UNIVERSITY OF PADOVA

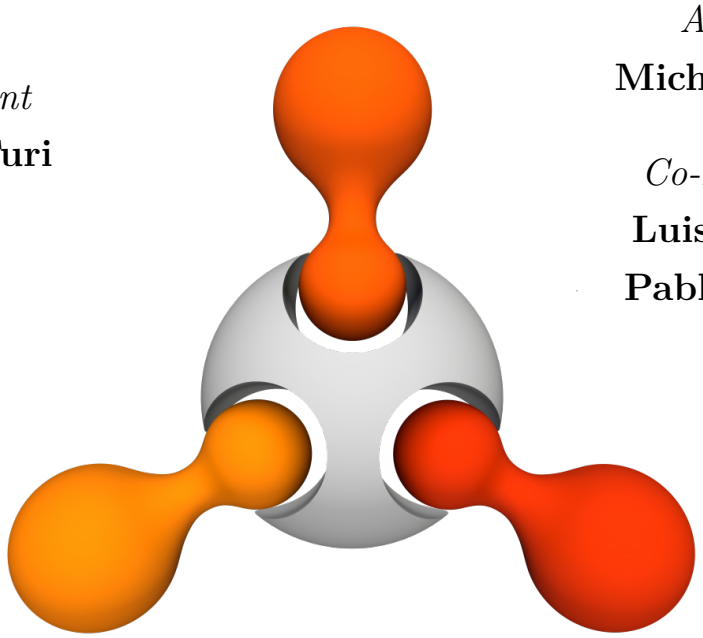
Master Degree in Telecommunication Engineering

Contribution to the Federation of the asynchronous SmartSantander service layer within the European Fed4FIRE context.

Student
Leo Turi

Advisor
Michele Zorzi

Co-Advisors
Luis Muñoz
Pablo Sotres



Department of Information Engineering

Academic Year 2014/2015

13^{th} October, 2015

To my parents.
For all they do, all they go through while I am away, and for all the love they give me.
Every time, everywhere I am.
I love you.

To Pippo, Matt, Fede, Gi and Eleo. And to Gabb and Marco, newly found.
You're the brothers and sisters I never had.
I'm glad I met you all.

To the "malvagi", our beautiful and varied TLC group.
I hope we won't lose contact in the next years, for I would miss you all deeply.
You were the best classmates ever.

To all the guys of the TLMAT lab: JuanRa, Rafa, Carmen, Luisco, David, Jabo, Javi, Laura, Nacho, Pablo Martinez y Pablo Garrido (el Aleman!), Vero, Jose, Jesus.
To Ramon Aguero, Luis Sánchez and Jorge.
You were the warmest hosts I could ever find.

And, last but not least,
To Pablo Sotres, who was my tutor at Unican, helped me write my thesis,
and always found time to assist me (Sundays and Holidays as well).
I'm happy your eternal cough ceased way before I left.

$\sim$

You have my deepest thanks.

Finally, I'd like to acknowledge the guide of Prof. Zorzi and Prof. Muñoz.
Their words and trust have been fundamental during this important experience.

"It's the questions we can't answer that teach us the most. They teach us how to think. If you give a man an answer, all he gains is a little fact. But give him a question and he'll look for his own answers."

– Patrick Rothfuss, *The Wise Man's Fear*

Abstract

This thesis is a contribution to the federation of asynchronous SmartSantander service layer within the European Fed4FIRE context. The thesis was developed in a Smart City background, and its main aims were both to gain knowledge of how Smart Cities, Testbeds and Federations of Testbeds are structured by working on a real deployed system, i.e. SmartSantander framework and Fed4FIRE federation, and to contribute with some of the components required for the integration.

The technical development carried out as part of this thesis mainly deals with three aspects of the testbed: resource discovery, asynchronous subscription management and measurement data delivery. As a result, a series of software components have been deployed on SmartSantander hardware and it will be running as part of the complete framework on the next testbed iteration. Together, they provide a new way of accessing to the sensor information SmartSantander can provide.

During the development phase, we experimented with real hardware and software and worked with off-the-shelf technologies for testbed and federations. The complete work was developed at the University of Cantabria in collaboration with the TLMAT laboratory, which currently presides over SmartSantander.

Table of contents

List of figures

Chapter 1

Introduction

1.1 Motivation

Testbeds are ICT experimentation sites usually deployed on real hardware, although they can sometimes be deployed in software running on virtual machines. Their purpose is to test and validate different technologies in the ICT field, like routing algorithms, load balancing techniques, network coding, system scalability and many others.

An issue concerning testbeds, though, is that it is hard to implement and sustain large facilities. As a consequence, it's unlikely that users set-up their own experimentation site for their own experimentation purposes. Entities that can maintain extensive testbeds make up for this fact, providing external users access to their resources by means of ad-hoc interfaces. This works only as long as the visibility of these testbeds is large enough, so that users may be aware of their availability. Moreover, the services offered need to find enough demand by the researchers, or else the large facility may remain unused. At the same time, it is in the interest of testbeds that their experimenters are reliable, certified, so that they don't waste their resources, as an extreme example, for malicious purposes like DDoS attacks.

To aid visibility, diversity, compatibility and security, it is often a best practice for general service providers to join together with other compatible providers, and form associations that work both as multiple service providers and share security schemas. In the testbeds' perspective, the role of these large associations is played by the so called "federations".

