SMARTSANTANDER

SEVENTH FRAMEWORK PROGRAMME

SMARTSANTANDER PROJECT

INFSO-ICT-257992 SmartSantander

D1.3

Third Cycle Architecture Specification

Contractual Date of Delivery: 30th April 2013

Actual Date of Delivery: 10th May 2013

Editor(s): UNIS

Author(s): See Authors list

Participant(s): CEA; UNIS; ALU-SP; ALU-IT; EYU; CTU; ALU-IT; UZL; UC; TID.

Work package: WP1

Estimated person months: CEA 0.5PM; UNIS 3.3PM; ALU-SP 1.5PM; ALU-IT 0.1PM; EYU 2.0; CTI 0.5; ULANC 0.5PM; UZL 2.0; UC 1.88; TID 0.2.

Security: Public

Version: 1.0

Abstract: This deliverable presents the SmartSantander architecture evolution for the third cycle of the project. New use cases have been addressed aiming at supporting additional functionalities such as federation or experimentation on top smart phones. These new use cases along with the winning proposals of the first Open Call have driven the evolution of the architecture definition and its realisation.

Keyword list: *Architecture, components, development phases, DTN, federation, virtualization.*

Authors

Organisation	Author(s)
UNIS	Alex Gluhak
UC	Luis Munoz, Pablo Sotres, Luis Sanchez
CEA	Pierre Roux
ALU-SP	Beatriz Sanchez
CTI	Evangelos Theodoridis
ULANC	Rajiv Ramdhany
UZL	Sebastian Ebers, Daniel Bimschas
EYU	Srdjan Krco, Stevan Jokic
TID	Ana Leticia Hernández

Table of Contents

List of Figures

List of Tables

Acronyms and Abbreviations

AAA	**Authentication, Authorisation and Accounting**
ACI	Access Control Interface
ASI	Application support interface
CM	Configuration Management
DoS	Denial of Service
ESI	Experimental Support Interface
FCAPS	Fault, Configuration, Accounting, Performance, Security
FM	Fault Management
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
MSI	Management support interface
NMS	Network Management System
OLAP	*On-Line Analytical Processing*
OSA/V	OS Abstraction and Virtualisation
OTA	Over-The-Air
OTAP	Over-The-Air Provisioning
PM	Performance Management
RFID	Radio Frequency Identification
SAN	Sensor Area Network

TLS	Transport Layer Security
WP	Work Package
WSN	Wireless Sensor Network
SFA	Slice Federation Architecture
OML	OMF Measurement Library
OMF	Orbit Management Framework
VTD	Virtual Testbed Device
DTN	Delay Tolerant Networking
FIRE	Future Internet Research and Experimentation
M2M	Machine to Machine
LXC	Linux Containers
GPS	Geo-Positioning System
OEDL	OMF Experiment Description Language
TRED	Testbed Runtime Experiment Description
EC	Experiment Controllers
AM	Aggregate Manager
RC	Resource Controller
MP	Measurement Points
SQL	Syntax Query Language
RD	Resource Directory
DB	Database

WAN	Wide Area Network
GW	Gateway
PS	Portal Server
FRXX	Functional Requirement XX
USN	Ubiquitous Sensor Network
SE	Smartphone Experimentation
SEI	Smartphone Experimentation Interface
NFR	Non-Functional Requirement
DoS	Denial of Service
O&M	Observations and Measurements
OTAP	Over The Air Programming
SNAA	Sensor Network and Authorization
TR	Testbed Runtime
URN	Uniform Resource Name
CT	Custody Transfer

1. EXECUTIVE SUMMARY

This deliverable presents the third and final cycle architecture of the SmartSantander facility, which is built incrementally on the basis of the second cycle architecture presented in deliverable [D1.2]. The process in justifying this architecture update is thoroughly presented in the document in the form of motivating use cases, derived requirements and further architectural specifications.

As a starting point for the specification process served a set of recommendation from the expert reviewers, which highlighted further gaps of the second cycle architecture and new ideas of how to make the facility more useful for a broader set of community stakeholders. Further ideas where motivated from emerging interests among the consortium partners. Community needs as of the second open call for experiments where considered, however did not result in additional use case demands.

Based on these recommendations and ideas, a set of use cases have been developed. The use cases cover novel interactions of experimenters with the testbed and provide a basis for the motivation of detailed functional and non-functional requirements. Overall the use cases span two different federation scenarios, extensions to allow for experimentation with more powerful IoT nodes such as mobile phones and gateway tier devices as well as DTN based experiment configuration and data collection. In addition security and privacy implications have been considered for end-users participating in an experiment, e.g. via Smartphones.

Overall 46 new functional and non-functional requirements have been derived for different subsystem functionalities. The requirements were further analysed and compared with the features and capabilities of existing system functions of the second cycle architecture. In the case that a functional requirement could not be fulfilled with the existing specification, a further analysis was performed in order to accommodate the open requirement into the architecture. As the goal was not to unnecessary increase the complexity of the final architecture, it was first verified whether existing system functions could be modified or extended in order to accommodate a (set of) requirements. New system functions where only specified in cases this was found not to be possible.

A major architectural evolution represents the introduction of a new subsystem for *Federation* support. It addresses the need for different SmartSantander testbed sites to be federated among each *or internally* by the introduction of the *Federator*. A second requirement addressed by the Federation support is the *external* Federation with other testbeds in the FIRE community. These include the Slice Federation Architecture (SFA) for managing discovery, access and reservation of experimentation resources and Orbit Management Framework (OMF) for experimentation control and OMF Measurement Library (OML) for experiment data retrieval.

The third cycle architecture introduces also new system functions to existing subsystems for the support for experiments that rely on Smartphones as experimentation resource. This resulted in both updates to system functions on server and IoT device tier as well as introduction of new service functions. These functions now allow effective configuration of experiments for Smartphones that respect user preferences on resource use and privacy. Furthermore they allow efficient scheduling of experiments, provisioning of experimentation code and experimental data collection.

In order to support experimentation with scheduled or opportunistic contacts of mobile experimentation resources with the testbed infrastructure, delay tolerant network (DTN) capabilities have been added to the architecture. As a result provisioning of experimentation code or the collection of experimentation data does not have to rely any more on permanent connections to the testbed backbone infrastructure. This is achieved by an upgrade of several existing components of the architecture in all three tiers of the architecture.

A further new feature that the architecture introduces is the support for the execution of experimentation on GW tier devices. For this the *Virtual Testbed Device* (VTD) support that has been initial introduced in the 2nd cycle architecture specification is utilised and further enhanced by the addition of the *VTD scheduler* on the configuration subsystem on the GW tier. The latter manages the priority of experimentation code to be run on GW tier devices and that they do not impact the performance of the primary system functions.

The final set of upgrade to existing system functions addresses the challenges of increased end user privacy in experimentation. Both PSense Client and PSense Server components of the application subsystem are upgraded with security functions such as encryption.

A final contribution presented in this document are initial design considerations and guidelines that serve as starting point for the detailed design and implementation of the newly specified architectural features. They serve as input for further technical work in WP2 and WP3.

2. INTRODUCTION

The initial SmartSantander architecture [D1.1] has been conceived within the first year of the project and proved itself as a solid foundation for the experimental facility design and implementation. The project selected a three phase spiral approach for its design and realisation, in which an initial architecture from the first cycle is incrementally enhanced and modified to accommodate new features that satisfy emerging requirements of the community. This document provides the third and final update to the architecture specifications and builds upon the second phase specification [D1.2].

The third architecture realisation cycle takes mainly into consideration the recommendations made at the second year review by the project reviewers, requirements from the FIRE community of projects through the FIA architecture board. It also considers the requirements from the second call projects, which are in most cases already supported by the second cycle architecture release.

More specifically the third architecture introduces the following new features:

- Federation support for different SmartSantander testbed sites. This allows all users of individual testbed sites to access all other testbeds within SmartSantander and to run experiments that utilise resources across the different testbeds.
- Federation support with other FIRE testbeds. New extensions have been developed to make the SmartSantander architecture compliant towards the SFA and OMF/OML. This will allow the SmartSantander testbed to become accessible by users of testbeds that are compliant to these specifications and testbed resources to be used in joined experimentation across the respective testbeds.
- DTN based communication support for experiment configuration and data collection. The experimentation management plane of the testbed does not need to rely any more on continuous connectivity through dedicated infrastructure, but can instead take advantage of opportunistic contacts of experimentation resources with infrastructure devices.
- Experimentation support for more powerful SmartSantander nodes such as gateway nodes. This is based on the virtual testbed device (VTD) concept already introduced in the second cycle architecture and adapted to provide resource containment features and life-cycle management.
- Experimentation support for mobile phones. It allows smart phones not only to be used as mere sensor data collectors as in the case of participatory sensing application, but also to integrate these as full experimentation resources into the SmartSantander framework. Experimenters are now able to install experimentation code on the smart phones and to perform the collection of statistics during experiments.
- Updated security, privacy and trust mechanisms that focus not only on protecting the testbed infrastructure but also end-users that are directly or indirectly involved in the experiment execution and data collection.
- Support for all new experiments from the second open call.

The remaining document is structured as follows. Section 3 provides an overview of third cycle use cases and requirements which have been developed as part of the work on IR1.4. Section 4 provides an overview of the third cycle architecture and provides detailed specification of the corresponding features that have been introduced in this final iteration. Section 5 outlines design considerations for the realisation of the newly specified featured as starting points for the detailed design and implementation activities in WP2 and WP3. Concluding remarks are provided in section 6.

3. THIRD CYCLE USE CASES AND REQUIREMENTS

With an increasing maturity of the SmartSantander experimentation facility, the third cycle architecture design was initially planned to focus the development of federation capabilities. However due to emerging demands, further additional use cases have been conceived aiming at supporting new experimenting and services provision possibilities.

On one hand, the defined set of use cases pursues capabilities of extensibility of the underlying experimentation platform. This includes federation capabilities, the virtualization of nodes and the easiness of including new experiments as well as new testbed sites. On the other hand, enhancing the end-user experience is also important, taking into account the new smartphones, the security and privacy of end user data, and how he/she connects to the network.

The new use cases comprise:

1. Federation Use Cases, both intra SmartSantander and with other experimentation facilities.
2. Delay Tolerant Network Use Cases: Efficient experimentation code and data collection downloading relaying on opportunistic network connection, using different wireless infrastructure possibilities depending on the environment.
3. Virtualization on top of powerful SmartSantander nodes such the gateways.
4. Execution of experiments on top of smart phones aiming at extending the capabilities so far exploited in the participatory sensing scenario.
5. Trust Security and Privacy from the perspective of the end-users, focusing on the current end-user worries about the data exchange and privacy.

3.1. Federation Use Cases

SmartSantander considers two different cases of federation: The first one is about allowing experiments which span across all testbed sites of the consortium of the SmartSantander project. This kind of federation allows users to utilize nodes of different testbed sites in a single experiment.

The second one is to open the SmartSantander experiment platform to other coexisting experimentation platforms. In particular, the work being carried out in FED4FIRE project [FED4FIRE] is one of the most promising approaches for enabling the federation with other FIRE facilities.

While the first type aims to provide a large and comparatively homogeneous WSN testbed, the second one will open SmartSantander to other experimental facilities which are not necessarily restricted to Internet of Things technologies.

3.1.1. Intra Smart Santander Federation

The SmartSantander facility is made of five main deployments sites, Guildford, Lübeck, Melbourne, Pancevo and Santander. Aiming at providing a uniform authentication and integrated resource

reservation framework the appropriate enablers and mechanisms have to be defined. In the picture bellow the interaction between the experimenter and the platform manager with the experimental facility is depicted.

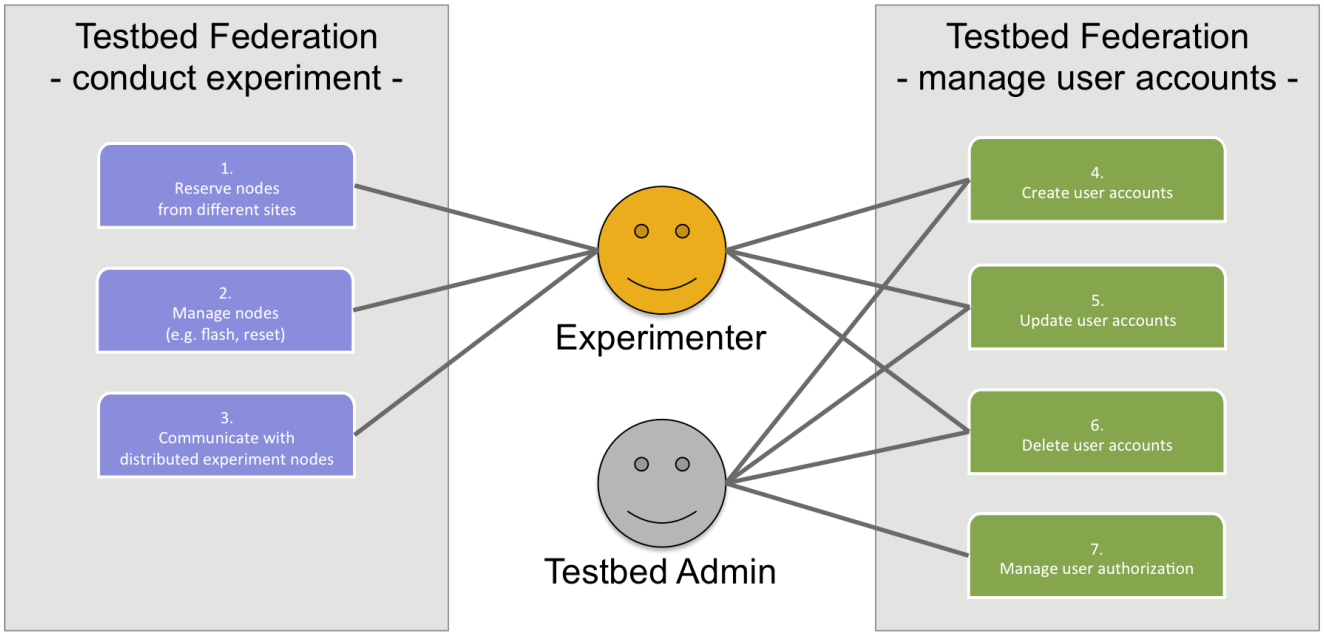

Figure 1: Intra SmartSantander Federation

In order to simplify the process of distributed (i.e. federated) experimentation the experimenter should not be concerned with the “internal” details of the federation (e.g. the API endpoints of the individual testbed sites). He is merely presented with the available set of node URNs and the corresponding meta data (in form of a WiseML document) and connect his client application to an endpoint provided by the federations “Federator” component, i.e. for a federated experiment a client must connect only to the federator and not to the individual testbeds. Below, a short description of the individual experimenter use cases is given:

1. By reserving nodes from different sites an experimenter creates a distributed experiment spanning several SmartSantander sites. For this the experimenter retrieves the federated testbed meta data (i.e. the merged WiseML description of the federated sites), chooses a subset of nodes from a subset of testbeds and reserves these nodes by simply referring to the individual node URNs.

2. Experimenters can manage (e.g. flash, reset) nodes of a distributed experiment independently of their physical location and testbed, again, by simply referring to the node URNs.

3. Communication with the distributed experiment nodes happens in the same manner as with nodes of one testbed. Every message sent to the nodes (a.k.a. downstream message) can be sent as a unicast or multicast message and every response or output from individual nodes is forwarded to the experimenter.

