE. The FTUE framework combines the perceived user experience, as provided by user feedback, with the monitoring data retrieved for the respective experiments.

6 Appendix A: SLA Management module REST API

6.1 API Introduction

The REST interface to the sla-core system has the following conventions:

- Every entity is created with a POST to the collection url. The body request contains the serialized entity in the format specified in the content-type header. The location header of the response refers to the url of the new allocated resource. The return code is a 201 on success. Templates and agreements have special considerations (see the corresponding section).
- A query for an individual item is a GET to the url of the resource (collection url + external id). The format of the response is specified in the http header with the accept parameter. The return code is 200. As expected, a not found resource returns a 404.
- Any other query is usually a GET to the collection's url, using the GET parameters as the query parameters. The result is a list of entities that match the parameters, despite the actual number of entities. The return code is 200, even if the list is empty.
- Any unexpected error processing the request returns a 5xx.
- An entity (or list) is serialized in the response body by default with the format specified in the Content-type header (if specified). The request may have an Accept header, that will be used if the resource allows more than one Content-type.
- Updating an entity involves a PUT request, with the corresponding resource serialized in the body in the format specified in the content-type header. The return code is 200.
- If a query has begin and/or end parameters, the following search is done: begin <= entity date < end

6.2 Generic operations

The generic operations of resources are shown below. Each particular resource (in following sections) shows the supported operations and any deviation from the behavior of generic operations.

The format of a resource can be modified by a project by using serializers.

GET /{resources}/{uuid}

Retrieve an entity by its uuid.

Request

GET /resources/{uuid} HTTP/1.1

Response in XML

HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<resource>...</resource>

Response in JSON

HTTP/1.1 200 OK
Content-Type: application/json

{ ... }

Usage (for JSON and XML)

curl -H "Accept: application/xml" http://localhost:8080/sla-service/resources/fc923960-03fe-41
curl -H "Accept: application/json" http://localhost:8080/sla-service/resources/fc923960-03fe-41

GET /resources{?param1=value1&param2=value2...}

Search the resources that fulfill the parameters. All resources are returned if there are no parameters.

Request

GET /resources?param1=value1 HTTP/1.1

Response in XML

HTTP/1.1 200 OK
Content-type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<resources>
<resource>...</resource>
<resource>...</resource>
<resource>...</resource>
<resources/>

Response in JSON

HTTP/1.1 200 OK
Content-type: application/json

[{...},{...},{...}]

Usage (for JSON and XML)

curl [-X GET] -H "Accept: application/xml" localhost:8080/sla-service/resources
curl [-X GET] -H "Accept: application/xml" localhost:8080/sla-service/resources?name=res-name
curl [-X GET] -H "Accept: application/json" localhost:8080/sla-service/resources
curl [-X GET] -H "Accept: application/json" localhost:8080/sla-service/resources?name=res-name

POST /resources

Create a new resource. The created resource will be accessed by its uuid. A message will be the usual response.

Request in XML

POST /resources HTTP/1.1
Content-type: application/xml

<resource>...</resource>

Request in JSON

POST /resources HTTP/1.1
Content-type: application/json

{...}

Usage (for JSON and XML)

curl -H "Accept: application/xml" -H "Content-type: application/xml" -d@<filename> -X POST localhost:8080/sla-service/resources
curl -H "Accept: application/json" -H "Content-type: application/json" -d@<filename> -X POST localhost:8080/sla-service/resources

UPDATE /resources/{uuid}

Updates an existing resource. The content in the body will overwrite the content of the resource. The uuid in the body must match the one from the url o not being informed.

Request in XML

PUT /resources/{uuid} HTTP/1.1
Content-type: application/xml

<resource>...</resource>

Request in JSON

PUT /resources/{uuid} HTTP/1.1
Content-type: application/xml

{...}

Response in XML

HTTP/1.1 200 Ok
Content-type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<resource>
...
</resource>

Response in JSON

HTTP/1.1 200 Ok
Content-type: application/json

{...}

Usage

curl -H "Accept: application/xml" -H "Content-type: application/xml" -d@<filename> -X PUT localhost:8080/sla-service/resources/{uuid}
curl -H "Accept: application/json" -H "Content-type: application/json" -d@<filename> -X PUT localhost:8080/sla-service/resources/{uuid}

DELETE /resources/{uuid}

Deletes an existing resource.