Federations are a recent concept, which hasn't been standardized yet, and is still evolving through the years. For starters, they are groups of inter-operative testbeds, that share their resources and use the same external interfaces to serve users. Second, they work as certification authorities, where users need to be registered and possess a

valid, standardized certificate, that validates that they are trusted users of the system. Finally, Federations are also showrooms, where testbeds can increase and share their support community, grow together, and be able to more easily sustain maintenance expenses.

Different federation paradigm and structures have been implemented throughout the last decade, but a comprehensive overview of these paradigms, as well as testbeds, has not yet been done. This thesis work aims in part to do so, in a sort of compilative manner; although, it still falls under the category of "experimental thesis", because a large part of it deals with the implementation of a complete system service inside a consolidated framework.

On the latest months, different works have been carried out to federate SmartSantander testbed inside the European Fed4FIRE project, which is a Federation composed by more than 15 heterogeneous testbeds. The work done during this thesis contributes to this task, hence is heavily influenced by the existing requirements and constrains imposed by that project.

1.2 Objectives

In this section we state the objectives that lead to the development of this contribution. The fundamental objective was the final contribution (1.2.3), but in order to achieve it there was the need to get a deeper understanding of the smartcity and federation of testbeds concepts.

Understanding the IoT and SmartCity concepts

IoT is still an active field of development and research. Hence, one of the objective of this thesis was to learn how IoT can be applied to everyday life, and which are the infrastructures required to sustain and maintain a wide IoT network. We focused on the concept of SmartCity because it is one of the most promising, and is already developed enough to provide a handful of practical examples, let alone the training site the thesis was developed into, SmartSantander.

Understanding the Testbed and Federation of Testbeds concepts

Another objective was to get to know and understand how a real, deployed testbed work. Moreover, learning how to use the tools as well as learning the project structure

is necessary also for the successive objective: to understand exactly the purpose of federations, and how they aim to join testbeds and extend their capabilites.

Understanding how SmartSantander and Fed4FIRE work

All this knowledge would be useless in case it were not applied somewhere. Therefore, it is important to make practice and understand how the actual training site works, i.e. SmartSantander and the European Fed4FIRE federation. The development of the thesis requires to employ the majority of the tools offered by the two environments, and understand exactly the protocols, interfaces and technology they communicate with.

Integration of Fed4FIRE protocols into SmartSantander Service Layer

The last objective is also the most important, because it is covers all the practical implementation aspects. All the previous objectives aim to enable this part: any addition or contribution in SmartSantander and Fed4FIRE context requires deep understanding of the systems. Therefore the aim was to develop an useful component for SmartSantander framework, in accordance with Fed4FIRE specifications, so as to contribute to the federation of the Spanish testbed within the European federation context. The contributed work is finally required to be successfully deployed to release, and to be up and running as a SmartSantander framework component by this thesis completion.

1.3 Document Overview

This thesis continues as follows. In Chapter 2 we present the **General Concepts** that allow to comprehend this thesis. In Chapter 3 we introduce **SmartSantander Testbed** structure, with a particular focus on its history and latest development. In Chapter 4 we then introduce **Fed4FIRE Federation** structure, the european initiative it fits into (FIRE) and its main objectives. Later, in Chapter 5 we provide details on the structure of the **Technical Contribution** derived from this thesis' work, how it fits between SmartSantander and Fed4FIRE, how it is designed, and how it works; all this by means of detailed description of the system, pieces of code, as well as practical demos. Finally, in the last chapter, Chapter 6, we discuss our results, take our **Conclusions** and make also a brief analysis of possible **Future Works**, extensions to the system, and improvements.

Chapter 2

General Concepts and State of the Art

In this chapter we give a brief introduction to the three main concepts behind this thesis: the concept of *testbeds*, that of *interoperability* between testbeds and, finally, that of *federation* of testbeds. Among these, the latter is pivotal, as the whole work revolves around it. Briefly, the chapter is structured as follows:

- first, we explain what a testbed is, providing examples of its structure by describing existing deployments (in particular, with a focus on smart city ones);
- then, we mark the importance of cooperation and interoperability, which is fundamental to extend testbed visibility, capabilities and provisioning;
- finally, we introduce federations, which are self-sufficient organizations of cooperative testbeds, usually supported by the European Union or private corporations, and may be the natural continuum of the testbed paradigm.

2.1 Testbed and Smart City

2.1.1 Testbed

Definition 1 *"A testbed is a platform for conducting rigorous, transparent and replicable experiments." [1]*

In ICT, testbeds are composed by a large number of communicating devices (or *resources*) which form a network, and that can be monitored, managed and reprogrammed from remote. The users that submit experiments to the testbed can be either

internal or external to the testbed organization, and are generally called *experimenters*. In order to provide then full control over experiments, a reservation, submission and monitoring system is usually employed.

The experiments could meddle with brand new routing algorithms, network coding, distributed data storage, cloud computing, P2P or M2M communication, as well as many other topics. The experimentation purpose defines the testbed requirements, which are expressed in terms of extension, location, device typology and communication interface.

These characteristics are what directly define the kind of network - and therefore of testbed - we have. As an example, a Wireless Sensor Network [WSN] [2] could be composed of a fixed or varying number of wireless sensing devices, and be designed to be either indoor or outdoor, depending on the communication interface (meters or kilometers of range). On the other hand, a Mobile Ad-Hoc Network [MANET] [3] contains a large set of 3G/LTE-A/5G terminals together with few base stations, and usually spans over kilometers; fiber-connected devices form Fiber Networks, whose extension can reach up to hundreds of kilometers for testing purposes [4]. Another example can be Underwater Networks [UWN] [5], which may be composed of a mixture of acoustic and optic communication devices, designed to be set in a large underwater environment.