In addition to the experimentation use cases just described, an additional set of use cases for managing user accounts is briefly described below:

4. The creation of user accounts can be done by both experimenters and testbed admins. In order to conduct a distributed experiment a user account has to be created on each of the federated testbeds in use, i.e. there is no Single sign-on, common user or authorization database shared by the individual testbeds. However, a front end can be used that allows the creation of "the same" user account on multiple testbeds simultaneously, so that the user credentials are identical on each testbed and the process effort is reduced to a minimum.

5. Updating user accounts (including password change) can also be done by both the experimenters and the testbed admins. While the experimenter can only update his own account on each testbed, the admins of the individual testbeds have the right to update any account in their domain.

6. The deletion of user accounts can be done by both the experimenters and the testbed admins. While the experimenter can only delete his own account on each testbed, the admins of the individual testbeds have the right to delete any account in their domain.

7. Finally, fine-grained authorization of users is managed by the individual testbed administrators, giving them full control over who is allowed to access which resources in their testbed site. To simplify the process of managing authorization the users can be assigned with roles. Each role has a set of certain access rights to a specific node set. Users that are assigned a role inherit these access rights.

3.1.2. Exposing SmartSantander to External Experimentation Facilities

Use Case: Experiment involving multiple FIRE testbeds

An SME with large expertise in urban traffic management is trying to build a new driver guidance service based on real-time traffic information. Using the speed and location information gathered from the sensors installed on vehicles as well as the data reported by traffic monitoring sensors at the main entrances of the city and the noise level sensors deployed at Santander's main avenues, they have created an algorithm that is able to create both real-time traffic maps and ten minutes ahead predictions of traffic status in the city. Combining this algorithm with route planners, this SME is planning to create an App for mobile devices that will guide drivers away from traffic jams.

Besides the access to the raw data that SmartSantander testbed is able to provide, the bandwidth and computing requirements that this service implies is too high for SME financial capabilities. Thus, they are planning to use several FIRE research testbeds to make the testing of their initial developments.

They are in contact with a FI networking testbed ready to serve high and dynamic interconnection demands based on OpenFlow so that they can test which are the best options for taking the information from the sensors to the Cloud infrastructure they are using for making the algorithms calculations and exposing the service. There is a huge amount of information to be used in the traffic

maps calculations and it is quite important to have it on a strict time manner. Moreover, they are planning to use a Cloud testbed for all storage, computing and service provisioning aspects. In this sense, depending on the amount of users and available information from the sensor infrastructure, computation capacity might be adjusted dynamically. The SME wants to develop mechanisms that allow them to estimate this necessities so that they can anticipate them and not incur in overspending (i.e. paying for capacities not used) or underestimation (i.e. failure in service provision due to not enough resources).

Enabling common mechanisms that enable this SME to seamlessly access to the capabilities of these three heterogeneous testbeds is of paramount importance. In other case, an experimenter on similar circumstances would need to learn how to run experiments on each of the testbeds. Supporting the use of SmartSantander facility in conjunction with other FIRE testbeds by sharing a uniform experiment life-cycle management increases its usability since it allows to use it on experiments with a holistic Future Internet scope not restricted to IoT only.

3.2. Experimentation Software Updates through Scheduled/Opportunistic Contacts

Two connectivity-related factors are notable in the exploitation of the SmartSantander infrastructure: **performance** and **costs**. In particular, certain urban exploitation scenarios such as target-tracking, traffic monitoring may involve the transmission of large volumes of sensor data. Similarly, experimentation often requires experimenter code to be downloaded on the sensor nodes for node-reprogramming. The fact that occasional software updates in the form of image-based reprogramming may be required for applications has to be considered as well.

When nodes are static and accessible through high bandwidth and reliable Internet connection such as IEEE 802.11, the performance associated with the above mentioned tasks are within operational limits and data transfer costs are kept low. However, nodes such as mobile sensing nodes and nodes deployed in remote locations are characterised by low-bandwidth transmissions and/or intermittent connectivity. In these situations, the traditional solution is to resort to WAN connectivity solutions such as GPRS to ensure a long-lasting transmission link for updating software on the nodes. However, cellular internet connectivity is slow (the impending rollout of 4G notwithstanding) and incur high costs when data transfers are frequent and relatively large updates have to be uploaded. We therefore explore the possibility of using Delay/Disruption Tolerant network technology to circumvent the low data rates and potentially high running costs associated with cellular internet.

DTNs overcome the problems associated with intermittent connectivity, long variable delay, asymmetric data rates and high error rates by using store-and-forward message switching. Messages (whole blocks of user data) or fragments of such messages are moved/forwarded from a storage place on one node (DTN routers) to a storage place on another node (another DTN router) along a path that eventually reaches the destination node. DTN routers must therefore hold messages in persistent storage as a communication link to the next hop may not be available for a relatively long time. When mobile sensor nodes, e.g. nodes mounted on top of vehicles move, in and out of range of static gateways, the intermittent connectivity causes network partitions which cause loss of data (TCP

reacts to packet drops with slower retransmission timings or ends the session). DTNs are able to withstand network partitions caused by mobile nodes by forwarding messages to DTN routers along the path of the mobile node. They use store-and-forward techniques to ensure that intermittently connected nodes are able to receive expected messages through contacts with DTN routers. This contact can be **opportunistic** or **scheduled** (if the motion path of the node is known) and **short-lived** or **long-lived**. **Duration of contact** and **contact-predictability** are key factors that influence the choice of routing protocols to be used in the DTN.

In the context of mobile sensing in SmartSantander, we identify one potential use of DTN protocols for facilitating the reprogramming of IoT nodes mounted on top of buses (for environmental monitoring and fleet management). Other interesting DTN use cases are theoretically possible such as using the mobile nodes as DTN routers (i.e. *data ferries)* to collect sensor readings from islands of isolated sensor nodes and transfer them to the USN data portal via *opportunistic* contacts with WiFi hotspots. However, only use cases can be realised within the remaining duration of the project are documented here.

Use Case: DTN-Enabled Software Updates for Mobile Sensor Nodes

As WAN connections via GPRS are slow and costly for downloading software images on mobile sensor nodes for experimentation purposes, we exploit the scheduled contact of buses with Wi-Fi hotspots at the bus depot to enable this feature. Thus, when a set of mobile sensor nodes are reserved for a particular experiment, the Experiment Scheduler sub-system computes a deployment plan to optimally upload the software image on the required nodes based on the following parameters: *experiment start time*, *size of software image*, *scheduled contact of sensor node with Wi-Fi hotspot*, *duration of contact*, *data rate available during connection*. The deployment plan specifies the set of DTN routers where contact is predicted to be made, the expected duration of the contact, the subset of mobile nodes that need to be reprogrammed and the set of software image fragments to upload.

Once a deployment plan has been computed, it is scheduled for execution such that software image fragments are transferred via multicast protocols to the set of DTN routers **before** the predicted contacts. DTN routers at the Bus Depot form a multicast group for software image fragment dissemination. Retransmission delays due to data transfer packet errors must be factored in to ensure a timely delivery of software image fragments to the DTN routers at the Bus Depot. The expected time of contact can be estimated from the scheduled arrival of the buses at the Bus Depot.

DTN routers also act as IoT gateway nodes and therefore run the functionality for IoT node detection and registration and for node OTAP. Once the DTN routers have received the software image fragments and a copy of the deployment plan, they subscribe to the event notifying the availability of the target mobile nodes. When bus-mounted nodes arrive at the depot, they are detected by the NodeManager component, which in turn notifies the DTN router of the node's availability. The DTN router verifies that the duration of the node's scheduled contact is long enough for the transfer of the software image (accurate data rates are obtained to particular nodes by frequent node-link probing). A successful check results in the DTN gateway to proceed with the software image fragments transfer. By sending Deployment Completion Status message, the DTN gateway informs the Experiment

Scheduler with the result of the attempted deployment. A Deployment Plan may span over a number of scheduled contacts at the Bus Depot if the contact duration is not long-lived enough for the transfer of all the software image fragments.

Bus arrival and departure times are only indicative and nodes can leave the Bus Depot during software fragment transfers. The Deployment Completion Status plan therefore states the progress status of the node reprogramming attempt. The Experiment Scheduler sub-system can decide to use WAN connections to the node or other short-lived opportunistic contacts with DTN routers along the bus route (e.g. at bus stops) to finish the node software update if it was interrupted unexpectedly and an experiment is due to start before the next scheduled contact.

Alternatively, if an experiment has to start and cannot wait for the scheduled Wi-Fi contact (normally preferred) at the Bus Depot, DTN gateways that lie along the route of the buses may be used. The fact that buses have to conform to particular routes and schedules can be exploited. It is for instance known that buses will spend about 1-2 minutes at bus stops for picking passengers. Hence, if the Experiment Scheduler knows the set of scheduled short-lived contacts with Wi-Fi-enabled DTN routers (located near bus stops preferably) as well as the data rates of the links with the stopped mobile nodes, it can determine the set of software image fragments to send to a particular DTN router. The dissemination of software image fragments has to be epidemic to take into account the possibility of corrupted fragments or busses not stopping at particular bus stops (no passenger alighting/mounting). Thus, an experimentation node on top of a bus will collect software fragments in a redundant manner from DTN routers along the bus route. Once the whole image has been reconstituted, it sends back an acknowledgement to the Experiment Scheduler through an opportunistic contact with another DTN router. It may receive a directive to start the experiment from the Experiment Scheduler if all participating nodes have been similarly reprogrammed.

The experiment software dissemination to the reserved nodes is best-effort since there is no guarantee that all selected nodes will be reprogrammed prior to the beginning of an experiment. Thus at the scheduled start of an experiment, only the nodes with full copies of the experiment code will participate in the experiment.

3.3. Virtualization on top of powerful Smart Santander nodes

The current SmartSantander architecture supports the experimentation life-cycle with resources of the IoT node tier and the creation of services and applications that exploit the IoT generated data through the USN subsystem.

However more complex experiments may require the execution of code on *gateway tier* or *server tier* resources. While the these tiers are defined in the SmartSantander architecture and physically utilised for the current testbed realisation, resources of these tiers are currently not directly accessible and managed through the SmartSantander/WISEBED testbed run-time environment. In other words, users cannot easily discover such resources in the testbed, allocate these for experiments, deploy code on them or collect experimental results/statistics through a well- defined API. This section

provides initial consideration for the utilisation of *gateway tier* resources as part of the SmartSantander experiments.

Properties and constraints of GW tier resources

In the following we briefly describe some of the main characteristics and constraints for gateway tier resources, which are important to understand when designing a solution for their integration in the experimentation life-cycle:

- Gateway tier resources provide backbone connectivity services for nodes of the IoT node tier
- Gateway tier resources are typically more powerful than IoT node tier resources, however less powerful than server tier resources. Typical gateway tier resources used in the testbeds are small laptops (Luebeck), embedded ARM5 based servers such as Plugcomputers (Guildford) or low end x86 board (Meshlium).
- Like IoT resources, GW tier resources are heterogeneous and may support different operating systems, allow the execution of programs/scripts of different languages and provide different wireless/wired communication interfaces
- GW tier resources run all software necessary for the management of IoT nodes that are attached to them, e.g. testbed run time end point.

Potential use cases of gateway tier resources in experiments

There may be various reasons why an experimenter would like to utilise gateway tier resources for experiments. On one side, an experimenter may be interested in studying the performance of solutions that directly target the IoT gateway tier. This could be a new M2M gateway protocol or an edge router solution for 6LoWPAN. On the other side a user may also want to use GW tier resources to support more realistic experimentation among IoT nodes. Some examples of the latter a provided as follows:

- Enable interactions between server tier application components and IoT node components under realistic conditions
- Enable interaction among multiple, potentially heterogeneous IoT node technologies
- Utilise resources on gateway tier devices to extend the computational or communication capabilities of IoT nodes
- Utilise resources on gateway tier devices to emulate IoT node capabilities

Making gateway resources available for experimentation

There are different possible ways how gateway resources can be made available for experimentation in the SmartSantander testbed. The resulting options may be subject to different constraints and come at different implementation complexity, pose different risks to testbed or provide certain constraints to the experiments. In the following we only provide a potential list of options and leave their analysis to later discussions.

Options for gateway resource interactions:

- Simple remote login, data transfer (e.g. using SSH, sftp)
- Through a well-defined testbed API, providing similar abstractions as the for IoT nodes

Options for code execution on gateway resources:

- Allowing the creation of process(es) on top of the host OS, exploiting only the OS native user management and access control functions. This is relatively simple to realise and exploits the OS's capability to schedule processes for execution. However, it presents a system's vulnerability in that a misbehaving experiment code can monopolise the CPU and may cause the remaining OS processes to be unresponsive.
- Providing sand-boxed execution of experimentation code through lightweight container tools such as Linux Containers (LXC) [LXC]. Containers effectively partition the resources managed by a single operating system into isolated groups to better balance the conflicting demands on resource usage between the isolated groups. In contrast to virtualization, neither instruction-level emulation nor just-in-time compilation is required. Containers can run instructions native to the core CPU without any special interpretation mechanisms. None of the complexities of para-virtualization or system call thunking[1] are required either. By providing a way to create and enter containers, an operating system gives applications the illusion of running on a separate machine while at the same time sharing many of the underlying resources.
- Middleware-level application segregation and scheduling. This involves the use of a middleware-level application segregation system to specify quantitative constraints on the share allocated to applications for each node resource and to enforce these constraints through appropriate scheduling policies. The middleware framework encompasses a set of resource scheduling mechanisms and policies aimed at maximising application performance without violating resource limitations. Applications are scheduled for execution using real-time fixed-priority round-robin scheduling i.e. they get time slices of constant duration. Low-priority tasks are pre-empted by high-priority processes. Experiments will always run at a priority lower than the middleware's administrative process, thus barring the opportunity for misbehaving experiment applications to monopolise the CPU. An example implementation of

[1] System call thunking refers to the interception and transformation of system calls to run alternative routines. For example, translating a system call from 32-bit code to 16-bit code in a virtual machine.

this type of middleware-level application segregation and scheduling framework is described in more details in [Anglano].

- Hardware virtualisation, allowing the creation of one or more VM on top of the gateway node.

3.4. Virtualization on top of Smartphones

As mentioned before, the current SmartSantander architecture supports the experimentation life-cycle with resources of the IoT node tier and the creation of services and applications that exploit the IoT generated data through the IDAS. In the context of WP4, experimentation on smartphones is performed, with the realisation of the "Pace of the City" and "Augmented Reality" applications. In these complex experiments, experimentation source code is required at both aforementioned two levels: native iOS and ANDROID applications are deployed on smartphones and custom web services are deployed at server level, for exploiting the gathered data from mobile phone applications through IDAS and for interacting with these applications for data exchange. In the current approach smartphones and experiment lifecycle are not directly managed by the SmartSantander platform as the experimentation code is deployed and executed on smartphones through the corresponding market stores of the iPhone/Android platforms.

In order to allow more generic experimentation with Smartphone resources, it is desired that the platform would support a comparable experimentation life-cycle management as with IoT resources. However there are several differences as to the ownership of the underlying experimentation platforms, connectivity, configuration capabilities and constraints. In the following several use cases are discussed from different stake holder viewpoints in order to motivate requirements for this envisioned feature.