Request

DELETE /providers/{uuid} HTTP/1.1

Response in XML and JSON

HTTP/1.1 200 Ok
Content-type: application/[xml | json]

... (free text indicating that the resource has been removed)

Usage (for JSON and XML)

curl -H "Accept: application/xml" -X DELETE localhost:8080/sla-service/resources/fc923960-03fe-41
curl -H "Accept: application/json" -X DELETE localhost:8080/sla-service/resources/fc923960-03fe-41

Messages

Some of the above mentioned methods might return a message. Messages can be returned as XML or JSON.

Message Response in XML

Content-type: application/xml

```
<?xml version="1.0" encoding="UTF-8"?>
<message code="xxx" elemendId="..." message="..."/>
```

Message Request in JSON

Content-type: application/json

```
{"code":"xxx", "elemendId":..., "message": ...}
```

6.3 Providers

- Provider collection URI: /providers
- Provider URI: /providers/{uuid}

A provider is serialized in XML as:

```
<provider>
  <uuid>fc923960-03fe-41eb-8a21-a56709f9370f</uuid>
  <name>provider-prueba</name>
</provider>
```

A provider is serialized in JSON as:

```
{"uuid":"fc923960-03fe-41eb-8a21-a56709f9370f",
 "name":"provider-prueba"}
```

GET /providers/{uuid}

Retrieves a specific provider identified by uuid

Error message:

404 is returned when the uuid doesn't exist in the database.

GET /providers
Retrieves the list of all providers

POST /providers
Creates a provider. The uuid is in the file being sent **Error message**: 409 is returned when the uuid or name already exists in the database.

DELETE /providers/{uuid}
Removes the provider identified by uuid. **Error message**: 404 is returned when the uuid doesn't exist in the database. 409 is returned when the provider code is used.

6.4 Templates

- Templates collection URI: /templates
- Template URI: /templates/{TemplateId}

The TemplateId matches the TemplateId attribute of wsag:Template element when the template is created. A template is serialized in XML as defined by ws-agreement.

An example of template in XML is:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:Template xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement" wsag:TemplateId="netmode">
    <wsag:Name>Template for Netmode service</wsag:Name>
    <wsag:Context>
    <wsag:AgreementResponder>netmode</wsag:AgreementResponder>
```

<wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>

<wsag:ExpirationTime>2015-06-07T12:00:00+0100</wsag:ExpirationTime>

<sla:Service xmlns:sla="http://sla.atos.eu">Netmode service</sla:Service>

</wsag:Context>

<wsag:Terms>

<wsag:All>

<wsag:ServiceDescriptionTerm wsag:Name="SDTName1" wsag:ServiceName="Netmode service">

The template for Netmode service

</wsag:ServiceDescriptionTerm>

<wsag:ServiceProperties wsag:Name="NonFunctional" wsag:ServiceName="Netmode service">

<wsag:VariableSet>

<wsag:Variable wsag:Name="Availability" wsag:Metric="xs:double">

<wsag:Location>testbed</wsag:Location>

</wsag:Variable>

<wsag:Variable wsag:Name="ResourceComplianceRatio" wsag:Metric="xs:decimal">

<wsag:Location>testbed</wsag:Location>

</wsag:Variable>

</wsag:VariableSet>

</wsag:ServiceProperties>

<wsag:GuaranteeTerm wsag:Name="GT_Availablity">

<wsag:ServiceScope wsag:ServiceName="sla:testbed"/>

<wsag:ServiceLevelObjective>

<wsag:KPITarget>

<wsag:KPIName>UpTime</wsag:KPIName>

<wsag:CustomServiceLevel>

{"constraint" : "UpTime GT 0.99"}

</wsag:CustomServiceLevel>

</wsag:KPITarget>

</wsag:ServiceLevelObjective>

</wsag:GuaranteeTerm>

<wsag:GuaranteeTerm wsag:Name="GT_ResourceComplianceRatio">

<wsag:ServiceScope wsag:ServiceName="sla:testbed"/>