The testbed could also be specialized to provide specific testing conditions. As an example, LTE-A is designed to provide connectivity to high speed trains, moving at around 200 km/h [6] at several km of distance, therefore special equipment is needed to test this feature. At the other hand of the spectrum there are optical UWN, which attain communication only if the transmitting laser device is almost perfectly aligned with the optical receiver interface, if it has high directionality, and if the communication is within 100 meters due to high attenuation in water environment [7]. The difference in water temperature and micro materials concentration could play a huge role and requires to be tested thoroughly.

As specialized architectures could be expensive and difficult to provide, the testbed could otherwise employ already existing, real-world technology. In the previous example, high speed performance could be evaluated by positioning an LTE-A terminal on top of a local train, and test its performance on a daily base in a real-world setting. Take as reference Samsung early tests on 5G communication [8]. By testing in a real-world scenario (although lacking interference), they were able to reach 1.244 Gbps in down-link using 5G 28 GHz communication. The system set up was a single 5G base station

transmitting to a single mobile station. The mobile station was set on top of a car, which was cruising at 112 km/h in the Everland Speedway, Korea.

Of course, the opposite need could arise in another context: that of a general purpose testbed, which is mostly designed in software and can be easily customized by the experimenters. Let an example be future Internet testing, which requires very large testbed with IP connectivity, heterogeneous interfaces and devices with specific functionalities (DHT, DNS, etc).

We employ this two different applications to classify testbeds into two main categories: *real-world deployments* and *laboratory testbeds*. The latter is the most adherent to the definition of testbed we provided, as experiments are easily replicable. In addition, devices could be also virtual, and communication channels are highly supervised, up to the point where channel noise could be in part or completely generated [9]. Real-world deployments are on contrary testbeds set in an uncontrolled environment, be it an urban or a natural area, where experiments are not easily replicable, and channels are absolutely not ideal. The purpose of such testbeds is indeed that of providing *real world data*, which is fundamental for real-world application validation, a context where laboratory conditions do not fit.

Spread around the world are a large number of consolidated testbeds which belong to both these categories. Some testbed may be split into the two categories, therefore those are classified as “hybrids". Let us now briefly list some of the existent testbeds, and give additional details on the most significant ones:

- *Laboratory: NorNet [10], Helsinki Testbed [11], w-iLab.t [12]*

  e.g. *StarBED*[3] [13]: a large-scale emulation testbed with more than 1,000 nodes. Supports virtualization and emulation, allows to use multiple OS (SpringOS, Xburner, etc), and the experiments can be easily expanded with external hardware. A common example is a general-purpose IP network simulator, which can be customized as desired.

- *Real-World: HarborNet [14]*

  e.g. *MoteLab* [15]: a Web-based sensor network testbed, consists of a set of 26 permanently deployed wireless nodes connected to a central server. The nodes can be reprogrammed through an Ethernet interface, and communicate using wireless 802.15.4 protocol. The back-end server handles reprogramming, data logging, and provides a web interface for running jobs on the testbed.

- *Hybrid: WHYNET [16]*

  e.g. *ORBIT (Open Access Research Testbed for Next-Generation Wireless Networks)* [17]: an indoor radio grid emulator for controlled experimentation and an outdoor field trial network for end-user evaluations in real-world settings. The indoor system is composed of a 1m-spaced 8x8 grid of 802.11a/b/g wireless nodes, connected via multiple Ethernet links to transfer applications and control/management information. The outdoor system is a 5x2 km that provides a real-world testing environment for either cellular (3G) or wireless (802.11x) access networks.

This thesis work orbits around the real-world testbed category, as it was developed in the SmartSantander environment. SmartSantander is a huge testbed, which collects more than 20.000 sensors, and was deployed in the omonimous city of Santander, Spain. The testbed is deeply entwined with the city structure and management, to the point that the city of Santander can now be called “smart". Therefore, to continue we first need to introduce a specific concept: that of Smart City.

2.1.2 Smart City

Smart City is among the latest entries in the ICT vocabulary. Given the breadth of technologies that have been implemented under the smart city label, a comprehensive definition is hard to find, and many have expressed their own unique definition. Though, similarities in these definitions arise frequently, therefore we will use these underlying similarities to build a most comprehensive definition.

Giffinger et al. provide their own definition in a 2007 survey [18] on the smartest cities of Europe, and state that “Regional competitiveness, transport and Information and Communication Technologies economics, natural resources, human and social capital, quality of life, and participation of citizens in the urban governance" are pivotal in the smart city context.

Deakin and Al Wear [19] define the smart city as one that utilizes ICT to meet the demands of the market (the citizens of the city), and that involves the community actively. A smart city would thus be a city that not only possesses ICT technology in particular areas, but has also implemented this technology in a manner that impacts the local community.

Later on, a more classificatory approach is present in 2013 report on Smart Cities, *Strategic Opportunity Analysis of the Global Smart City Market* [20], where Frost & Sullivan identify “eight key aspects that define a Smart City: smart governance, smart

energy, smart building, smart mobility, smart infrastructure, smart technology, smart healthcare and smart citizen."