View Point of Smartphone Owner

The following use case describes the participation of a SmartPhone as an experimentation resource in the SmartSantander facility from the view point of the smartphone owner. The scenario covers the following aspects:

- Bootstrapping – how a smartphone can become an experimentation resource
- Personalisation – Setting of resource restrictions for potential experiments and other experimentation preferences
- Opportunistic experimentation – An experiment executes in the background on the device without direct user involvement
- Interactive experimentation – An experiment executes on the device which requires at times direct user involvement as part of the experiment
- User incentivisation – a possible idea how users can be incentivised to participate more frequently in experiments
- Withdrawal from an on-going experiment

Use case scenario

Pedro is interested in making his Android smartphone available as an experimentation resource to the SmartSantander facility, in order to contribute to the development of innovative services for the benefit of his community. He visits the SmartSantander website and clicks on a link that redirects him to google play appstore, in which he can find the SmartSantander *ShareMyPhone* application.

After installation of the *ShareMyPhone* application, Pedro launches it for the first time. A privacy configuration assistant appears that guides him through several configuration options. This includes settings about what sensors, storage resources and communication devices he wishes to share for experiments, periods of time during which he may not like to provide to his device for experiments and whether he wants to be explicit informed about an experimentation request and provide consent to it. More advanced configuration options allow him to change default thresholds for experiment execution based on actual smartphone resource availability such as remaining power, CPU load or memory utilisation as well as data transmission restriction on different network interfaces. Pedro opts for making all sensors available apart from the microphone and GPS. As he has an unlimited 3G data plan with his operator, he consents to the occasional use of mobile network data traffic should he not be in Wifi coverage. Pedro also requests an implicit notification for an experiment. After Pedro finalises the setup wizard the application thanks him for participation and informs that he can modify his preferences at any time. The application synchronises the settings briefly with a network server in the SmartSantander testbed.

Several days later Pedro receives a notification for a first experiment, which comes with a brief description of the nature and what on smart phone resources are required and the duration and frequency of data collection and estimated data traffic requirements. Pedro accepts the experiment which is downloaded and then executed on his devices on a prescheduled time. The experiment captures the trajectories of the users when bicycling. The experimentation code also detects puddles, obstacles etc. by evaluating different frequency/granularity levels of GPS recording, various machine learning algorithms or signal processing methods for detecting puddles, obstacles etc. The experiment successfully completes after 1 week and he is provided with 1000 Santander points. His ranking is updated in a wall of fame on the city website, which lists top 100 contributors.

At a later stage the experiment is repeated with a semi-supervised machine learning algorithm that requires occasional manual labeling of the data set by the user. Pedro consents to it although the experiment is expected to last for a two week period. After each detected bike journey, Pedro is asked to provide additional information that is used for labeling the data captured by the experimentation code.

One week into the experiment, Pedro decides to withdraw from the experiment as he is expected to travel for a business trip on short notice. He opens the application, selects the current experiment and revokes his consent for participation. *ShareMyPhone* application synchronizes the remaining data with the experimentation data base of the network server and uninstalls the experiment from his phone.

View Point of Experimenter

The following use case describes the participation of a SmartPhone as an experimentation resource in the SmartSantander facility from the view point of the experimenter. The scenario covers the following aspects:

- Interaction with the SmartSantander facility for resource selection
- Interaction with the SmartSantander facility for experiment submission
    - source code of experiment
    - various parameters of the experiment
- Experiment agreement describing the data recorded by the experiment and how they are going to be used

Use case scenario

A research team would like to perform a complex experiment studying mobilization of bicycles in urban environments using users with smartphones. The experiment they propose is twofold:

- They would like to experiment on methods for bicycling context and event sensing like detecting puddles, obstacles, avg. speed when moving around the city. Furthermore, along with this they would like to experiment on implementing the aforementioned event sensing with minimum resource consumption (CPU time, memory, energy) so they would like to evaluate a number of different frequency/granularity levels of GPS recording, various machine learning algorithms or signal processing methods for detecting puddles, obstacles etc from smartphone sensor signals.
- Furthermore, at service level they would like to test several algorithms for trajectory recommendation for proposing to users alternative paths (most frequent, safer, and fastest). In order to do that the plan is to evaluate several data mining algorithms for trajectory recommendation, optimization algorithms, multivariate analysis methods etc.

In order to perform their experiment, experimenters should register to the SmartSantander platform. They should describe their experiment in depth, the various data recorded from smartphones and how they are going to maintain the anonymity and privacy of the end-users. They select a period of time for their experiment. They write their experimentation code either in native Android source code or in a proper scripting language, and along with the proper accompanying data files they upload their experiment in the SmartSantander platform. The experiment they propose is divided in two phases, by the end of the first phase they update the source code and data files of their experiment and re-upload it in the SmartSantander platform:

- In the first phase of the experiment, they implement a smartphone application which records the GPS traces of the users, the accelerometer signals, battery level etc. and then by various methods they generate events (obstacle, paddle etc) that are shown to the user. The user then confirms the validity of each event.

- At the end of the first phase of the experiments, experimenters gather from the SmartSantander platform all the recorded data. They process the recorded data and the feedback of users and later on fine tune their smartphone application in order to produce more accurate events. Furthermore, they process all recorder bicycle trajectories and build a trajectory recommendation model for suggesting an “optimal” bicycle route.

- In the second phase of the experimenters they upload the new optimized version of their application that beside the more accurate event detection the show to the end-users recommendations of bicycle routes according to several criteria (safest, fastest etc.). End-users rank the provided recommendations.

- At the end of the second face experimenters gather once more all recorded datasets and user-feedback, in order to evaluate once more the event detection methods and the trajectory recommendation models.

View Point of SmartSantander Manager

The following use case describes the participation of a Smartphone as an experimentation resource in the SmartSantander facility from the view point of the SmartSantander Manager (facility). The scenario covers the following aspects:

- Bootstrapping – how a smartphone can become an experimentation resource

- Reservation of smartphone devices for an experiment

- Deployment of smartphone experiment to the devices

- Maintenance of the anonymity and privacy of the smartphone owners

- Alternative ways of communication of smartphone experiments with the SmartSantander platform.

Use case scenario

SmartSantander experiment manager receives a new smartphone experiment. He inspects the description of the experiment, the source code and accompanying data files. He validates and confirms that the experiment will maintain the anonymity and privacy of the end users. Moreover, he sets various parameters regarding the experiment like the minimum/maximum number of smartphone devices that can participate to the experiment. Furthermore, he configures the experimentation parameters for the collection of datasets from the smartphones. After this setup phase, he activates the experiment either by enabling the experiment scripts to be downloadable by the smartphones or by uploading the experimentation code to an online store to make it available for download by interested end users. By the end of the period of the experiment he disables the experiment by removing the experimentation source code/app and by stopping the data gathering interface.

This aforementioned use cases envision a closer incorporation of smartphones in the SmartSantander platform and an experiment execution management in similar way as the rest IoT deployed nodes. The SmartSantander platform will register a number of smartphones, running a proper mobile phone application capable to interact with the main platform. Experimenters will upload

the mobile phone experimentation code (source or binary) to the SmartSantander platform and will reserve a fraction of SmartSantander smartphone nodes for a specific time interval. The platform will inspect the experimentation code (for privacy issues) and then at the selected time interval will deploy the code on the smartphones through the proper mobile phone application. This application will execute the experimentation code on the mobile phone for the required time interval and record all experimentation data that later on are pushed back on the SmartSantander platform. Finally, experimenter will retrieve the gathered experimentation data. There are several alternative ways to deploy experiments on smartphones and several ways to forward experimentation data on the SmartSantander platform (synchronous or asynchronous).

3.5. Trust Security and Privacy from the perspective of the end-users

User privacy preservation is a well-known concern in the Internet world. As the Internet is playing an ever growing role by enabling more and more services, it also introduces more and more risks in terms of user privacy preservation. The emergence of the Internet of Things (IoT) in this landscape is both encouraging on the one hand new kinds of services and on the other hand introducing new kinds of threats.

Privacy concerns in IoT context depend on the kind of sensor data that are collected and exploited. See [privacy] for a survey about this subject.

If sensor measurements characterize environmental data that are not related to individuals (such as e.g. the temperature at a given geographic location) then privacy preservation concerns are low. Of course operators may claim ownership about collected data, in order to protect their investments for IoT infrastructures. Then a proper security architecture such as the one defined for SmartSantander can efficiently protect infrastructure operators against eavesdropping risks. Authorization means are also available in order to let IoT infrastructure operators grant access only to intended recipients, such as experimenters, end users, or third parties service providers.

Such statements apply to a civilian context where the motivation for privacy is mainly economical (still assuming that sensor data doesn't relate to individuals). If we assume a military context instead, strategic reasons may be substituted to economical reasons.

Now if we consider sensor data that relates indeed to an individual, then the ownership concept is somehow shifted from the IoT infrasturcture operator to the concerned individual. This statement becomes quite clear if we consider e.g. medical application where sensor data may carry health diagnostics. However this principle is not limited to medical applications. The location of individuals for example is also typically considered as sensitive information which belongs to the private sphere.

Various strategies may be proposed to address privacy preservation issues. Of course, several of the strategies below may be combined.

One possible strategy consists in considering that private data is owned by the concerned individual, and giving to this individual a complete control about who he/she will share his/her private data with. This strategy somehow implies end to end security between the individual owning data, and

individuals that they decide to share their private data with. This strategy is well suited for medical applications.

An alternative end to end security may also consist in having a trusted central entity that mediates communications between sensor data producers and sensor data consumers. Such an architecture choice may be induced if e.g. legal interception aspects are considered.

As a third strategy, anonymity treatment may be applied at the source of potentially private sensor data. For example if the identity information is discarded from the sensor source, then the remaining measurement data can be considered anonymous, and it may be used to feed databases for statistical purposes. One may claim that this kind of anonymous data is equivalent to non-individual-related sensor data.

In the context of SmartSantander, privacy preservation concerns should be considered in view of which use cases are addressed. Health applications have not been addressed so far, so that end to end security has not been introduced in the architecture. However an authentication and authorization infrastructure has been defined and implemented in order to let testbed administrators have a complete control about which user is accessing which sensor data. In case of testbed federation, each testbed administrator has still a complete control about who should get access to sensor data that is generated inside in his/her testbed domain.

Cryptographic techniques are in place in order to provide a good trust level against eavesdropping. Therefore the SmartSantander security architecture provides fair guarantees against dissemination of private information.

However, a focus can be set on two specific SmartSantander applications that may be considered as sensitive in terms of user privacy preservation. In this respect, Participatory Sensing and Augmented Reality deserve special attention.

Participatory Sensing:

The Participatory Sensing application should be considered carefully with respect to user privacy, because user's personal SmartPhones are exploited in order to complement classical sensors in terms of environmental measurement production. Such measurements are geo -localized and time-stamped, therefore concerns may rise about the risk of user privacy exposure.

However, this application benefits from anonymity treatment, as exposed earlier. Smartphone sensing information is attached only to a specific device, not to its owner whose identity is absent from sensor reports.

Furthermore, the sensing reports are encrypted in such a way that external observers that may spy message exchanges over the air or over the Internet cannot restore any meaningful information. Only the concerned testbed operators can restore the original reports (that doesn't expose any user identity anyway) and make them available to feed the Participatory Sensing application with properly anonymized data.

Augmented Reality:

The augmented Reality application is another example where specific concerns may be fed with respect to user privacy protection.

One of its objectives is to provide third party developers with feedback about tourist behaviours around points of interest. Therefore this application supposes recording of tourist behaviour.

Once again, the key factor for protecting user privacy in this case is to treat user data in a statistical and anonymous way. Visitor identity should not be part of collected data, though some tourist information such as visitor nationalities can be collected for statistical purposes.

Of course, it should not be possible to spy message exchanges over the air or over the Internet, which can be obtained through the use of state of the art encryption. Furthermore, only statistical and anonymised data should be made available to the subscribed application consumers.

In summary, both exposed use cases (Participatory Sensing and Augmented Reality) present risks in terms of user privacy preservation. In both cases, it is possible to mitigate such risks by combining counter acting techniques, as detailed below.

- User anonymisation is achieved by ensuring that any reports or messages originated from smartphones don't contain any field related to a user's identity.
- Message encryption ensures that only testbed administrators may exploit reports or messages originated from smartphone.
- User anonymization is further achieved by providing only statistical data and anonymised data in terms of application databases.

3.6. Initial Requirements

From the above use cases, the following set of requirements have been identified:

Module	ID	Title	Text
AAA	FR180	Certification Authority	SFA authentication is based on X.509 certificates, so a root certificate must exist to authenticate user certificates
	FR181	User DB synchronization	User authentication in SFA is managed in the SFA Registry, so SmartSantander user database should be synchronized with the SFA one
	FR182	User authorization	Different Access policies could be defined for users accessing testbed through SFA or through SmartSantander traditional interfaces
	FR183	User data base for authorization	Each testbed site which intends to restrict access to certain functionality and devices of their testbed have to set up a user data base

	FR184	User Management	To ease the management of users for experimenters and the users themselves, a GUI client should allow for creating and updating accounts for multiple testbed sites simultaneously
Reservation	FR185	SFA slice management	In SFA, resources are organized in slices, so a component must be defined to translate slice management operations into SmartSantander reservations (for native experimentation on top of infrastructure) and subscriptions (for service experimentation).
	FR186	Federated Resource Reservation	Experimenters have to add resources from different sites of the federated SmartSantander testbed to a single reservation to create a federated experiment.
Resource Management	FR187	SFA testbed exposure	In order to expose SmartSantander as a SFA compliant testbed, a component understanding SFA protocol must be created. SFAWrap can be an option as a first approach and, in this case, a SmartSantander SFAWrap driver has to be developed.
	FR188	RSpec Resource Description	A RSpec version covering SmartSantander specific resource information has to be defined.
	FR189	SFA Resource discovery	SFA understands RSpecs resource description, so a component to translate Resource Directory (RD) responses into RSpec formatted data must be created.
	FR190	Unified Meta Data Access	In Order to easily access meta data from federated but otherwise independent testbeds, the access of a testbed's meta data should be unified
Experiment Configuration	FR191	OMF Aggregate Manager	SmartSantander architecture needs to expose an OMFv6 Aggregate Manager in order to make itself OMF compliant

	FR192	OMF Resource Controller	Each of the SmartSantander resources must be controlled through and OMF Resource Controller (RC), which will be able to interact with an OMF Experiment Controller. There is a need to implement at least two different kinds of RCs: one for native experimentation on top of nodes, which will interact with Testbed Runtime; and another one for service experimentation, which will interact with SmartSantander DCA Platform either directly or through a gateway to handle AAA in a simplified way.
	FR193	FRCP XMPP Server	For the usage of Federation Resourcer Control Protocol (FRCP) in OMF the testbed owner should provide an XMPP server. This XMPP server should be accessible from a private network, in order to connect with the testbed resources and resource controllers, but also it has to be connected to a public network, to allow the experiment controller to talk to the resources through it.
	FR194	OML Experiment Monitoring	The results coming from experiments have to be sent to OMF Experiment Controllers by using OML protocol, so at least one OML Measurement Point will have to be deployed for each experiment running on top of SmartSantander
	FR196	Federated Experiment Control	When the researcher uses the SmartSantander experimental facility to perform an experiment on devices which span over multiple testbed sites, the system must provide a mechanism to control (e.g. start, stop) these experiments.