```
        <wsag:ServiceLevelObjective>
                <wsag:KPITarget>
                <wsag:KPIName>Performance</wsag:KPIName>
                <wsag:CustomServiceLevel>
                {"constraint" : "ResourceComplianceRatio GT 0.99"}
                </wsag:CustomServiceLevel>
                </wsag:KPITarget>
        </wsag:ServiceLevelObjective>
        </wsag:GuaranteeTerm>
        </wsag:All>
        </wsag:Terms>
</wsag:Template>
```

An example of template in JSON is:

```
{
   "templateId": "cm",
   "context": {
      "agreementInitiator": null,
      "agreementResponder": "cm",
      "serviceProvider": "AgreementResponder",
      "templateId": "4c4341ac-5fac-43a8-993e-41d1ff06dd6f",
      "service": "dummy service",
      "expirationTime": "2015-03-07T12:00:00CET"
   },
   "name": "Template for testing service",
   "terms": {
      "allTerms": {
         "serviceDescriptionTerm": {
            "name": "SDTName1",
            "serviceName": "dummy service"
         },
         "serviceProperties": [
            {
               "name": "NonFunctional",
```

```
      "serviceName": "cm service",
      "variableSet": {
        "variables": [
          {
            "name": "Availability",
            "metric": "xs:double",
            "location": "testbed/Availability"
          },
          {
            "name": "ResourceComplianceRatio",
            "metric": "xs:decimal",
            "location": "testbed/ResourceComplianceRatio"
          }
        ]
      }
    }
  ],
  "guaranteeTerms": [
    {
      "name": "GT_Availability",
      "serviceScope": {
        "serviceName": "sla:testbed",
        "value": [
          "sliver_Ids"
        ]
      },
      "qualifyingCondition": null,
      "serviceLevelObjetive": {
        "kpitarget": {
          "kpiName": "Availability",
          "customServiceLevel": "{\"constraint\" : \"Availability GT 0.99\"}"
        }
      }
    },
```

```
        {
          "name": "GT_ResourceComplianceRatio",
          "serviceScope": {
            "serviceName": "sla:testbed",
            "value": [
              "sliver_ids"
            ]
          },
          "qualifyingCondition": null,
          "serviceLevelObjetive": {
            "kpitarget": {
              "kpiName": "ResourceComplianceRatio",
              "customServiceLevel": "{\"constraint\" : \"ResourceComplianceRatio GT 0.99\"}"
            }
          }
        }
      ]
    }
  }
}
```


GET /templates/{TemplateId}
Retrieves a template identified by TemplateId. **Error message**: 404 is returned when the uuid doesn't exist in the database.

GET /templates{?serviceIds,providerId}
Parameters: • serviceIds: string with coma separated values (CSV) with the id's of service that is associated to the template

- providerId: id of the provider that is offering the template

POST /templates

Creates a new template. The file might include a TemplateId or not. In case of not beeing included, a uuid will be assigned.

Error message:

409 is returned when the uuid already exists in the database.

409 is returned when the provider uuid specified in the template doesn't exist in the database.

500 when incorrect data has been suplied

PUT /templates/{TemplateId}

Updates the template identified by TemplateId. The body might include a TemplateId or not. In case of including a TemplateId in the file, it must match with the one from the url.

Error message:

409 when the uuid from the url doesn't match with the one from the file or when the system has already an agreement associated

409 when template has agreements associated.

409 provider doesn't exist

500 when incorrect data has been suplied

DELETE /templates/{TemplateId}

Removes the template identified by TemplateId.

Error message:

409 when agreements are still associated to the template

404 is returned when the uuid doesn't exist in the database.

6.5 Agreements

- Agreements collection URI: /agreements
- Agreement URI: /agreement/{AgreementId}

The AgreementId matches the AgreementId attribute of wsag:Agreement element when the agreement is created. An example of agreement in XML is:

<wsag:Agreement wsag:AgreementId="3a14e963-d800-42b3-a17d-831ef9e15565" xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement" xmlns:sla="http://sla.atos.eu">

<wsag:Name>

Dummy aggrement

</wsag:Name>

<wsag:Context>

<wsag:AgreementInitiator>testExperimenter</wsag:AgreementInitiator>

<wsag:AgreementResponder>testbed_name</wsag:AgreementResponder>

<wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>

<wsag:ExpirationTime>2015-04-01T12:00:00CET</wsag:ExpirationTime>

<wsag:TemplateId>template01</wsag:TemplateId>

<sla:Service>Dummy service</sla:Service>

</wsag:Context>

<wsag:Terms>

<wsag:All>

<wsag:ServiceDescriptionTerm wsag:Name="SDTName1" wsag:ServiceName="dummy service"/>

<wsag:ServiceProperties wsag:Name="NonFunctional" wsag:ServiceName="dummy service">

<wsag:VariableSet>

<wsag:Variable wsag:Name="Availability" wsag:Metric="xs:double">

<wsag:Location>testbed_name/Availability</wsag:Location>

</wsag:Variable>

<wsag:Variable wsag:Name="ResourceComplianceRatio" wsag:Metric="xs:decimal">

<wsag:Location>testbed_name/ResourceComplianceRatio</wsag:Location>

</wsag:Variable>

</wsag:VariableSet>

</wsag:ServiceProperties>