An IBM report on Smart Cities [21] polishes these key aspects, by removing the "Smart Technology" entry, as technology permeates all the other fields. In addition, the aspects are separated into three main categories:

1. *Planning and Management:* Public Safety, Government and Agency Administration, City Planning and Operations, Buildings.
2. *Infrastructure:* Energy, Water, Transportation.
3. *People:* Education, Smarter Care, Social Programs.

Now, the general definition, that we will use from now on, derives from the common ground of these concepts:

Definition 2 *"A Smart City is a city that uses digital technologies or ICT to enhance quality and performance of its fundamental services. These technologies should provide better urban mobility and transportation, smart management of urban finance and expenses, improved governance and administration, more efficient, greener and friendlier industry together with an extended involvement and interaction with the citizens. All this, provided a keen eye is kept on security and privacy."*

For what regards technological aspects, ISO's Smart Cities report [22] identifies the following key characteristics that smart city technology should provide:

- Scalable data collection about city life;
- Data aggregation;
- Easy-to-use data visualizer;
- Real-time, detailed knowledge about the city;
- Analytics and decision-making systems, which should possibly involve the citizens.
- Automation of specific city functions;
- Networks of collaborative spaces;

ISO team identified also specific city needs that technology should satisfy. These are thoroughly detailed inside the text, and, in general, they are split into three categories:

technological (e.g. reliable communication), market (e.g. secure and adaptive markets), societal (e.g. improve quality of life).

Requirements and needs allow to build smart city system models, which can be very detailed or only general, but which are fundamental to define the technologies required. In addition, ad-hoc frameworks, domain knowledge and data/services models are sketched.

Next comes the list of fundamental technology. All text within commas has been extracted from [22]:

- *Ubiquitous computing*, i.e. the pervasion of computing devices in the whole smart city environment. This requires distributed systems and multi-hop communication technologies to be implemented, as well as secure access methods and the set up of valid certification authorities.

- *High performance and capacity networking*, by means of "FTTH, 4G LTE, IP Multimedia Systems, as well as future networking technologies". This helps improving the pervasiveness of digital technologies.

- *Open Data*, which "generally refers to a public policy that requires public sector agencies and their contractors to release key sets of government data to the public for any use, or re-use, in an easily accessible manner". An example are GPS and mapping data, which could be combined to provide diverse location services. Data discovery, collection and reservation should be accessible through ad-hoc APIs.

- *Big Data*, a cover name to identify any technology to handle very large data sets in an efficient and clever manner. Given the size that cities take, and the number of devices that everyday should collect and send data to the city main databases, efficient algorithms and scalable systems should be provided, so that further data aggregation and analysis are not limited by any kind of external bottleneck.

- *GIS (Geographical Information System)* is fundamental in the smart city context, as location based services would benefit many everyday urban aspects (public transportation, parking, etc...).

- *Cloud Computing* is "the delivery of computing as a service rather than a product", and as clouds may be public, private or hybrid, both citizens, business and the government could benefit from it. Transparent and reliable interface to the middleware should be provided and accessible to the developers.

- *Service-Oriented Architecture (SOA)* is "a software design and software architecture design pattern based on distinct pieces of software providing application functionality as services to other applications." The use of SOA would improve any other aspect of the system, allowing to create a "flexible, shared model that leaves room for scalable, incremental growth."

- *E-government* "refers to the utilization of Information Technology (IT), Information and Communication Technologies (ICTs), and other web-based telecommunication technologies to improve and/or enhance on the efficiency and effectiveness of service delivery in the public sector". To implement this in the city environment, standardised technologies and infrastructures that are necessary to provide personalised and location-based services need to be developed.

- *Embedded networks* "of sensors and devices into the physical space of cities are expected advancing further the capabilities created by web 2.0 applications, social media and crowdsourcing"

- *Internet of Things (IoT)*, i.e. "the interconnection of uniquely identifiable embedded computing like devices within the existing Internet infrastructure". It is an important emerging strand, whose worldwide market is fore-casted to be of $7.1 trillion in 2020 [23]. IoT is the perfect medium to provide a city with a pervasive system, due to the diverse types of devices that can be used, their long-lasting battery life and efficient communication protocols. The current issue with IoT regards system and device security, which lacks in many aspects [24] and is currently a hot research topic [25] [26].

An example of real-world system is given by Thales [27], thanks to their supervision system called Thales Hypervisor. It supports the smart city concept through its ability to coordinate separate urban information systems. It is open, has a service-oriented architecture, and enables interconnected systems and subsystems to share the data needed to optimize individual applications as needs evolve. The system was developed and deployed in a handful of cities around the world:

- *Mexico City* - showcases the world's largest urban security system, with 8000 video cameras, gunshot sensors, automatic number plate recognition cameras, aerial surveillance drones and emergency call points in the city's streets. Five local command-and-control (C2) centers control these subsystems, and are supervised by a C4I center (command, control, communications, computers and intelligence).

The system transmits in real-time the information to police, fire crews and emergency services.

- *Strasbourg* - illustrates the trend towards integrated city management by combining its traffic control operations with an urban video surveillance center, deployed by Thales.

Smart cities are only the last step in the technological evolution that we went through in these last 50 years. As of now, 54% of world population lives in urban areas, and this percentile is forecast to increase to 66% by 2050. Also due to this fact, Smart Cities may represent the ideal "future city", which could be able to achieve the long-lasting benefits we stated above, at the expense of seldom additional expenses in technological updates, ad-hoc installations and infrastructure management. Due to their novelty, there is still no proof that, on the long run, the savings will overcome the initial expenses, but, if we also value how important is citizens quality of life and health, the premises are very good.