Infrastructure Monitoring	FR197	OML based Monitoring	OML provides an simplified way of sending monitoring information to upper layers. An OML monitoring server could be used to store a testbed monitoring information. Most of the monitoring in SmartSantander is based in the information in service frames and its behaviour, so an OML Measurement Point could be defined to analyze them.
Experimentation Software Updates through Scheduled /Opportunistic Contacts using DTN protocols	FR198	DTN Deployment Plan	Based on the parameters such as experiment start time, size of software image, scheduled contact of sensor node with DTN routers, duration of contact, data rate available during connection, the Experiment Scheduler subsystem shall compute a deployment plan. The deployment plan specifies the set of DTN routers where contact is predicted to be made, the expected duration of the contact, the subset of mobile nodes that need to be reprogrammed and the set of software image fragments to upload on each DTN router.
	FR199	DTN Deployment Plan Scheduling	The deployment plan shall be scheduled for execution on the Portal Server with the Experiment Scheduler. The scheduled time of execution takes into account the experiment start time and software fragment transfer times and retransmission delays.
	FR200	DTN Multicast Tree Dynamic Construction	The set of DTN routers in the deployment plan are requested by the Experiment Scheduler to join a multicast group for the software image fragment dissemination. The number of multicast trees constructed will depend on the number of different fragment-sets that need to be transmitted and the number of intended recipient DTN routers.
	FR201	Permanent Multicast Trees	For DTN routers where scheduled IoT-node-contact is expected to be regular and long-lasting (enough for whole image transfer, e.g. Bus Depot), a permanent multicast tree shall be constructed.
	FR202 a	DTN Deployment Plan Distribution	The DTN deployment plan shall be distributed with the targeted set of DTN routers.

	FR202b	Deployment Plan Execution at Portal Server	At the scheduled time, the DTN deployment plan shall be executed by the Experiment Scheduler to initiate the transfer of sets of software image fragments to their respective group of DTN routers. The Experiment Scheduler shall use the existing iWSN interface to upload the software fragments on the IoT nodes.
	FR203	Authentication	Image fragment transfers between the Portal Server and Gateway Devices must be reliable and authenticated via an authentication key
	FR204	Persistent Storage at DTN router	When a software image fragment is received by each DTN router, it shall be checked for transmission errors and for conformance to DTN Deployment Plan. If corrupted fragments are detected, the Experiment Scheduler will be requested to retransmit them. If it is a member of the set of expected fragments for the host DTN router, it is saved to persistent storage. Else, it is discarded.
	FR205	Fault tolerance and dependability	Fragment sets shall be distributed to redundant DTN routers for fault tolerance. But only one DTN router shall proceed with fragment transfer to the mobile IoT node.
	FR206a	Detection of online mobile IoT nodes	Each DTN router shall subscribe for node-arrival event notifications from the NodeManager component on the Gateway for when the mobile nodes are in close proximity. The event notification shall specify the type of network connection (e.g. 802.11b/g or 802.15.4).
	FR206b	Detection of online mobile IoT nodes	When mobile nodes become available via two or more network connections, the higher-bandwidth network connection is selected by the DTN router for image fragment transfers
	FR207	Fragment MOTAP Upload	When a DTN router is notified of the proximity of an IoT node/a set of IoT nodes, it should verify that the duration of the node's scheduled contact is long enough for the transfer of the software image fragment(s). It shall then unicast/multicast the image fragment(s) to it/them using the MOTAP functionality on the Gateway.

	FR208	Software Image reconstitution	Each target IoT node shall acknowledge the receipt of software image fragments and commit them in persistent storage. If all the image fragments have been received, the IoT node confirms the reception of the whole program and waits for the command to load the binary image in the flash memory.
	FR 209	Deployment Execution Report	After the transfer of image fragments has been completed, the DTN router shall send a Deployment Execution Report to the Experiment Scheduler, which describes the outcome of the image fragment uploads.
	FR210	Remediation Action	If the software image fragment transfer failed for an IoT node, the Experiment Scheduler may select another IoT node with similar characteristics and instruct the DTN router to initiate the software image fragment upload.
Experimentation support on gateway tier devices	FR211	VTD support	The testbed should allow the deployment of experimentation code on GW devices in the form of virtual testbed devices. VTD provide the same interface towards the testbed as physical device drivers.
	FR212	VTD lifecycle management	The testbed should allow the creation and deletion of previously deployed virtual testbed device on GW devices.
	FR213	VTD resource constraint specification	The VTD Scheduler component on the GW device shall provide an interface for GW owners to specify quantitative resource constraints on the share allocated for each VTD for each node resource.
	FR214	VTD Priority Classes	Resource allocation for VTDs shall be performed based on priority classes. Higher priority VTDs shall receive larger resource shares than low-priority VTDs.
	FR215	VTD Scheduling	The VTD CPU Scheduler shall use scheduling mechanisms and policies to schedule the VTD for execution on the CPU as per their resource allocation.
	FR216	VTD Preemption	Low priority VTDs shall be preempted from resource utilisation by high priority VTDs or processes.

	FR217	VTD Processes	The VTD Scheduler shall support the execution of experimentation code written in different languages.
IoT Nodes/Smartphones	FR218	Access Configuration to Smartphone resources	The smartphone end-user should configure various access rights for his smartphone resources: sensors (Gps, accelerometer, microphone etc), network interfaces (wifi, Bluetooth, 3g), smartphone configuration settings (language, country etc.)
	FR219	Experiment Execution Parameters	The smartphone end-user should configure various execution parameters of utilization of his smartphone by an experiment like maximum memory used, maximum battery used. Furthermore, should be possible for smartphone end-user to turn on/off experiment execution.
	FR220	Recorded Experimentation Datasets Upload	Experimentation smartphone applications should upload the recorded datasets to the SmartSantander server.
Experiment Configuration and Execution	FR221	Configuration of future experiment	A testbed user should be able to configure an experiment for a resource reservation time that lies in the future and that may not be explicitly known to the experimenter in advance.
	FR222	Selection of execution time	In cases where the start time of an experiment is not crucial, the testbed may decide an optimal exectution time for an experiment that best matches the reource requirements expressed in the testbed configuration and other local policies
	FR223	Automatic execution of experiments	The testbed must be able to execute experiments on behalf of based on a configured or determined execution time, if a valid experiment configuration exists, without the need of a user or client side tool to be involved in the process.
IoT Nodes/Smartphones	NFR180	Privacy of Smartphone end-users	Experiment execution should maintain the privacy/anonymity of smartphone end-users.

	NFR18 1	Integrity of SmartPhone	Experiment execution should maintain the proper operation of smartphone: no battery drain, no memory drain, no erase of user files
	NFR18 2	Recorded Experimentation Datasets	Experimentation smartphone applications should record smartphone resources (gps, batter etc.) and end-user feedback.

Table 1 List of functional and non-functional requirements

4. THIRD CYCLE SMARTSANTANDER ARCHITECTURE SPECIFICATION

This section provides an overview of the third cycle specification of the SmartSantander architecture. The third cycle architecture represents an evolution of the second cycle architecture specification presented in [D1.2]. It addresses the use cases and subsequently derived requirements that have been presented in Section 3 of this document.

The section is structured in two main parts. The first part, Section 4.1, provides a high level overview of the third cycle architecture and introduces the extensions and modifications that have been proposed in the context of the overall architecture. Section 4.2 then presents each of the proposed modifications in more detail. For reasons of clarity, the discussions are structured analogous to the use case specifications in the previous section. This allows the reader to easier understand how the use cases and resulting requirements have been addressed.

4.1. Overview of Third Cycle Architecture

The new requirements addressed by the third cycle design and specification process resulted in several new system functions and upgrades to existing ones that were present in the previous architecture specification. An overview of the resulting architecture can be found in Figure 2, highlighting newly introduced or updated system functions in colour. In the following each of these additions and modifications are introduced by briefly explaining their nature and purpose. For a detailed description of the new system functionality, the reader is referred to section 4.2.

A major architectural change represents the introduction of a new subsystem for *Federation* support, which is highlighted in yellow in the top left part of the architecture diagram. It addresses the need for different SmartSantander testbed sites to be federated among each by the introduction of the *Federator*, a feature that has been adopted from the WiseBed [WiseBed] base. This feature is also referred to in this document as *internal* federation. The Federator has been designed in such a way that it does not introduce a new interface, but instead provides convenient access to different SmartSantander testbeds via the existing APIs. Some changes to the underlying data model of resource directory have to be made, in the form of additional metadata for describing resources in the different testbeds. A second requirement addressed by the Federation support is the *external* Federation with other testbeds in the FIRE community. Through an analysis of existing tested frameworks and discussions with the FIRE community several existing standard APIs have been identified that are widely supported. These include the Slice Federation Architecture (SFA) for managing discovery, access and reservation of experimentation resources and Orbit Management Framework(OMF) for experimentation control. Respective APIs are exported to support experimentation clients that comply to these specifications through the introduced SFAWrapper and OMFv6 system functions. The system functions ensure the translation of the respective API invocations to the respective testbed internal system functions.

The third cycle architecture introduces also new system functions to existing subsystems for the support for experiments that rely on Smartphones as experimentation resource.

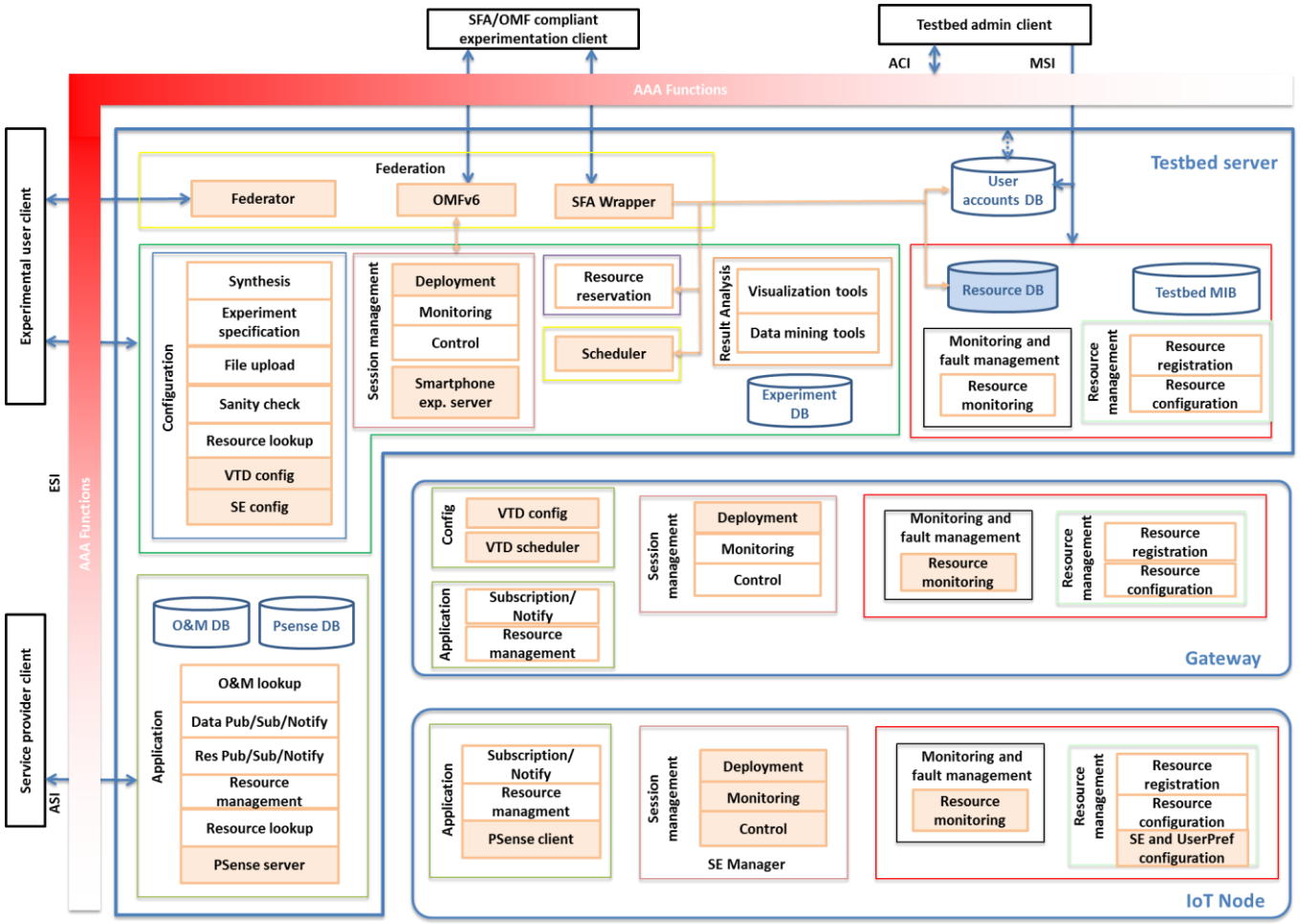

Figure 2: Overview of third cycle architecture.Additons of and mofications to system functions are highlighed in colour.

This includes the addition of the *Smartphone Experimentation Configuration* function (SE Config) to the configuration subsystem and the *Smartphone Experimentation Server* as part of the session management subsystem of the server tier. Corresponding functions on the Smartphone side are realised by updates to the *Deployment*, *Monitoring* and *Control* Functions of the session management subsystem - also referred to as *SE Manager* functions and the addition of *SE and User Preference configuration function* as part of the resource management subsystem. This allows both the effective configuration of experiments for Smartphones that respect user preferences on resource use and privacy. Furthermore they allow efficient scheduling of experiments, provisioning of experimentation code and experimental data collection.

In order to support experimentation with scheduled or opportunistic contacts of experimentation resources with the testbed infrastructure, delay tolerant network (DTN) capabilities have been added to the architecture. As a result provisioning of experimentation code or the collection of experimentation data does not have to rely any more on permanent connections to the testbed backbone infrastructure. This is achieved by an upgrade of several existing components of the architecture, namely the *Deployment* function of the Session Management subsystem in all three tiers of the architecture, the *Scheduler* on the server tier as well as *Resource monitoring* in the resource management subsystem of the GW and IoT node tier.

A further new feature that the architecture introduces is the support for the execution of experimentation on GW tier devices. For this the *Virtual Testbed Device* (VTD) support that has been initial introduced in the 2nd cycle architecture specification is utilised and further enhanced by the addition of the *VTD scheduler* on the configuration subsystem on the GW tier. The latter manages the priority of experimentation code to be run on GW tier devices and that they do not impact the performance of the primary system functions.

The final set of upgrade to existing system functions addresses the challenges of increased end user privacy in experimentation. Both *PSense Client* and *PSense Server* components of the application subsystem are upgraded with security functions such as encryption.

4.2. Revised specification of system functions

The previous section provided only an overview of the different architectural additions and upgrades in order to support the third cycle requirements and use cases. This section describes in more detail how each of the requirements are addressed by the introduction of new features and how the proposed architectural extensions and modifications contribute towards it. The structure of the section and follows the order of the different use cases introduced in section 3.

4.2.1. Federation of Smartsantander tesbed sites

So far, SmartSantander experimenters perform experiments on devices located in a single SmartSantander testbed. However, since the testbed sites are similar in terms of their architecture and capabilities, an experimenter should be able to use devices of multiple testbeds in a single experiment. This will allow users to run large scale experiments and to mix devices and device types from the different testbed sites. This kind of federation is called *Intra SmartSantander Federation*.