```
<wsag:GuaranteeTerm wsag:Name="GT_Availability">
<wsag:ServiceScope wsag:ServiceName="sla:testbed_name">
        [<sliver Ids>]
</wsag:ServiceScope>
<wsag:ServiceLevelObjective>
        <wsag:KPITarget>
        <wsag:KPIName>Availability</wsag:KPIName>
        <wsag:CustomServiceLevel>
        {"constraint" : "UpTime GT 0.99"}
        </wsag:CustomServiceLevel>
        </wsag:KPITarget>
</wsag:ServiceLevelObjective>
</wsag:GuaranteeTerm>
<wsag:GuaranteeTerm wsag:Name="GT_ResourceComplianceRatio">
<wsag:ServiceScope wsag:ServiceName="sla:testbed_name">
        [<sliver Ids>]
</wsag:ServiceScope>
<wsag:ServiceLevelObjective>
        <wsag:KPITarget>
        <wsag:KPIName>ResourceComplianceRatio</wsag:KPIName>
        <wsag:CustomServiceLevel>
        {"constraint" : "ResourceComplianceRatio GT 0.99"}
        </wsag:CustomServiceLevel>
        </wsag:KPITarget>
</wsag:ServiceLevelObjective>
</wsag:GuaranteeTerm>
</wsag:All>
</wsag:Terms>
</wsag:Agreement>
```

An example of agreement in JSON is:

```
{
  "agreementId":"3a14e963-d800-42b3-a17d-831ef9e15565",
  "name":"ExampleAgreement",
  "context":{
    "agreementInitiator":"client-prueba",
    "expirationTime":"2014-03-07T12:00:00+0100",
    "templateId":"template02",
    "service":"service5",
    "serviceProvider":"AgreementResponder",
    "agreementResponder":"provider03"
  },
  "terms": {
    "allTerms":{
      "serviceDescriptionTerm":null,
      "serviceProperties":[
        {
          "name":"ServiceProperties",
          "serviceName":"ServiceName",
          "variableSet":{
          "variables":[
            {"name":"metric1","metric":"xs:double","location":"testbed"},
            {"name":"metric2","metric":"xs:double","location":"testbed"}
          ]
        }
      }
    ],
    "guaranteeTerms":[
      {
        "name":"GTMetric1",
        "serviceScope":{"serviceName":"ServiceName","value":"[sliver Id list]"},
        "serviceLevelObjetive":{
          "kpitarget":{
            "kpiName":"metric1",
            "customServiceLevel":"{\"constraint\" : \"metric1 BETWEEN (0.05, 1)\"}"
          }
        }
      },{
        "name":"GTMetric2",
        "serviceScope":{"serviceName":"ServiceName","value":"[sliver Id list]"},
        "serviceLevelObjetive":{
          "kpitarget":{
            "kpiName":"metric2",
            "customServiceLevel":"{\"constraint\" : \"metric2 BETWEEN (0.1, 1)\"}"
          }
        }
      }
    ]
  }
```

}

GET /agreements/{AgreementId}

Retrieves an agreement identified by AgreementId.

Error message:

404 is returned when the uuid doesn't exist in the database.

GET /agreements/

Retrieves the list of all agreements.

GET /agreements{?consumerId,providerId,templateId,active}

Parameters:

- consumerId: uuid of the consumer (value of Context/AgreementInitiator if Context/ServiceProvider equals "AgreementResponder").
- providerId: uuid of the provider (value of Context/AgreementResponder if Context/ServiceProvider equals "AgreementResponder")
- templateId: uuid of the template the agreement is based on.
- active: boolean value (value in {1,true,0,false}); if true, agreements currently enforced are returned.

GET /agreementsPerTemplateAndConsumer{?consumerId,templateUUID}

Parameters:

- consumerId: uuid of the consumer (value of Context/AgreementInitiator if Context/ServiceProvider equals "AgreementResponder").
- templateUUID: uuid of the template in wicht the agreement is based

POST /agreements

Creates a new agreement. The body might include an AgreementId or not. In case of not being included, a uuid

will be assigned. A disabled enforcement job is automatically created.

Error message:

409 is returned when the uuid already exists in the database

409 is returned when the provider uuid specified in the agreement doesn't exist in the database

409 is returned when the template uuid specified in the agreement doesn't exist in the database

500 when incorrect data has been suplied.

DELETE /agreements/{AgreementId}

Removes the agreement identified by AgreementId.

Error message:

404 is returned when the uuid doesn't exist in the database

GET /agreements/active

Returns the list of active agreements.

GET /agreements/{AgreementId}/context

Only the context from the agreement identified by AgreementId is returned.

Error message:

404 is returned when the uuid doesn't exist in the database

500 when the data agreement was recorded incorrectly and the data cannot be supplied

Request in XML

GET -H "Accept: application/xml" /agreements/{agreement-id}/context HTTP/1.1

Request in JSON

GET -H "Accept: application/json" /agreements/{agreement-id}/context HTTP/1.1

Response in XML

HTTP/1.1 200 OK

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wsag:Context xmlns:sla="http://sla.atos.eu" xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement">
   <wsag:AgreementInitiator>RandomClient</wsag:AgreementInitiator>
   <wsag:AgreementResponder>provider02</wsag:AgreementResponder>
   <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
   <wsag:ExpirationTime>2014-03-07T12:00:00CET</wsag:ExpirationTime>
   <wsag:TemplateId>template02</wsag:TemplateId>
   <sla:Service>service02</sla:Service>
</wsag:Context>
```

Response in JSON

HTTP/1.1 200 OK

```
{"AgreementInitiator":"RandomClient",
 "AgreementResponder":"provider02",
 "ServiceProvider":"AgreementResponder",
 "ExpirationTime":"2014-03-07T12:00:00CET",
 "TemplateId":"template02",
 "Service":"service02"}
```

Usage (for JSON and XML)

```
curl -H "Accept: application/xml" http://localhost:8080/sla-service/agreements/agreement01/context
curl -H "Accept: application/json" http://localhost:8080/sla-service/agreements/agreement01/context
```

GET /agreements/{AgreementId}/guaranteestatus

Gets the information of the status of the different Guarantee Terms of an agreement.

There are three available states: NON_DETERMINED, FULFILLED, VIOLATED.