Future developments in this paradigm go towards intra-city connectivity and association, in a way that multiple cities join to provide better services to the population and to care for each other's lack.

2.2 Federation and Interoperability

2.2.1 Interoperability

Smart Cities show that, as we previously said, testbeds can be of any size and extension. Yet, they also show that building huge testbeds is very expensive, independently from the mere cost of the technology employed. Infrastructure deployment and maintenance could be a heavy toll for small authorities like universities, small laboratories and the city itself.

The lack of resources needs to be overcome to build extensive testbeds. There could be many solutions, from searching for external private financing, to crowd funding, or to change the system architecture and enabling virtualization/multiplication techniques.

Still, one could recall what Aesop once said, that is "in union there is strength" [28].

Therefore, here comes into play the concept of "interoperability":

Definition 3 *"The ability of systems, units, or forces to provide services to and accept services from other systems, units, or forces and to use the services so exchanged to enable them to operate effectively together." [29]*

Interoperability among testbeds is fundamental to extend testbed functionalities and dimensions, and to satisfy the ever increasing experimentation requirements. In addition, cooperation between testbeds also benefits the experimenters, which then may enjoy the additional available resources, the now-standardized testbed tools, or even, in case the testbed interoperate smoothly, may gain from the obfuscation of all the steps between experiment designing and deploying. The improved ease of use allows also unexperienced experimenters to easily gain knowledge of the inter-operative system, which eventually may led to an increase of the community backbone and that users can help all the inter-operative testbeds to gain popularity.

Let an example be the Global Internet: under a fuzzy point of view, it is but the largest inter-operative network in the world, and accepts multiple protocols, multiple type of devices, multiple types of connection; moreover it is very general-purpose, and any user can easily employ it without knowing the global architecture. This is another fundamental aspect of interoperability: through it, a testbed can improve not only in dimension, but also in variety of devices and connections.

As a natural trade-off, though, working interoperability is not an easy feature to obtain. The most common issues are the following:

- Network heterogeneity requires additional interfaces to let the testbed communicate and share resources;
- Some testbeds may scale worse than others, becoming bottlenecks for the network;
- Resources must be managed and reserved through secure and organized tools.

The most serious issue among those is resource management and reservation. Let two testbed A and B be inter-operative. Testbed A has its own private reservation methods, that takes into account also B's resources. The same happens for B. What if two different experimenter choose to reserve all the resources in the network, but one researcher reserves them through A, and the other through B? If the two testbed reservation systems don't communicate one another, there most likely will be a collision between requests, and both the experiments will either be unsuccessful or suspended.

Therefore, this inter-operative network of testbeds must be fully organized. May it be a P2P network, a centralized one, or any other kind of network. The importance is that all the fundamental services like resource reservation and management are either centralized or, in case of a distributed system, synchronous.

Through the years, different projects were developed to enable interoperability within testbeds [30] [31] [32], but current state-of-the-art in testbed interoperability

is the concept of "federation". In the following section, we focus on the concepts underlying federations architecture.

2.2.2 Federation

Definition 4 *"A Federation is an organization or group within which smaller divisions have some degree of internal autonomy." [33]*

In our context this definition fits perfectly: a Federation of Testbeds is an organization within which *the single testbeds* have a certain degree of internal autonomy. In practice, the federation interconnects all the federated testbeds and manages all the interconnected operations, from data routing to device reservation, while each testbed is free to independently manage its own resources.

Federations could be considered as the natural step in the evolution of testbeds: mounting, managing and maintaining a testbed can be, on the long run, very expensive and time consuming [34] [35] [36]; therefore, systems and organizations that ease these tasks and plan to provision multiple testbeds [37], become necessary for the survival of testbeds.

Other than providing sustainability to testbeds, federations are also bridges between testbeds, and enable to expand the federated testbed capabilities and features, as well it provides easy-to-implement tools to ease testbed management.

Example: Let's suppose that an experimenter, Steve, requires a large testbed of heterogeneous devices and connection interfaces. Let's suppose that a certain degree of security is required to run the experiment, e.g. IP sec connection. Let's suppose that three testbeds are fit for his needs, but only in part. Let testbed A be a small testbed with a rich selection of interfaces; let then testbed B be a huge testbed (2000+ nodes) which can only communicate using ZigBee technology [38]; finally, let testbed C be an IPsec certified testbed, with multiple machines that can work as secure gateways for other networks. If these three testbeds were to form a federation, sharing thus their services (e.g. providing now secure resource reservation and communication), Steve would be able to test successfully his own "Distributed data sharing protocol over IPsec for heterogeneous networks" protocol, provided he designs the experiment correctly, using the tools provided by the federation.

In this example we note how testbed services and resources can be shared between testbeds; but this is not only what federations do. An important aspect of experimentation is linked to experimentation security and verification. As experiments may last long and have a high human cost, infrastructure certification and security is

fundamental to detect experiment failure causes and responsibilities. Therefore, the federation may choose to provide specific authentication, authorization and certification services to allow secure and controlled use of the infrastructures. These services can then be exploited locally by each testbed when independently running experiments.

The federation architecture is not fully standardized, but as of today it has been employed in diverse extensive European and International projects, with good results. Those which either are still in development or finalized, are:

1. *FIRE* [39], the current largest ICT European initiative, aims to provide a solid and extensive background for Future Internet Reasearch and Experimentation. It is properly a federation of projects. In this case, the proper role of "federation of heterogeneous testbeds" is lead by one of FIRE projects, *Fed4FIRE*.