Generally such federation, and the API interactions described next, was available in basic form in the Testbed Runtime (TR) of the WISEBED project, which SmartSantander inherited. However, due to the various enhancements of TR to increase the performance, to allow for the integration of new components and to match requirements introduced by SmartSantander, the components and API interactions had to be updated and the access to the testbeds' meta-data has been unified.

Intra SmartSantander Federation architecture

As shown in Figure 3:, the client application which conducts a federated experiment accesses the testbed via the Experimentation Support Interface (ESI) as usual. But instead of connecting to a specific testbed, it connects to the Federator component which is used as a proxy between the client and the testbeds which resources are to be used. As can be seen in Figure 3:, all components present in the (WISEBED compatible) testbeds are present in the Federator as well.

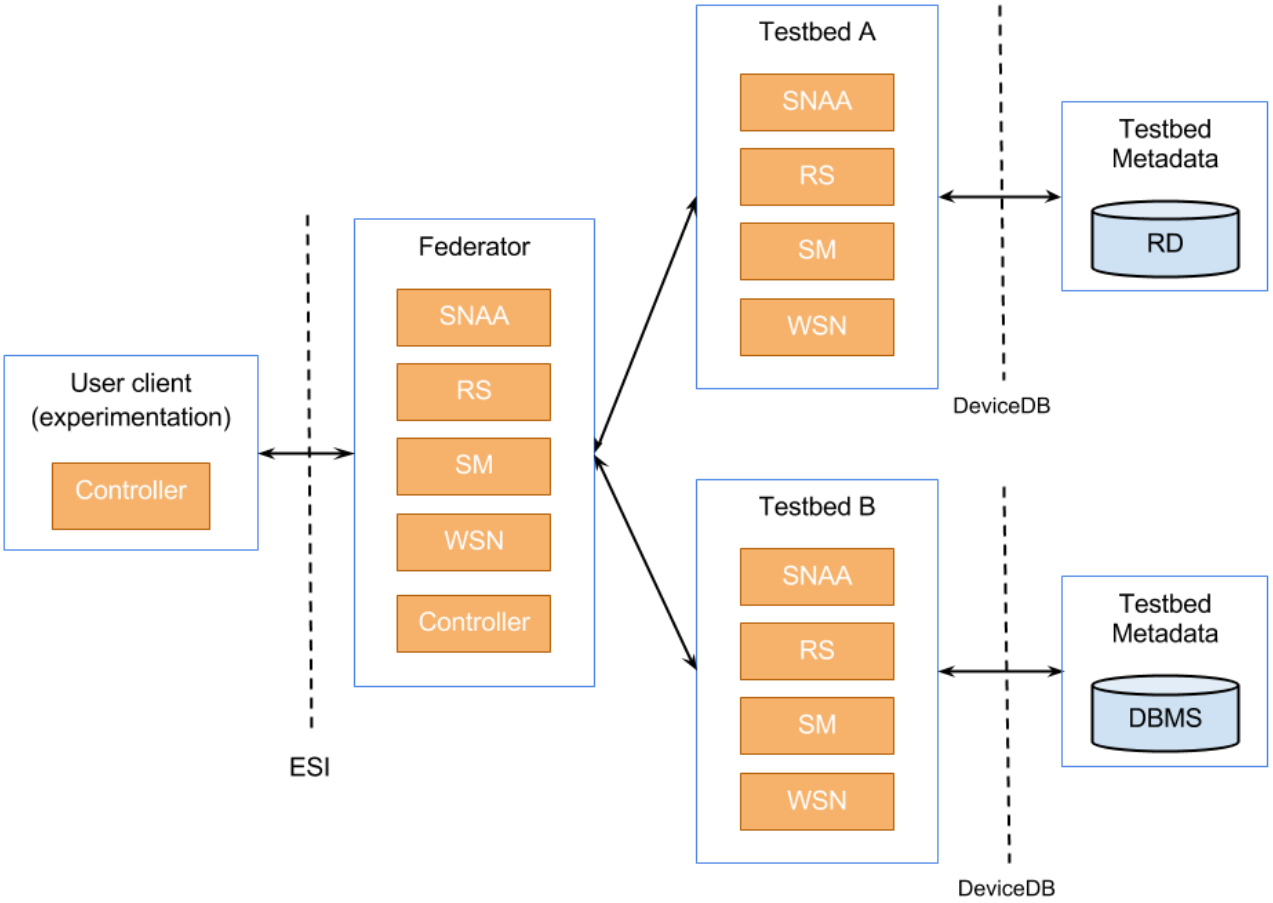

Figure 3: Intra SmartSantander Federation architecture

These components safeguard the access to the testbed (SNAA), manage reservations of testbed resources (RS), manage the experimentation sessions (SM) and provide access to the testbed in

means of flashing, resetting, etc. Thus, for the clients there is no difference in accessing a single testbed or multiple testbeds at once via the Federator component.

Since the client features a Controller component, which is responsible for the reception of asynchronous messages, the Controller component is present in the federator as well. That is, for the testbeds the Federator acts as an experimentation client.

Testbed Meta Data for the Federation

Since we federate independent testbed sites, their meta-data may be persisted in different ways. This meta-data includes the WiseML-based topology description of a testbed as well as configuration data of individual nodes (e.g. serial port parameters). Several components of the testbed infrastructure (Testbed Runtime) rely on this information to work correctly. E.g., the gateway hosts of a testbed have to find out serial port parameters, in order to connect to a device and the portal host has to produce a topology self-description in WiseML.

As depicted in Figure 3:, testbed A manages all meta data in the Resource Directory (RD) which was inherited from SENSEI and enhanced by the SmartSantander project. For tesbed B, the meta-data is managed in a simple Database Management System (DBMS) which is not further specified.

In order to be able to treat both meta-data sources identically the *DeviceDB* interface was created for the next release of Testbed Runtime. It abstracts from the way where and how meta-data is stored and allows a uniform retrieval (FR190). Independently, the actual description of resources from different testbeds should be the unified as well (FR083).

4.2.2. Federation of SmartSantander with external testbeds

An increasing number of the FIRE facilities deployed in the latest years are converging into OMF/OML and SFA as its public interfaces to expose themselves to the outside world. For this reason, the best approach to enhance SmartSantander's federation capabilities with external testbeds is to also provide those OMF/OML and SFA interfaces to cover the whole experimentation life cycle.

OMF Overview

OMF (cOntrol and Management Framework) is a generic framework for controlling and managing networking testbeds developed in the last couple of years mainly by NICTA. The motivation behind OMF is to provide the experimenters with a way of defining their experiments in generic terms, by using a domain specific language, while hiding testbed specific hardware details. In this sense, OMF provides a set of tools to describe and instrument an experiment, execute it and collect its results. Nevertheless, OMF does not currently deal with resource discovery and reservation, so these are stages that have to be done prior to any OMF interaction with a testbed.

The OMF framework is also a possible way toward testbed federation as it assumes that several testbeds may be jointly involved in the experiments.

The key components of OMF are:

- A description language, named OEDL, which allows the user to create its Experiment Description (ED). This ED contains a description of the resources needed for an experiment, including their required configuration; a description of the applications to use in an experiment and the measurements to collect from them; and a state machine describing the different tasks to perform during an experiment, and the time or event that trigger them.

  In the context of SmartSantander, OEDL can be seen as a language similar to the Testbed Runtime API scripting language that was introduced in the WISEBED project. However, the abstraction that OMF provides means that OEDL is a more generic description language than WiseML, which was designed to cover WSN requirements. From now on, the SmartSantander experiment description will be named Testbed Runtime Experiment Description (TRED).

- The Experiment Controllers (EC), which is the entity that is responsible to deploy, control, and instrument an experiment on behalf of the user. The EC takes an ED as its input and processes it. The EC cooperates with the Aggregate Manager(s) of the testbed(s) involved in the experiment, to configure the required experimental resources according to the experiment description. Once these resources are configured, the EC communicates with each corresponding Resource Controllers, and sends them commands (according to the ED), which will effectively realize the experiment.

  In the context of SmartSantander this component works just as the Testbed Runtime scripting client, which instead of an ED gets a TRED.

- The Aggregate Manager (AM) is the OMF entry point in any testbed. It manages all the resources that are provided by a given testbed. More specifically, the AM consists of a collection of services that allow external parties to request management actions on a set of resources, or query their status. It also includes an OML experimentation server that can be used to store any experiment results in the testbed domain, for a future query of an experimenter.

  The Resource Controller (RC) is an entity that listens for commands issued by an EC, executes them, and reports back any information resulting from the command executions. Those RCs are entities that reside on each of the resources available within a testbed, but it is not a mandatory requirement.

- The Measurement Points (MP) are specific points inside any application where a set of measured metrics can be collected by the OMF/OML instrumentation tools. Those components will be covered in the OML section, although MPs were initially defined under OMF.

OMF has had multiple versions through the years, but the latest one, OMFv6, includes a complete redefinition of the communication protocol between the different entities that conforms an OMF enabled testbed. This protocol, known as Federated Resource Control Protocol (FRCP), uses an

Extensible Messaging and Presence Protocol (XMPP) server by using a PUB-SUB mechanism with different topics to exchange information between all EC's, AM's and RC's. With FRCP new components can be created and/or reconfigured dynamically. On the other hand, OMFv6 provides Resource Proxies. They are software entities acting on the behalf of resources, thus they are the software that process messages published to resources, and generate any replies. A resource proxy may have its instance running on the resource itself or on any other platform which has access to the resource.

OML Overview

OML (OMF Measurement Library) was developed together with OMF, in order to take care of reporting experiment results to a common database. Though it started as an OMF component, OML is now autonomous and can be used with or without OMF.

OML enables real-time collection of data from the various applications being run during an experiment. It follows a client/server approach, and consists of two main components:

- The OML client library provides a C API for applications to collect measurements that they produce by allowing experimenters the Measurement Points (MPs) definition inside their application source code. The measurement library includes a dynamically configurable filtering mechanism that can perform some processing on each measurement stream before it is forwarded to the OML Server. OML MPs are quite flexible, and can be configured to send information to more than one OML Server.

- The OML server component is responsible for collecting and storing measurements inside a database. Currently, SQLite3 and PostgreSQL are supported as database backend.

In a traditional OML environment, once an experiment is complete, the stored measurements or results can be accessed for further analysis. An experimenter can then manipulate these data using SQL queries or export them in some other format for further processing and analysis. However, although OML servers usually are database components, an external application which understands OML measurements can work as an OML Server as well. This way, it would be possible to receive OML measurements in an external Experiment Controller.

SFA Overview

SFA (Slice Federation Architecture) has emerged as the de facto standard for enabling federated slice-based network substrates to interoperate. It forms the control plane for browsing, allocating and reserving resources offered by a federation of heterogeneous testbeds. In an SFA context, testbed-specific information is captured in a resource model, called a resource specification (RSpec), which is an XML transported by the SFA layer. However, SFA itself does not cover such aspects as resource model, policies, experiment configuration, experiment execution or data collection.

SFA also provides a common authentication layer based in X.509 certificates. However, authorization management, usually based on local policies, relies on the underlying testbeds. This means that an entity can be authenticated even if later it turns out that it is not allowed to perform actions or access resources of a particular testbed.

SFA designates a set of four main object types that represent the different entities involved in the testbed federation:

- *Authorities*. These represent testbeds, parts of testbeds to which trust or rights may be delegated, and/or communities of users.
- *Resources*. These consist of nodes, links, or any other experimental resource provided by the testbeds, and exposed to the users.
- *Users*. These are experimenters wanting access to resources.
- *Slices*. A slice is the basic unit of interaction between users and resources. Slices can be seen as a set of virtualized resources connected together to provide a single virtual testbed for an experimenter. One can think of a slice as corresponding to an experiment, encompassing all the users and resources associated with that experiment.

In order to become an SFA compliant testbed, its resources must be described in an RSpec that the testbed's aggregate manager can both send and understand. For this purpose, a generic reference implementation called SFA Wrapper is provided. This component will interact with the local testbed user database, its resource database and the scheduler used for reservations.

SmartSantander proposed adaptations to OMF-OML-SFA

An initial approach to transform SmartSantander testbed into and OMF/OML and SFA compliant testbed is depicted in the Figure 4. It is important to note that, as illustrated in the figure that this approach intends to be a complement to the current SmartSantander platform as it does not imply that current federation mechanisms are overridden or. Instead it enables federation with other OMF-OML-SFA enabled testbeds.

Although the proposed architecture is based on *SFAWrapper*, there is an alternative that will be also analysed known as AMSoil. The SFA component will then be used to provide authentication, resource discovery and reservation. SmartSantander Driver component will interact with the next components:

- The Resource Directory (RD), named in the functional architecture as resource DB in order to get information of the SmartSantander resources.
- The user accounts DB to authenticate and authorize users accessing the testbed.
- The resource reservation and the scheduler components for creating the needed slivers inside SmartSantander and contribute to slices creation.

The SmartSantander SFA component covers the requirements FR180 to FR181 and FR185 to FR189.

OMFv6 will be used in the description and provisioning of the experiments. Besides, OML will be used in this context for measurement collection both in Native and Service Experimentation. FRCP will be used to allow the creation of real time per-experiment RCs, as well as to remove them as soon as all resource reservations have ended. By deploying an OMF infrastructure inside SmartSantander

requirements FR191 to FR193 and FR196 are covereded. On the other hand, requirement FR194 is covered by the OML dynamic measurement points that will be created on request in the proposed architecture.

The OMF/OML interactions with the SmartSantander infrastructure will be different depending on the experimentation layer.

- In Native Experimentation layer, the workflow will involve almost all the session management components.
- In Service Provision layer, the data notification system will be the main component that will interact with the resource controllers, in order to gather all the data from the data repository.

OML will also be used in a static way to provide Monitoring capabilities to SmartSantander. This way, requirement FR197 is covered as well.