Error message:

404 is returned when the uuid doesn't exist in the database

Request in XML

```
GET -H "Accept: application/xml" /agreements/{agreement-id}/guaranteestatus
HTTP/1.1
```

Request in JSON

GET -H "Accept: application/json" /agreements/{agreement-id}/guaranteestatus
HTTP/1.1

Response in XML

```
HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<guaranteestatus AgreementId="agreement02" value="FULFILLED">
   <guaranteetermstatus name="GTResponseTime" value="FULFILLED"/>
   <guaranteetermstatus name="GTPerformance" value="FULFILLED"/>
</guaranteestatus>
```

Response in JSON

```
HTTP/1.1 200 OK
{"AgreementId":"agreement02",
 "guaranteestatus":"FULFILLED",
 "guaranteeterms":
   [{"name":"GTResponseTime", "status":"FULFILLED"},
    {"name":"GTPerformance", "status":"FULFILLED"}]
 }
```

Usage (for JSON and XML)

curl -H "Accept: application/xml" http://localhost:8080/sla-service/agreements/agreement01/guaranteestatus
curl -H "Accept: application/json" http://localhost:8080/sla-service/agreements/agreement01/guaranteestatus

6.6 Enforcement Jobs

An enforcement job is the entity which starts the enforcement of the agreement guarantee terms. An agreement can be enforced only if an enforcement job, linked with it, has been previously created and started. An enforcement job is automatically created when an agreement is created, so there is no need to create one to start an enforcement.

- Enforcement jobs collection URI: /enforcements
- Enforcement job URI: /enforcements/{AgreementId}

An enforcement job is serialized in XML as:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<enforcement_job>
   <agreement_id>agreement02</agreement_id>
   <enabled>true</enabled>
   <last_executed>2014-08-13T10:01:01CEST</last_executed>
</enforcement_job>
```

An enforcement job is serialized in JSON as:

{"enabled":true,
"agreement_id":"agreement02",
"last_executed":"2014-08-13T10:01:01CEST"}

GET /enforcements/{AgreementId}

Retrieves an enforcement job identified by AgreementId.

Error message:

404 is returned when the uuid doesn't exist in the database

GET /enforcements

Retrieves the list of all enforcement job.

POST /enforcements

Creates and enforcement job. Not required anymore. The enforcement job is automatically generated when an agreement is created.

Error message:

409 is returned when an enforcement with that uuid already exists in the database

404 is returned when no agreement with uuid exists in the database

PUT /enforcements/{AgreementId}/start

Starts an enforcement job.

Error message:

403 is returned when it was not possible to start the job

Request

PUT /enforcements/{agreement-id}/start HTTP/1.1

Response in XML and JSON

HTTP/1.1 200 Ok
Content-type: application/[xml | json]

The enforcement job with agreement-uuid {agreement-id} has started

Usage (for JSON and XML)

curl -H "Accept: application/xml" -X PUT localhost:8080/sla-service/enforcements/fc923960-03fe-41/start
curl -H "Accept: application/json" -X PUT localhost:8080/sla-service/enforcements/fc923960-03fe-41/start

PUT /enforcements/{AgreementId}/stop

Stops an enforcement job

Error message:

403 is returned when it was not possible to start the job

Request

PUT /enforcements/{agreement-id}/stop HTTP/1.1

Response in XML and JSON

HTTP/1.1 200 OK
Content-type: application/[xml | json]

The enforcement job with agreement-uuid {agreement-id} has stoppped

Usage (for JSON and XML)

curl -H "Accept: application/xml" -X PUT localhost:8080/sla-service/enforcements/fc923960-03fe-41/stop
curl -H "Accept: application/json" -X PUT localhost:8080/sla-service/enforcements/fc923960-03fe-41/stop

6.7 Violations

- Violations collection URI: /violations
- Violation URI: /violations/{uuid}

A violation is serialized in XML as:

```
<violation>
  <uuid>ce0e148f-dfac-4492-bb26-ad2e9a6965ec</uuid>
  <contract_uuid>agreement04</contract_uuid>
```

```
  <service_scope></service_scope>
  <metric_name>Performance</metric_name>
  <datetime>2014-08-13T10:01:01CEST</datetime>
  <actual_value>0.09555700123360344</actual_value>
</violation>
```

A violation is serialized in JSON as:

```
{"uuid":"e431d68b-86ac-4c72-a6db-939e949b6c1",
 "datetime":"2014-08-13T10:01:01CEST",
 "contract_uuid":"agreement07",
 "service_name":"ServiceName",
 "service_scope":"",
 "metric_name":"time",
 "actual_value":"0.021749629938806803"}
```

GET /violations/{uuid}

Retrieves information from a violation identified by the uuid.

GET /violations{?agreementId,guaranteeTerm,providerId,begin,end}

Parameters:

- agreementId: if specified, search the violations of the agreement with this agreementId,
- guaranteeTerm: if specified, search the violations of the guarantee term with this name (GuaranteeTerm[@name]),
- providerId: if specified, search the violations raised by this provider.
- begin: if specified, set a lower limit of date of violations to search,
- end: if specified, set an upper limit of date of violations to search.

Error message:

404 when erroneous data is provided in the call

7 Appendix B: SLA Collector REST API

The SLA Collector is currently being located as a central component at iMinds. Similar to the REST API developed for the SLA Management module, this API will return 200 OK messages, with the requested information if any, when the query has been processed correctly. The responses are returned in JSON format.

GET /testbeds
Retrieves a list with all the testbeds that support SLAs

GET /sla/slice/{sliceId}
Retrieves a list with the SLAs under a specific slice. **Parameters**: • sliceId: urn of a slice. **Error message**: 404 when the requested slice does not contain any SLA.

POST /agreements/create{?testbed}
Creates a new agreement based on the {testbed} SLA template and redirects the call to its SLA Management module. The POST data have to be provided in JSON format and have to contain the following information: "SLIVER_INFO_AGGREGATE_URN": "testbed_urn", "SLIVER_INFO_EXPIRATION": “date in ISO format: 2014-10-29T16:17:55+01:00", "SLIVER_INFO_SLICE_URN": "slic_urn", "SLIVER_INFO_CREATOR_URN": "experimenter_urn", "SLIVER_INFO_URN": [“list with sliver Ids”], "SLIVER_INFO_SLA_ID": "SLA Id using UUID standard" **Parameters**:

- testbed: urn of the testbed.

Error message:

400 when the submitted parameters contain any error.

Errors returned from the SLA Management API.

7.1 SLA Collector as a Proxy

The following pattern is to use the SLA Collector as a proxy to make a request to a testbeds. This allows any REST operation that the SLA Management module supports.

GET | POST | PUT | DELETE /sla/{SLA Management API call}{?testbed}

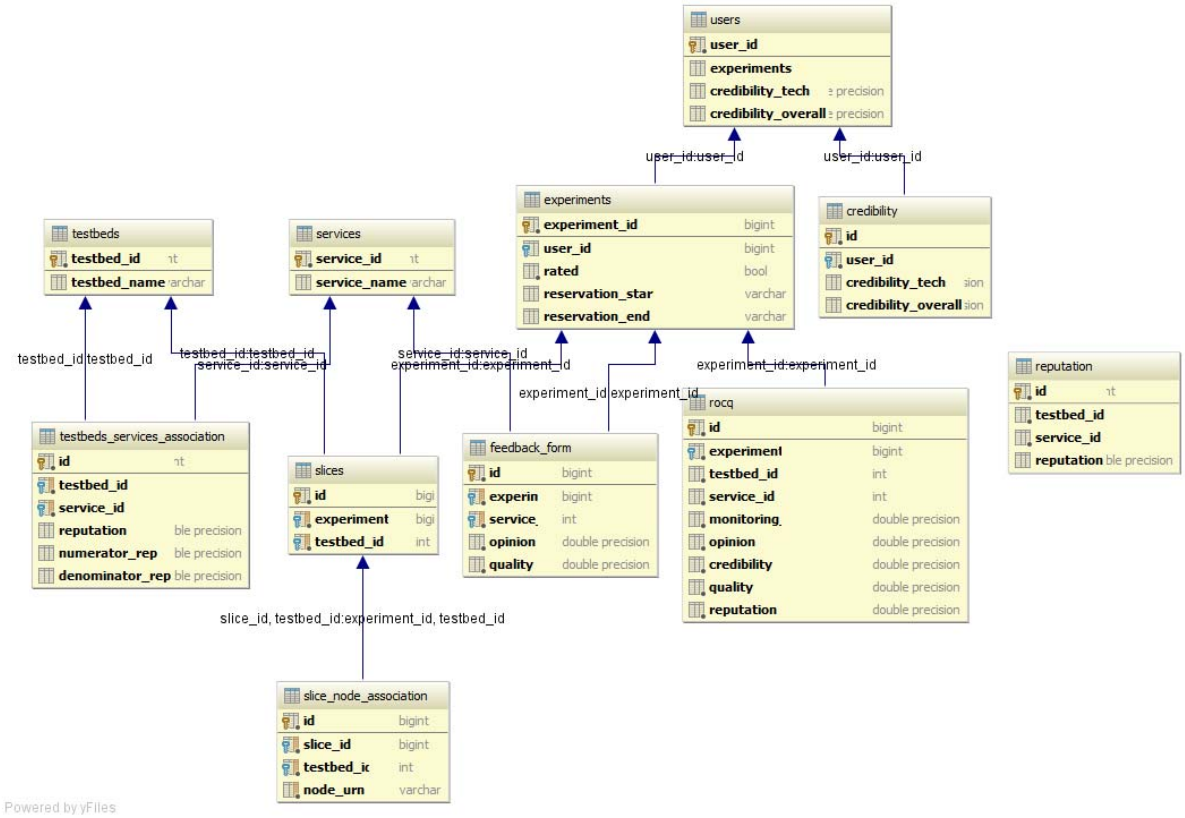
Redirects the API call to the SLA Management module of the specified testbed.

Parameters:

- SLA Management API call: request following the described API of the SLA Management module.
- testbed: urn of the testbed.

8 Appendix C: Reputation Service Repository data model

The following picture depicts the *Reputation Service Repository* data model:



9 Appendix D: Reputation Service REST Interface

This appendix lists the input/output of the *Reputation Service* REST interface.

- REST interface for Reputation Service
  - **POST / reputation/showrep:** Returns the reputation values for each testbed service.
    accept: application/json

Request Body