2. *GENI* [40], stands at US almost like FIRE stands at EU, therefore it is (one of) the largest ICT projects in the US, and is an already developed and consolidated Federation.

3. *FanTaaStic* [41], is a rather recent project (2013) by the EIT ICT Labs, called "Fanning out Testbeds as a Service for the EIT ICT Labs" (FanTaaStic). The project explores best practices of testbed federation and brokering mechanisms, and offers federated testbeds on a commercial basis. Recent collaboration with FIRE brought substantial benefits for both these projects.

4. *PlanetLab* [42], is a global-size experimentation laboratory that is migrating towards federation. It is formed by testbeds located all throughout Europe, up to the northern Russia, as well as all throughout Americas, Australia and Asia, and once the federation will be completed, it will most likely become the largest existing federation.

As Federation of testbeds is a wide and new concept, a current, proper, *State of the Art* does not exist. Therefore, we will make an analysis on which are the different solutions to build a federation of testbeds, and we will overview the architectures of existing testbed federations.

Federation Types

As a starter, we need to discern between two main types of federations:

- **Homogeneous Federations**: the federated testbeds run the same software architecture, and work as different-scale replicas of a basic model. Example: *Planetlab*.

- **Heterogeneous Federations**: testbeds have different architectures and software running on them, and the federation needs to allow a seamless cooperation of them. Example: *Fed4FIRE, GENI*

The distinction of these two categories is fundamental, as there are substantial differences between them. First, an homogeneous federation is very similar to a scalable testbed: anytime a new facility is available, it is integrated within the federation by deploying the same software and system architecture on it. This makes communication and resource management easier, as all runs under the same environment. Backfires of such a system are that the architecture needs to be perfectly scalable, and that the core development team could be overloaded: the team would not only need to deal with the interconnection between these testbeds, but it would be also faced with all the technical faults that could happen within each deployment. Moreover if the architecture needs to be modified to adapt a specific testbed, then it needs to be re-validated over all the other testbeds to assure system consistency.

On the other hand, heterogeneous federations may be easier to manage under this point of view, as each testbed authority is responsible for the management of its own testbed. Yet, other issues pop up, as cooperation of heterogeneous testbeds requires solid and comprehensive interfaces, as well as a consistent and fault-tolerant interoperative architecture, which could result to be equally difficult.

Federation Architectures

In [43] the authors identified three types of Federation model:

1. **Distributed Model**: the Federation is defined only by distributed interfaces, and the physical implementation on each testbed makes up for the lack of a physical federative body. Examples: *Fed4FIRE* [44], *GENI*.

2. **Tight-coupled Model**: there is an independent and stand-alone Global Broker or Portal that serves all the testbeds, which is like the "physical" body of the Federation. A tested model for this kind of Federation is the plug-in model, where testbeds work as plugins to the main body of the Federation. Example: *Deterlab* [45].

3. **Loose-coupled Model**: there is a broker per each testbed, and all the brokers of federated testbeds are interconnected in a broker network. Example: *ProtoGENI* [46]

Chapter 3

SmartSantander Testbed

The information contained in this chapter is fundamental for the understanding of Chapter 5. We shall first detail the location background of the experience, the SmartSantander facility, together with the wider context it is set into, which is SmartSantander Project. We are mostly interested to the highest implementation level of the system, because our contribution deals with this protocol stack layer, therefore lower level considerations - on the hardware that the system uses - are subministered to the eager reader, when necessary, by means of the relative documentation.

SmartSantander is a project that aims at the creation of an experimental test facility for the research and experimentation of architectures, key enabling technologies, services and applications for the Internet of Things, in the context of a modern city. The project envisions the city-wide deployment of 20'000 wireless and wired sensors in the cities of Belgrade (RS), Guildford (GB), Lübeck (DE) and Santander (Cantabria, ES), with the largest test facilities being that in SmartSantander [47], counting by itself 12'000 sensors. The European Union financed the project with 8.7M EUR between 2008 and 2012, with the aim to "create the world's first and unique experimental research facility for applications in the field of Internet of Things" [48]. Further aims dealt with assessing the social acceptance of IoT, providing useful services to the population, as well as involving the citizens to use and contribute to the city digital infrastructures.

After the project was completed, the four facilities continued to be developed independently, while still being connected in name under SmartSantander project. Hence, as our work was made entirely and only within SmartSantander's Santader facility, we leave out of this thesis any information regarding the other facilities, which can be recovered from SmartSantander's main site [49]. Also, from now on, the project's main facility in Santander will be always referred to as "SmartSantander" or [SmS], for the sake of simplicity.

3.1 Innovation

As we stated in the introduction, the SmartSantander project aimed to develop a large scale IoT research facility that allowed for evaluation under realistic operational conditions, and as well provided an useful service to the city of Santander and its citizens.

The city of Santander is the capital of the autonomous community and historical region of Cantabria, in the northern part of Spain (see Fig. 3.1), and has a population of around 180.000 (178.465 in 2013). The city covers an area of around 9 x 4 km (east-west, north-south), it adjoins the coast, and it is an holiday site that surfers yearn for all the year long. Due to the abundant presence of tourists, actually, the city doubles its population during summer, between July and late September.

Fig. 3.1 The autonomous community and historical region of Cantabria is highlighted in the map.