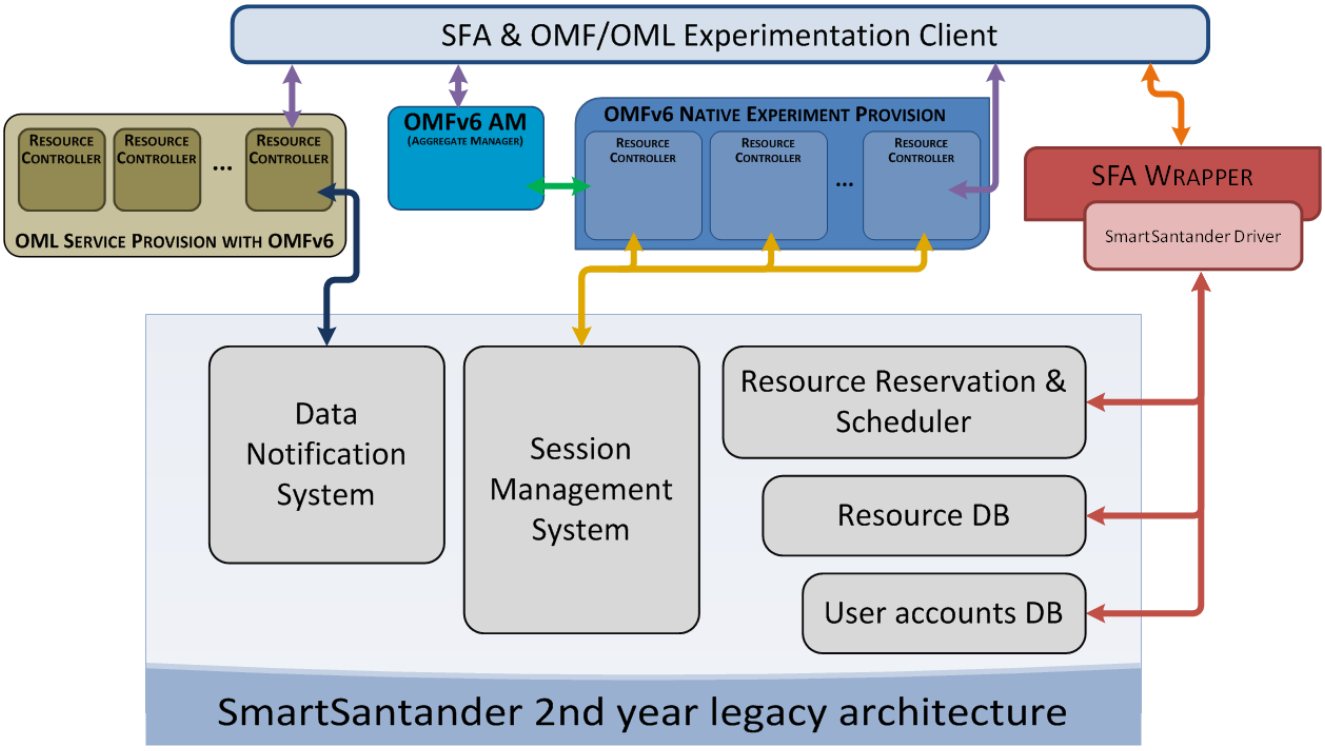

Figure 4: Proposed architecture for SmartSantander external federation

4.2.3. DTN support

In the original global architecture for the SmartSantander platform, the process for image-based software reconfiguration on sensor nodes (including mobile nodes) required the following functionality,

to be provided by the *Deployment* blocks at the Portal Server, Gateway and IoT Node tiers (SeeFigure 5: The Bundle Layer DTN protocol).

1. The Portal Server and Gateway Nodes are connected via a network overlay (or messaging middleware) for the routing and transfer of messages/commands between all IoT nodes (portal server, gateways and sensor/actuator nodes).
2. When a set of sensor nodes are to be updated with a new binary software image, the image is unicasted to the respective Gateway devices to which the sensor nodes (specified in the set) are attached/associated.
3. The Gateway device then initiates node reprogramming by using a multicast-based Over-The-Air-Programming protocol to transfer image fragments to the respective nodes.
4. Sensor node devices collect image fragments, send back acknowledgements upon correct reception and store them in external flash memory. Once a whole binary image is successfully built, the image is loaded in program memory and the node reset to run the new software configuration.

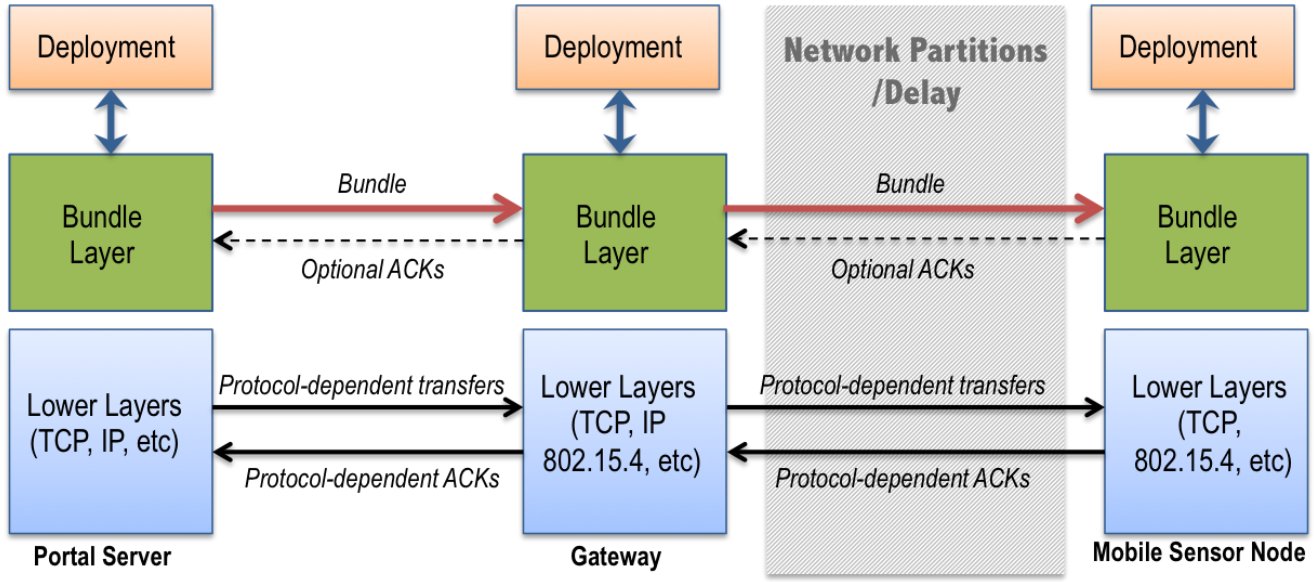

Figure 5: The Bundle Layer DTN protocol

The procedure for reprogramming mobile sensor nodes followed a similar approach in that a persistent connection to each mobile via a WAN/GPRS link is used for the dissemination of binary WSN software images.

In our DTN-enabled node software reconfiguration process, we use the functionality provided by DTN protocols to isolate delay and overcome network partitions, caused by mobile sensor nodes transiting from one gateway coverage to another. However, although our architectural amendments are intended for general DTN support for IoT node reprogramming, it is expected due to project time/resource constraints that design and implementation effort will concentrate on realising the ”IoT Node Reprogramming at the Bus Depot” Use Case. We envisage our delay-isolation solution to be

fully integrated with our existing solution for image-based node software reconfiguration. Therefore, we plan to augment the functionality of the *Deployment* function at each tier to provide support for the store-and-forward functionality required by DTN-participating devices and the end-to-end reliability that sensor node reprogramming entails. To the existing components, given that interactions for reprogramming occur in an asynchronous manner, delay isolation should be transparent.

In particular, we add an additional layer to the protocol stack used for node software reconfiguration, called the **Bundle Layer**, to perform delay isolation and ensure end-to-end reliability across image transfers. As illustrated by Figure 5, the Bundle layer (at the Portal Server) intercepts image transmissions from the application layer, breaks the bundle (software image) into fragments and multicasts then to the corresponding Bundle Layer at the receiving counterpart devices (e.g. Gateway devices).

The Bundle Layer performs delay isolation via *transport termination* i.e. DTN nodes terminate transport protocols at the Bundle Layer. The Bundle Layer therefore acts as a surrogate for end-to-end sources and destinations. The result is that the conversational lower-layer protocols of low-delay regions (e.g. network containing Portal Server and Gateway devices) are isolated at the bundle layer from long delays in other regions of the end-to-end path. The Bundle Layer alone supports end-to-end messaging. Bundles are typically delivered atomically from one node to the next, independent of other bundles, except for optional responses. However, a Bundler Layer may break a single bundle into a multiple fragments if, for instance the fragments must be disseminated to different points of contact.

The delay and intermittent connectivity between Gateway nodes and mobile sensor nodes caused by temporary contacts between them is shown by the grey area in Figure 5. In our usage scenario, the Bundle Layer performs the following functions:

1. Fragmentation of software images into bundles to fit in the underlying protocol's PDU (protocol data unit)
2. Selection of Gateway devices (called DTN routers) for store-forwarding ,where mobile sensor nodes are predicted to make contact
3. Transfer of software bundles to the selected Gateway devices and caching into persistent storage. Bundle transmissions may be acknowledged
4. Upon contact with a target mobile sensor node, it initiates the transfer of the software image fragments onto the node.

DTN Roles

We define the following DTN roles for devices in the SmartSantander infrastructure:

- **Host** – sends or receives bundles but does not forward them. A host can be a source or destination of a bundle transfer.
- **DTN Router** – Forwards bundles *within* a single DTN region (the internetwork at the point of contact) and may optionally be a host. The Bundle Layer of DTN routers that operate over long-delay links require persistent storage to queue bundles until outbound links are available. DTN routers may optionally support custody transfers i.e. the transfer of the responsibility to do retransmissions and send acknowledgements to the bundle source.

- **DTN Gateway** – Forwards bundles between two or more DTN regions and may optionally be a host. The Bundle Layer of DTN gateways must have persistent storage and support custody transfers. DTN gateways provide time-delayed conversations between the lower-layer protocols of the regions they span. In the SmartSantander project context, a DTN gateway can, for instance, be a mobile sensor node that acts as data mule between geographically distributed islands of sensor sub-networks. Or it can be a multi-homed mobile sensor node that makes opportunistic contacts with SmartSantander Gateway devices to upload temporally-aggregated sensor measurements or to receive sensor node software updates.

Custody Transfers for End-to-end Reliability

DTNs support node-to-node retransmission of lost or corrupt data at both the transport and the bundle layer. However, because of the absence of a single transport-layer protocol that operates end-to-end across the DTN, end-to-end reliability can only be implemented at the bundle layer.

The bundle layer supports end-to-end reliability by the means of custody-transfers. Such transfers are arranged between the bundle layers of successive nodes in the DTN, at the initial request of the source application. As a bundle custodian, a device (having accepted a custody transfer request) must store the bundle until (1) another node accepts custody or (2) the expiration of the bundle time-to-live. The bundle custodian is responsible for retransmissions to the next hop in the event bundle fragments are corrupted or lost. The bundle custodian may optionally send back an acknowledgement to report the status of an initiated transfer to another DTN gateway/host.

In the afore-mentioned DTN use-case in Section 3.2, the ‘deployment plan’ is an example of the content delivered as part of the custody transfer.

Architectural Extensions for DTN Support

To provide support for DTN, the functionality of the Deployment blocks located at the Portal Server, Gateway and IoT Node tiers must be extended. Figure 6 shows the functional blocks which will be extended in the reference architecture.

At the **Portal Server tier**, the following functions are added to implement the Bundle Layer thereby fulfilling the requirements listed in Section 3.6 for DTN support:

- Interception of software image transmission from Experiment Controller or Experiment Scheduler
- Selection of target DTN routers, based on predicted IoT-node contact, experiment start time and software image size (Requirements: FR198, FR199)
- Dynamic construction of a multicast tree containing all DTN routers with the Portal Server at the root (Requirements: FR200, FR201)
- Fragmentation of IoT node binary software image into bundles of smaller fragments (Requirements: FR202b)
- Dissemination of bundle fragments to DTN routers based on predicted contact durations (Requirements: FR202b).
- Sending of custody transfer request and processing of custody transfer responses (Requirements: FR202a)

At the Gateway tier, the following functions are added to fulfil the responsibilities the Bundle Layer:

- Reception and processing of multicast join request. (Requirements: FR200, FR201)
- Reception and processing of custody transfer request (Requirements: FR202a)
- Subscription for target IoT-node arrival and contact event notification (Requirements: FR206a, FR206b)
- Bundle/bundle-fragments reception and storage (Requirements: FR203)
- Transfer of bundle/fragments to DTN gateways as per deployment plan (Requirements: FR204)
- Retransmission of un-ACKed fragments
- Sending an ACK containing deployment plan execution results back to source

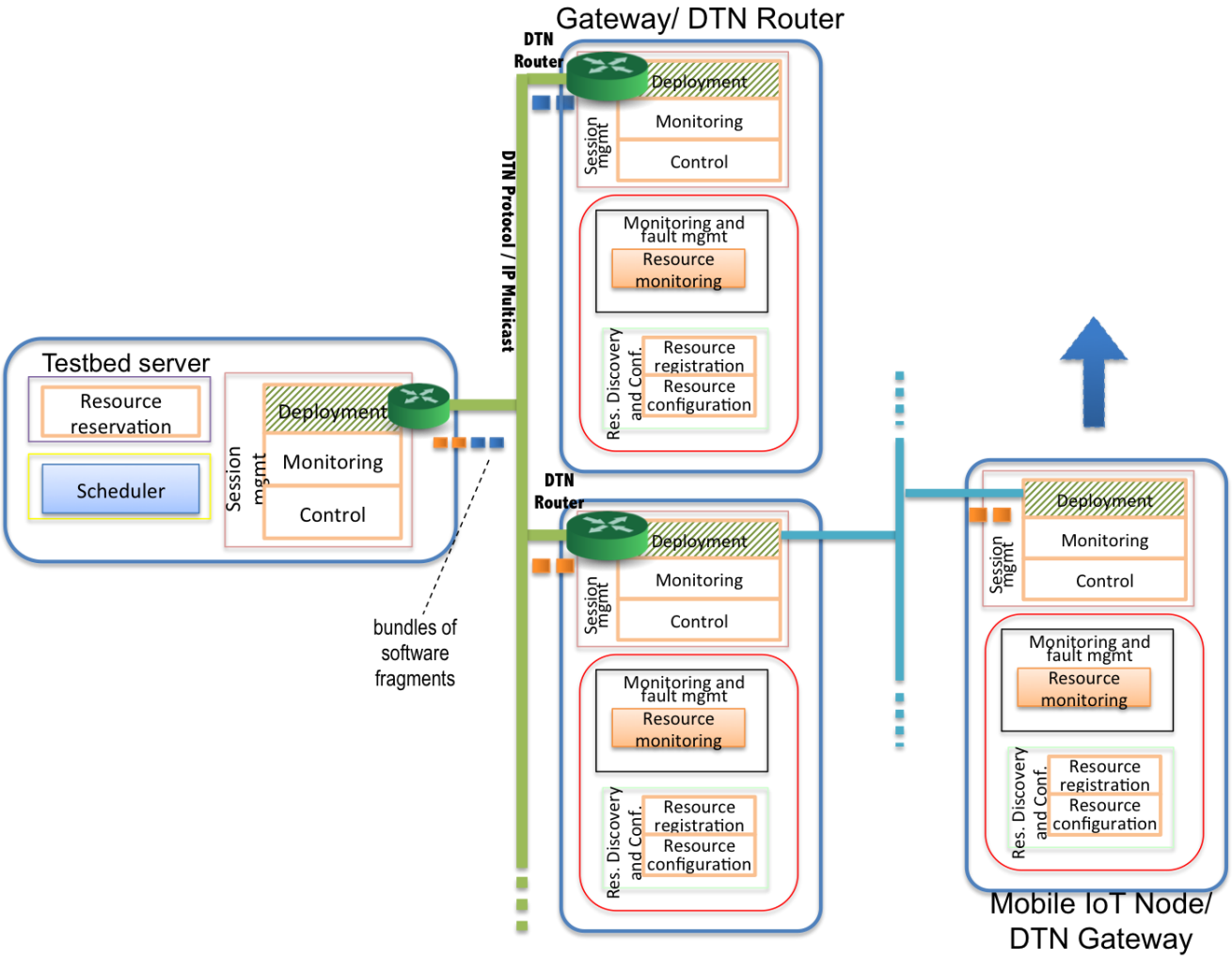

Figure 6: Architectural extensions for DTN-based IoT node software reprogramming[2].

As we intend to utilise the current over-the-air programming solution (M)OTAP deployed at the Gateway devices and IoT nodes, no extensions are required to the existing *Deployment* functional block on the IoT node tier. The current (M)OTAP feature already supports the delivery of software image fragments, the binary image reconstitution and loading in the IoT node's program memory.