```
{"key": "reputation"}
```

Response

200 OK

```
[
  {
    "testbed": "omf:netmode",
    "services": [
      {
        "overall": 1
      },
      {
        "availability": 1
      }
    ]
  },
  {
    "testbed": "omf:nitos",
    "services": [
      {
        "overall": 1
      },
      {
        "availability": 0.75
      }
    ]
  }
]
```

  - **POST / reputation/qid:** Receives a list of EIDs and returns the EIDs of experiments that are not rated yet.
    accept: application/json

Request Body

```
[
```

```
 "4c970389639ad73c024e04221a291a3564386355",
 "500918a68e66ae7de48b37e12448d34d3e63bc69"
]
```

Response

200 OK

["500918a68e66ae7de48b37e12448d34d3e63bc69"]

- **POST / reputation/json** Submits a new rating for an unrated experiment.

accept: application/json

Request Body

```
{
   "start_tunix": "1417541400",
   "user_hrn": "fed4fire.global.chrisap",
   "testbeds": {
      "omf:netmode": [
         "omf.netmode.node19",
         "omf.netmode.node18"
      ]
   },
   "end_tunix": "1417545000",
   "eid": "4c970389639ad73c024e04221a291a3564386355",
   "start_t": "2014-12-02 18:30:00",
   "end_t": "2014-12-02 19:30:00",
   "slice_hrn": "fed4fire.global.chrysa",
   "user_eval": {
      "reuse": "1",
      "pay": "1",
      "support": "5",
      "overall": "5",
      "problems": "1",
      "quality": "5",
      "availability": "5"
   }
}
```

Response

200 OK

"Data Received"