As reports shows, the abundance of different scenarios and activities in the city provided the perfect background for a deep and rich experimentation site.

In truth, the aim was not brand new: previous takes on the concepts of Smart City and real-world Testbed were Oulu Smart City [50] and CitySense [51], which had already been deployed. These projects provided IoT devices for service enablement (Oulu) or experiments (CitySense), but they lacked adequate scale, heterogeneity, and support of realistic mobility scenarios. The most similar project to SmartSantander was WISEBED [52], a large and heterogeneous IoT laboratory deployment with support for testbed federations. Though, in truth, SmartSantander framework is build upon WISEBED, while significantly expanding its capabilities to a larger scale, wireless outdoor environment. In addition, a completely new service oriented layer was added together with the WSN experimentation layer.

As of now, SmartSantander is composed of around 3000 fixed, permanent sensors, 2000 RFID tags for augmented reality applications, and a net of over 7.000 dynamic mobile sensors, mostly smartphones advertising on-board sensor measurements through an ad-hoc application (*Pace of the City*). The sensors are spread throughout the ∼33 km^2 that the city spans, and are able to take multiple measurements, from light and humidity to accelerometers and magnetic fields. Mobile sensors are also set on top of the urban buses and taxis, providing both environmental sensor and vehicle related information. It is possible to track all the devices and see all the collected measurements by using the SmartSantander Maps API (see Fig. 3.2), or *SmartSantanderRA* mobile app.

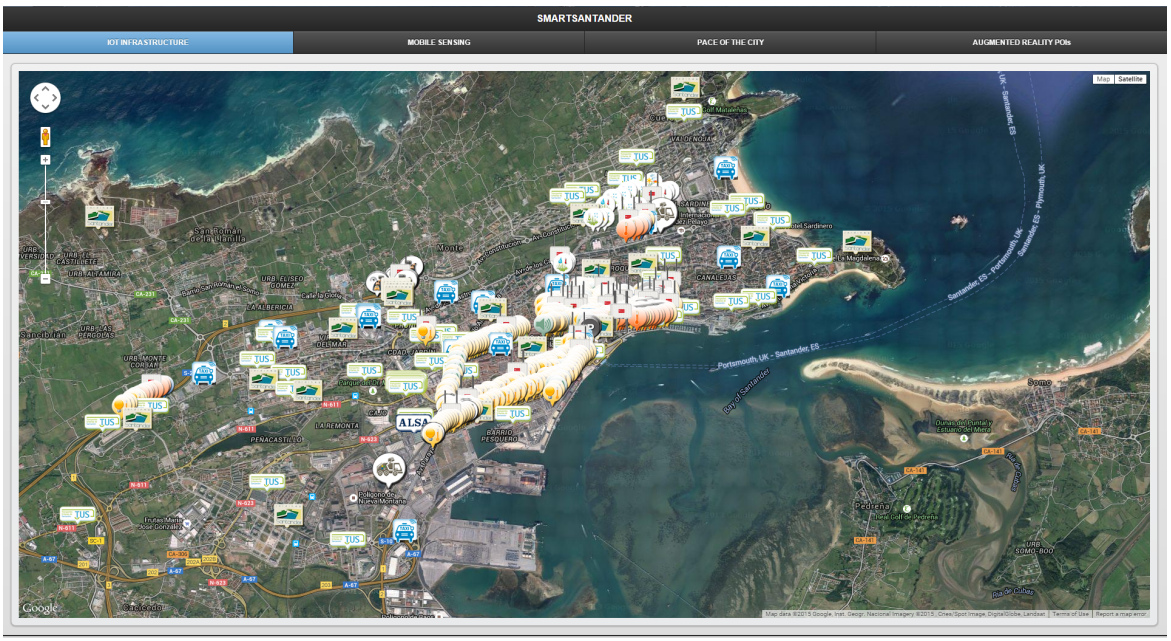


Fig. 3.2 Map of SmartSantander, showcasing the location of each and every sensor.

SmartSantander network is particularly heterogeneous, as the communication interfaces used by the terminals are multiple: fixed sensors employ mostly Digimesh and 802.15.4 PAN equivalent interfaces, gateways also includes those interfaces together with 802.11/1x, and smartphones use either 802.11a/b/g/n, GSM, EDGE, 3G, HSPA/+ and LTE cellular Internet.

It is important to note that SmartSantander revealed to be particularly ahead of its times: only this year, in 2015, that is 5 years after SmartSantander project started, Google planned to transform Carnegie Mellon University Campus into a "living laboratory" for IoT (or IoE) smart technologies [53], in the same fashion as SmartSantander.

3.2 SmartSantander Project

The project was financed by the EU under the FP7-ICT specific program (research theme 'Information and communication technologies'). The development ran from September 1, 2010, to November 30, 2013, and, while the EU financing was 6M EUR, the total cost of the installment was 8.465.355 EUR. Partners of the project were universities (from UK, Germany, Greece, Denmark, Spain and Australia), ICT companies, such as Telefonica I+D (Spain), Alcatel-Lucent S.P.A. (Italy/Spain) and Ericsson D.O.O. (Serbia), and of course the municipality of Santander [54].

The project was coordinated by Telefonica I+D, but the major part of R&D in Santander's facility was done by the Network Planning and Mobile Communications Laboratory [TLMAT], University of Cantabria [UC], under the guidance of Prof. Luis Muñoz Gutiérrez, head of TLMAT laboratory.