[2] Note: Although this diagram represents the general case, design and implementation effort will focus on realising the "IoT Node Software Updates at Bus Depot via DTN protocols" use case.

4.2.4. Experimentation on gateway tier devices

The second cycle architecture introduced the virtual testbed device (VTD) support as a new system function of the experimentation support subsystem. It allows the testbed users to create a set VTDs as part of an experiment configuration. These VTDs will be available on selected Gateway (GW) nodes for the experiment and will allow the execution of experimentation code on GW tier devices.

Figure 7 presents the respective function components that have been previously identified. They cover the requirements FR211 - FR212 through an adequate management of VTDs and their configuration with experimentation code and other behavioural properties through APIs provided by the experimentation support interface. By invoking VTD related functions on the ESI interface, an experimenter can:

- Select the GWs that require the execution of experimentation code
- Configure one or more VTD devices on these GWs which will execute the desired experimentation code, including the provision of the code and VTD properties such as priority level, resource requirements etc.
- Instantiate or delete configured VTDs on a GW tier node.

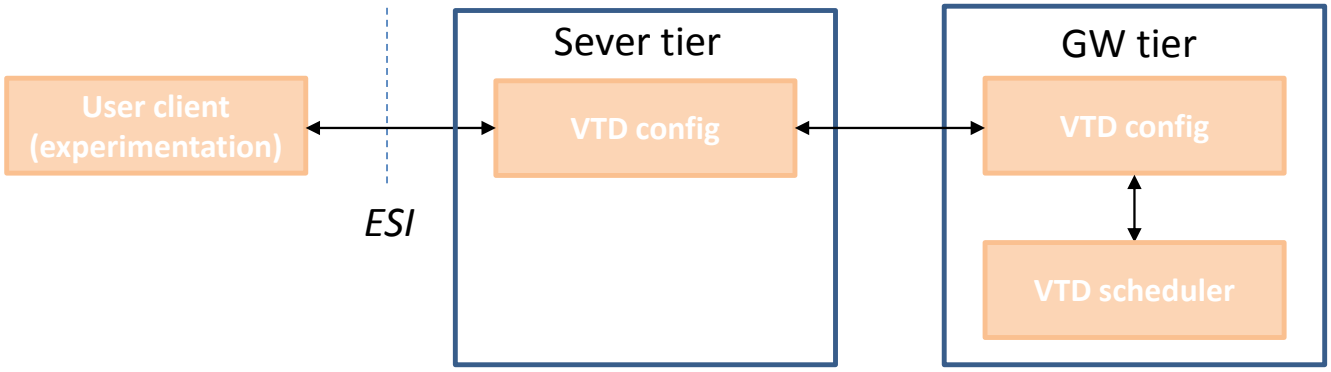

Figure 7: The virtual testbed device support can be used for the installation of experimentation code on GW tier device.

In order to enforce the configured behaviour of VTDs on GW nodes, a VTD scheduler is added as further system function to GW tier devices. The scheduler takes as input configuration parameters provided for a VTD device during configuration (FR213) and is responsible for allocating GW resource to (a set of) VTD(s) executing on it. It ensures that the experimentation support plane functionalities on the GW are not compromised and that VTDs among each other share resources according to their configured priority levels. The VTD scheduler thus addresses the requirements FR214-FR217.

4.2.5. Experimentation of Smartphones

SmartSantander's architecture supports the experimentation lifecycle with resources of the IoT node tier and the creation of services and applications that exploit the IoT generated data through the USN subsystem. At the 2nd year of the project, service level experimentation on smartphones was performed with the Participatory Sensing Application by introducing to the architecture a server

component **Psense Server** and **Psense client** at Testbed server and IoT node tier respectively. As described in the use cases included in IR1.4, experimentation on smartphones will pursue a more dynamic way of experimentation with smartphones. Experimenters will deploy their experimentation code to the end-user smartphones through the platform and then data recorded by the experiment will be gathered by the platform and finally dispatched to them. The current way of experimentation on smartphones does not allow the direct management of the experiment lifecycle through the SmartSantander platform. The experimentation code is simply deployed and executed on smartphones through the corresponding app stores of the iPhone/Android platforms. Smartphones, in contrast to other IoT devices in SmartSantander platform, are being carried by users. There is thus a need for special design components that will guarantee the uninterrupted normal operation of the smartphone and the privacy of the user as well. Several issues regarding the integration with the SmartSantander architecture have been identified, and some new components that will handle them are proposed as follows:

1. *Registration of mobile phone devices into the SmartSantander platform*. There is a need for installation of a specific application/component on mobile phones that will register the smartphone to the platform and manage (run/pause/stop) the deployed experiment. This application instantiates “SE and User Preferences Configuration” module, a software component where the end-user will give consent to contribute his smartphone and configure several parameters of experiment execution (e.g., which ones of the integrated sensors to share, how much CPU memory to share, etc.). This application will register through the “Resource Registration” component the device to the system. Registration will be handled by the “Smartphone Experimentation Server” at the testbed tier.

2. *Uploading of smartphone experimentation code to the SmartSantander system***.** Through an ESI experimenters will select the devices (number, type, etc.), reserve them for a time period and define an experiment for smartphones. The identified component “SE Manager” will consume all the parameters of an experiment and will prepare the experiment as a file to be deployed on smartphones.

3. *Validate experimentation code by the SmartSantander system.* This will be managed by the “Sanity Check” component. There is a need for a special sanity check software component regarding the smartphone experimentation code for checking specific API calls on smartphone-OS.

4. *Deployment of the experimentation code to smartphones and execution.* Deployment of experimentation code will be handled by the already identified components of Session Management at IoT tier. There is a new identified component “SE Manager” which manages the execution of the experiment and also the interventions of the users. Also there is a new identified interface SEI for facilitating the communication of smartphones to the Smartphone Experimentation Server. The actual deployment of experimentation code can be realized either by sending experimentation code to mobile phone devices directly or by deploying a native (iOs, Android) application through the corresponding application market. Experimentation deployment will be handled by the “Smartphone Experimentation Server” at testbed tier.

5. *Retrieval of Experimentation Data.* The retrieval of the experimentation data from the smartphones will be done through the new identified interface – SEI. It will be possible to retrieve experimentation messages through a specified online SmartSantander web service. The "SE Manager" component will record experimentation data. The recorded experimentation messages by the device could then be pushed continuously to a Smart Santander online web service, or with a bulk upload at the end of experiment through the SEI. Experimentation data pushback to the platform will be handled by the "Smartphone Experimentation Server" at testbed tier.

Overview of Functionalities of Identified Components

Figure 8 provides an overview of the functional components that have been identified for the realisation of the above envisioned features. The following functions are introduced on the server tier:

- *SE Configuration*: This component is responsible for the Smartphone experiment configuration, reutilizing and extending functionalities of the Configuration module (synthesis, specification, file upload, sanity check, resource selection). This component will cover FR219, FR220.
- *Smartphone Experimentation Server*: This server component will manage the scheduling of the experiments, the delivery of the experimentation code to the smartphones, the registration of Smartphones to RD, the gathering of experimentation data from the smartphones. This component will cover FR220 and NFR180, NFR182.

Furthermore the following enhancements to the IoT node tier are envisioned:

- *SE Manager:* This module will manage the experimentation code delivery to the smartphone, the execution, the monitoring of the experiment and experimental data recording. Finally, it will deliver the recorder data to the Smartphone Experimentation Server. This component will cover FR220 and NFR180,NFR181, NFR182.
- *SE and User Preferences Configuration:* This module inside Smartphone device will manage the configuration of the experimentation sharing of the smartphone by the end-user (e.g. allow specific sensors, time of day to be used, maximum memory used etc.). The module will register the Smartphone to the system. This component will also cover FR218, FR219 and NFR181, NFR182.

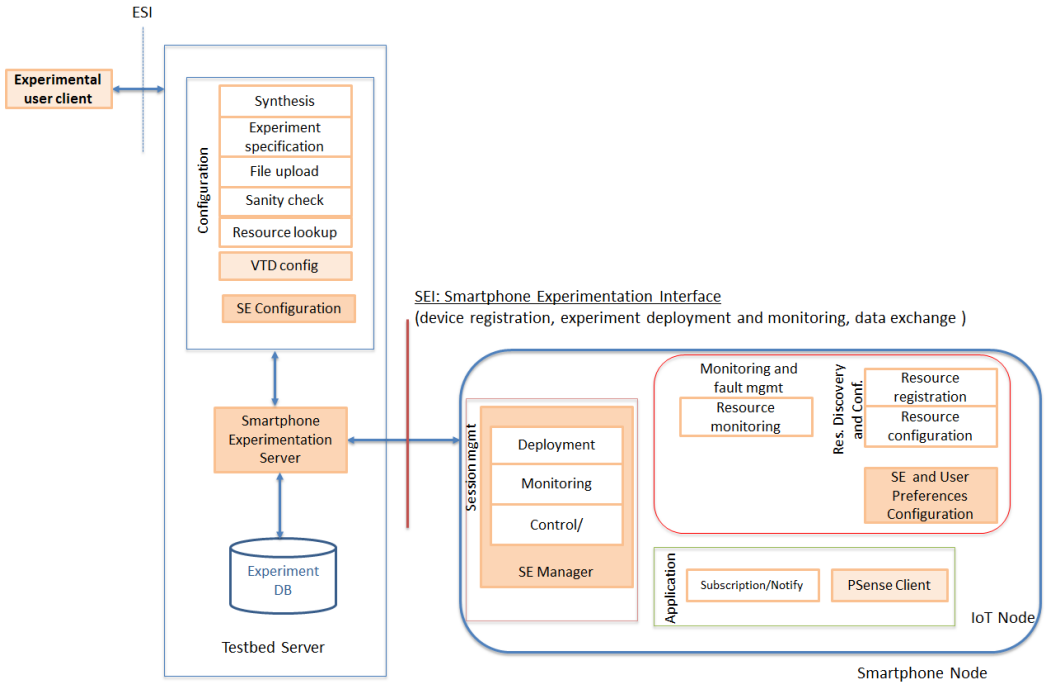

Figure 8: Identified Components for Experimentation on Smartphones

4.2.6. End-user centric securtity, privacy and trust

In section 3.5 new use cases have been introduced in order to address security threats in terms of privacy from the end user perspective. Those use cases involve personal smart phones being exploited for SmartSantander services. The use cases motivated new functional requirements for addressing such use cases (NFR180). A careful evaluation of the current SmartSantander architecture found concluded that no specific architectural extension are required to support the requirement. However some modification the *PSense server* and *PSense client* functions may be necessary.

In the following a section briefly outlines guiding design principles that must be taken into consideration for further component design:

- User awareness and willingness: the user should be informed about which personal information may be disclosed, and he should have to possibility to cancel application installation on this basis.
- Anonymization at the source: reports issued from smart phones shouldn't include any identification of the smart phone owner.
- Encryption: encryption should be used on reports sent from smart phone to testbeds so that no external observer (whether over the air or over the Internet) may have the possibility to intercept such reports.

- Anonymization at the service level: whenever possible, individual smart phone reports should not be available at the service level. Only statistics upon groups of anonymized users should be provided.
- Access control at the service level: authorization should be leveraged to avoid wide access to service that leverage smart phones, (even when previous principles are respected).

5. THIRD CYCLE SMARTSANTANDER ARCHITECTURE REALIZATION

This section outlines some design guidelines and considerations as starting points for the further design and implementation for the aforementioned architectural extensions for WP2 and WP3 respectively. It considers these for the federation, DTN support and Smartphone experimentation.

5.1.1. Federation of SmartSantander tesbed sites

The Federator which is used as proxy between multiple testbeds and a single client, features the same components for authorizing, managing and conducting experiments as the (WISEBED compatible) testbeds (FR186, FR196). Figure 9: depicts the API interaction in the intra SmartSantander federation case:

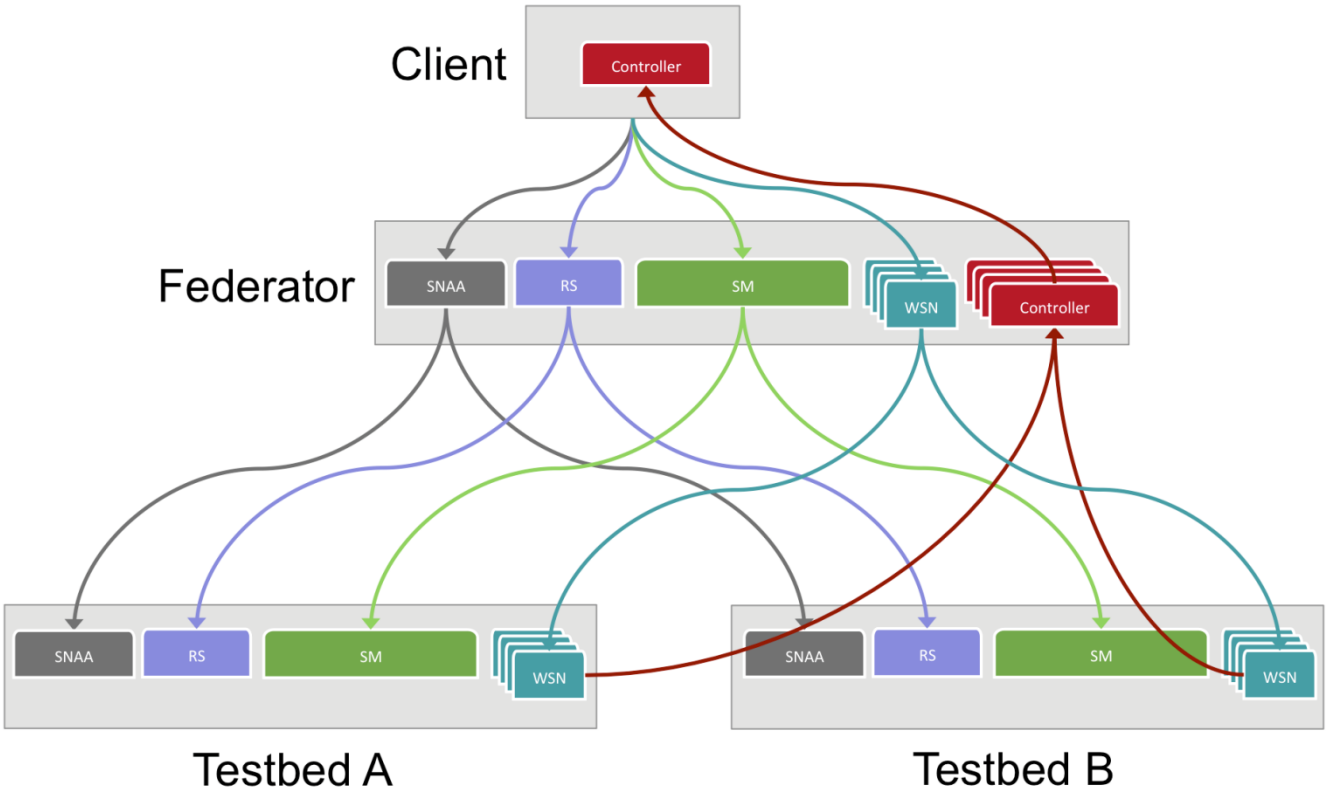

Figure 9: API Interaction in Intra Smart Santander Federation

The Federator component of Testbed Runtime implements *exactly the same APIs* (SNAA, RS, SM & WSN) as the individual testbeds in the federation. Every call that the client issues to one of the Federators API endpoints is transparently forwarded to the individual testbeds. The Federator manages the calls in two different ways:

1. For synchronous API methods, the Federator merges the returned values of the individual testbeds and returns them as one result to the client. Example: merged WiseML file when calling SessionManagement.getNetwork().

2. For asynchronous API methods, the Federator passes the calls to the federated testbeds, but exchanges the clients Controller API endpoint URL with its own. This way, all asynchronous replies will be returned to the Federator Controller API endpoint first, which then forwards these replies to the client. This way the Federator can track the status of asynchronous calls. Example: flashing nodes via WSN.flashPrograms(). The same mechanism applies for device outputs and testbed events (e.g. devices being attached/detached), i.e. the Federator subscribes to the federated testbeds on behalf of the client and forwards all messages received to the client.

5.1.2. Federation of SmartSantander with external testbeds

An initial approach for inter federation with external FIRE facilities is to transform SmartSantander testbed into and OMF/OML [OMF] [OML] and SFA [SFA] compliant platform. It is important to note that this approach intends to be a complement to the current SmartSantander platform as it does not imply that current federation mechanisms are overridden but instead enables federation with other OMF-OML-SFA enabled testbeds.

In order to cover all experimentation stages, SFA will be introduced in SmartSantander. There are several options that can be followed to evolve SmartSantander into a SFA enabled testbed. One option is to develop these interfaces from scratch. Alternatively existing libraries can be used, namely SFAWrapper and AMSoil. Both libraries provide similar functionalities but the first one is better supported by the developers' community so we will refer to it exclusively in the remaining of the document. The SFA component will then be used to provide authentication, resource discovery and reservation.

The SFAWrapper approach requires the development of SmartSantander driver components behind the wrapper libraries will interact with the following components in the following way:

- The Resource Directory (RD) will provide the WiseML formatted file with a detailed description of the SmartSantander resources, which will be translated into a valid RSpec format.
- Once authenticated a valid temporal user will be created in the TR-SNAA user data to be used in future reservations.
- If an authorized experimenter wants to reserve a set of resources for Native Experimentation, TR-RS will be asked for this resources availability. The generated reservation key will be returned for future interactions with the TR-iWSN.
- If an authorized experimenter wants to reserve a set of resources for Service Experimentation, the SmartSubscriber component will be notified to include these resources in the set of running subscriptions to the iDAS platform. This component is yet to be developed.

The second enabler for exposing SmartSantander to external facilities will be OMFv6. It will be used in the description and provisioning of the experiments. Besides, OML will be used in this context for measurement collection both in Native and Service Experimentation. FRCP will be used to allow the

creation of real time per-experiment RCs, as well as to remove them as soon as all resource reservations have ended.

The OMF/OML interactions with the SmartSantander infrastructure will include:

- All RCs will include one or more OML MP which will be instructed during the configuration stage to redirect the desired OML measurements to one or more OML Servers. The SmartSantander AM OML Server or any other one in the experimenter domain could be used, but for Service Experimentation the later approach should be mandatory in order to avoid information duplicity.
- In Native Experimentation, RC will also control a Testbed Runtime Experiment Controller (TR-EC). The RC will have to translate all OMFv6 ED primitives to Testbed Runtime API calls. All those Native Experimentation RCs will include a property containing the TR-RS reservation key that will be set up during the configuration stage.
- OML will also be used in a static way to provide Monitoring capabilities to SmartSantander. Those OML MPs will be subscribers of the Service Aggregator in the SPGW, and will be configured to send all its measurements to the Monitoring OML Server.

5.1.3. Design Considerations for DTN-Support Realisation

We illustrate our notion of DTN-based node reprogramming by providing an instantiation of our reference architecture using existing components in the SmartSantander platform. This is shown in Figure 10 below.

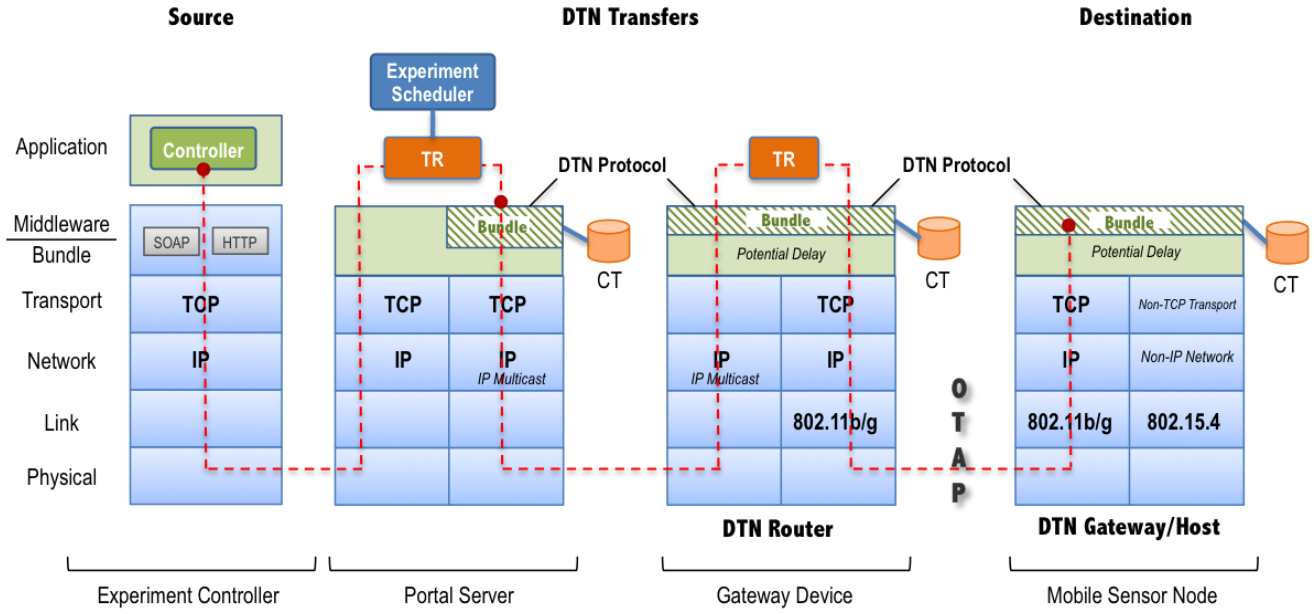

Figure 10: Bundle-Layer-brokered interactions between SmartSantander components for DTN support

Transparent Transport Service

Bundle Layer component instances implement the DTN protocol functionality at the Portal Server, Gateway and mobile IoT node. In essence, the delay and intermittent connectivity associated with mobile nodes is isolated and made transparent to the components using the transport service for data transfers. Therefore, the same transport API must be exported by the BundleLayer component as the current message transfer API to components such as the Testbed Runtime. In essence, the BundleLayer will use the existing event buses (described in D2.3 [D2.3]) used for enabling interactions between nodes belong to the different tiers.

Bundle Custody Transfer

Another design consideration is the caching of messages (e.g. software image bundles) at the source (Portal Server) and intermediate DTN routers (Gateway nodes). This is necessary for the supporting retransmission of messages in the presence of transfer errors. The responsibility of retransmission can be handed-over to the next DTN router along the path of propagation. The DTN router then becomes the custodian of the received bundles and ensures that these bundles are delivered to the next-hop or destination node. The custodian DTN router also has the responsibility of sending an ACK back the source (Bundle Layer at Portal Server) to indicate successful delivery of the fragments. Custody transfer (shown as CT in Figure 10) from one DTN router to another occurs through an exchange of request/response messages.

Multi-Homed Mobile IoT Nodes

Although the DTN-protocol is independent of the underlying protocols in the protocol stack, network interfaces that offer higher bandwidth for data transfers will improve the efficiency and throughput of the DTN protocol. This is because data transfers are constrained by the duration of the contact between the mobile IoT node and the Gateway devices. For instance, switching from the 802.15.4 to 802.11b/g MAC protocols is recommended if a WiFi connection is detected. This assumes that mobile IoT nodes are multi-homed and support 802.11b/g connectivity. The support of a TCP/IP stack on top of 802.11b/g on the mobile sensor node may be explored but may ultimately not be possible due to resource restrictions on the embedded devices.

Further details about the interface of the Bundle Layer and it's interactions between existing/additional components are provided in the project report deliverable D2.3 [D2.3].

5.1.4. Experimentation of Smartphones

Several architectural components have been identified in section 4.2.4 to support the experimentation on Smartphone. These are the *Smartphone Experimentation (SE) Server* and *SE Configuration components* at testbed server level and *SE and User Preferences Configuration* module and *SE Manager* component at smartphone level.

At *Testbed Server Tier* the main component would be the *Smartphone Experimentation Server***.** This server component will be the core component that will manage most functionalities regarding experimentation with smartphones. It will manage the registration of Smartphones to Resource Directory, scheduling of the experiments, the delivery of the experimentation code to the

smartphones, the gathering of experimentation data from the smartphones. Moreover, it will employ the second identifies component at testbed server level the *SE Configuration* module**.** This component will be responsible for the Smartphone experiment configuration, reutilizing and extending functionalities of the Configuration module realizing/extending functionalities regarding: synthesis, specification, experimentation code upload, sanity check, and resource selection of smartphone experiments. Regarding implementation issues it will be used/extended already existing service components (e.g. RD clients, web services, resource selection, resource registration as in Participatory Sensing scenario).

At *Smartphone Tier* the main identified component is *SE Manager***.** This module will manage the experimentation code delivery to the smartphone, the execution, the monitoring of the experiment and experimental data recording. Finally, it will deliver the recorder data to the Smartphone Experimentation Server. This module will employ the other identified *SE and User Preferences Configuration* module. This component will manage the configuration of the experimentation sharing of the smartphone by the end-user. The user will configure how his smartphone will be used during an experiment. He will give access to specific sensors; it would be possible to define time of day to be used, set various thresholds like maximum memory used, if my battery is below 20% stop the experiment etc. The module will perform also the license agreement with the end-users and will register the Smartphone to the system by communicating with the *SE Server*. As the main targeted platform are Android smartphones, in D2.3 several Android technologies are consider for realizing the remote programming of smartphones, of sandboxed code execution and monitoring and of secure communication schemes of smartphones to the main platform (https, use of certificates etc.).

6. CONCLUSIONS

SmartSantander was conceived to evolve according to three main iterations during the three years' time frame project duration. The consortium has converged towards a final architecture, which besides fulfilling and accommodating the requirements identified during the last phase of the project also provides the basis for making the platform fully interoperable with FIRE facilities.

Apart from the requirements intrinsically identified by the work committed from the initial phases of the project, we have also addressed as in previous years the recommendations of reviewers during last technical review held in October 2012 [GENREW].

Bearing in mind all these aspects, the architecture firstly reflects the needs emanating from the federation of the SmartSantander sites relying on the TR paradigm. Aiming at exposing the whole facility towards the external world the decision was to adopt SFA jointly with OMF and OML. The former is aligned with the decision taken in the FED4FIRE [FED4FIRE] project which pursues among others the interoperability among all the experimentation facilities. The decision on OMF and OML is consistent with what was decided in the FIRESTATION architectural board [FISTA] and assessed by the European Commission. Having both OMF and OML will increase the usability of the SmartSantander platform, because many experimenters in the community are using such framework when designing experiments and collecting data.

Besides federation it was also considered relevant to analyze the possibilities and implications of using WIFI and higher bit rate radio interfaces instead of cellular ones. This has also been linked to the concept of DTN aiming at collecting data using an opportunistic approach. Although sophisticated solutions might be proposed we will demonstrate such functionality as part of WP3 and relying on the equipment already running in the city. The basic idea is to make use of the IEEE 802.15.4 devices embedded on top of the public buses for downloading data once they are entering in the bus depot. We are aware that the bit rate is not really high, however the key aspects of the concept will be demonstrated and once the future equipment will be enriched with higher wireless bit rate interface, such as IEEE 802.11x, the solution provided will be equally valid.

Another important aspect we have addressed involved the experimentation on top of Smartphones. This requirement is inherent to the participatory sensing application running in Santander. In this document several use cases have been elaborated aiming at gathering the different views of the involved stakeholders, that is, the smart phone owner, the experimenter and the manager of the experimentation platform. When writing this document we can say that the state of the art does not provide yet an enough mature solution. Hence, we will be analyzing and eventually materializing a basic implementation. Anyhow, the architecture already reflects the required additions.

Last but not least, main considerations in terms of security and privacy from the en-user (as a citizen) perspective have been included as well as the corresponding requirements. These have been analyzed upon the experience we got from SmartSantanderRA and The Pace of the City applications running for more than 6 months both of them and with more than 22,000 downloads (April 2013).

7. REFERENCES

[Anglano]	Anglano, C., "Interceptor: middleware-level application segregation and scheduling for P2P systems," Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International , vol., no., pp.8 pp.,, 25-29 April 2006
[D1.1]	SmartSantander Project's WP1 Deliverable D1.1, Initial Architecture Specification
[D1.2]	SmartSantander Project's WP1 Deliverable D1.2, Second Cycle Architecture Specification
[D2.3]	SmartSantander Project's WP2 Deliverable D2.3, "Final Component Design"
[FED4FIRE]	FED4FIRE project, http://www.fed4fire.eu/
[FISTA]	http://www.ict-fire.eu/firestation
[GENREW]	SmartSantander Second Technical Review Report
[LXC]	Linux Container Tools, website: http://lxc.sourceforge.net/
[OMF]	Thierry Rakotoarivelo, Maximilian Ott, Guillaume Jourjon, and Ivan Seskar. 2010. OMF: a control and management framework for networking testbeds. SIGOPS Oper. Syst. Rev. 43, 4 (January 2010), 54-59. DOI=10.1145/1713254.1713267 http://doi.acm.org/10.1145/1713254.1713267 Website: http://mytestbed.net/projects/omf6/wiki/Wiki
[OML]	Olivier Mehani, Guillaume Jourjon, Thierry Rakotoarivelo, and Max Ott, "An instrumentation framework for the critical task of measurement collection in the future Internet," NICTA Technical Report, No. 6065, July 2012 [Online]. Available: http://www.nicta.com.au/pub?id=6065 Website: http://mytestbed.net/projects/oml/wiki
[Privacy]	Na Li, Nan Zhang, Sajal K. Das, Bhavani Thuraisingham. "Privacy preservation in wireless sensor networks: A state of the art survey.", Elsevier Journal, Ad Hoc Networks
[SFA]	Slice-Based Federation Architecture (SFA), PlanetLab Europe Management-Level Documentation Set, http://doc.planet-lab.eu/html/x3129.html

	Website: http://groups.geni.net/geni/attachment/wiki/SliceFedArch/SFA2.0.pdf?format=raw
[WiseBed]	Geoff Coulson, Barry Porter, Ioannis Chatzigiannakis, Christos Koninis, Stefan Fischer, Dennis Pfisterer, Daniel Bimschas, Torsten Braun, Philipp Hurni, Markus Anwander, Gerald Wagenknecht, Sándor P. Fekete, Alexander Kröller, and Tobias Baumgartner. 2012. Flexible experimentation in wireless sensor networks. Commun. ACM 55, 1 (January 2012), 82-90. DOI=10.1145/2063176.2063198 http://doi.acm.org/10.1145/2063176.2063198 Website: **http://www.wisebed.eu**