The project was scheduled to last for 36 month, and to host 2 open calls, one during M10, the other at M23. In truth the project lasted 38 months, due to the planning of a second generation architecture, but the calls dates were respected. Even though the project is closed, the framework is still active and provides useful services to the city. The migration to the 2nd architecture started once the project was finalized, it is now still in progress, and it is planned to be finalized by the end of the year. The new architecture is designed to improve management capabilities, efficiency and security, as well as to provide integration with Fed4FIRE, another EU project that we briefly cited in Chapter 1.

In the next section we show in minimal details the architectures of the first and second development cycle of SmartSantander, with a keen eye for their main aspects and the differences between the two cycles.

3.2.1 First Cycle Architecture

The first cycle architecture was modified multiple times during development, and all the versions are reported in WP1 of SmartSantander project [55]. The final version is structured as in Fig. 3.3.

SmartSantander operates and provides functionalities on three main layers: the Native Experimentation layer, the Service Experimentation layer, and the Control layer. In the first version it is difficult to clearly differentiate them, as they are mostly intertwined; although we may still make a rough separation. We refer to the various sections either by name or by direction (let "left or right' sides of the system" be the modules in the corresponding section of the schematics):

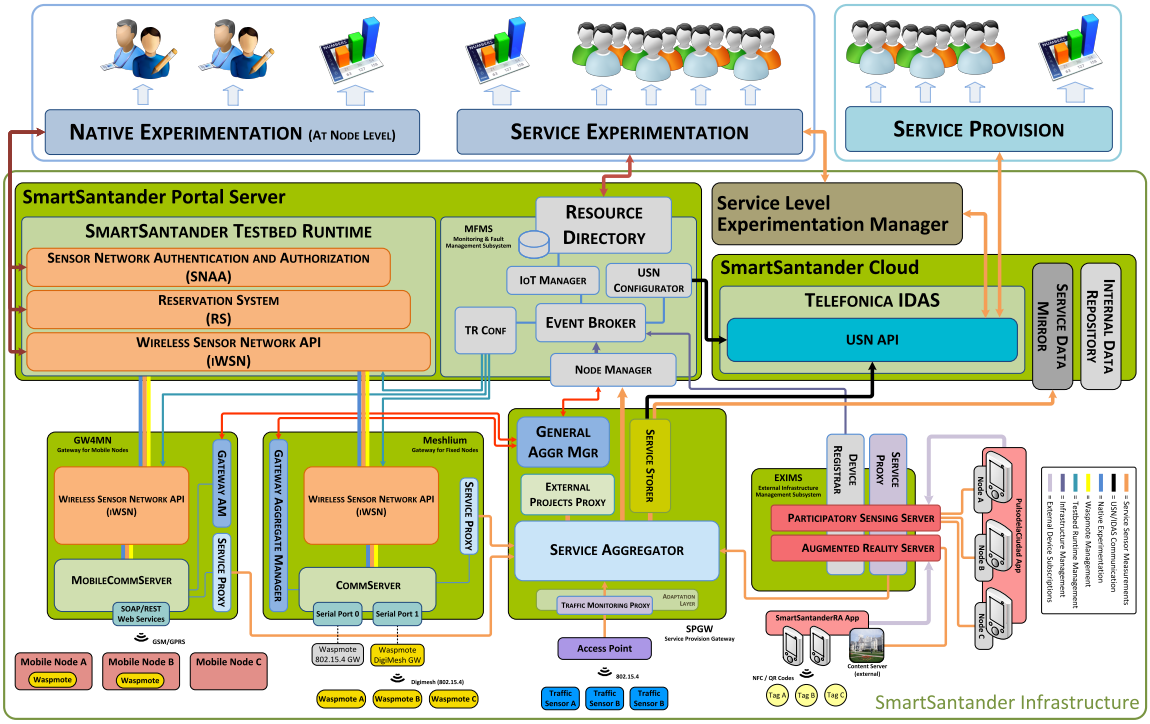


Fig. 3.3 SmartSantander First Cycle Architecture.

1. the *Native Experimentation* layer handles all the WSN oriented experiments that researchers schedule in the system, and is composed of the Testbed Runtime, together with the WSN API ("left" part of the system).

2. the *Service Experimentation* layer hosts the services offered to the citizen, from data storage to sensor information retrieval. This layer is composed by SmartSantander Cloud, the omonimous Service Layer Experimentation Manager, as well as the lower center - Service Aggregator and Service Storer - and right part of the system, i.e. the Participatory Sensing and Augmented Reality Servers.

3. the *Control* layer handles all the resources in the system, as well as the events that behappen, therefore it holds the Resource Directory, the Event Broker and all the related components.

The purpose of this thesis has little in common with these components, therefore we will not provide the complete description of these modules or services. Moreover, this architecture was consolidated by the end of the project in December 2013, but the development highlighted some deficiencies in the system and the necessity to implement other features in future. Therefore, the system architecture was modified, and the project entered the second development cycle.

Some of the reasons behind the migration to the second architecture follows here:

1. Scarce security in the service subsystem, which was neither flexible nor scalable. User authorization on the service layer was provided by firewalling ad-hoc interfaces.

2. There were no standardized protocols for external data injection, therefore ad-hoc adaptation layers were needed.

3. Multiple instances of each device information were present, and sometimes data was not consistent.

4. No unique resource identifier was used, as Unique Resource Number [URN] generators were not standardized neither global.

5. Lack of real-time notification system, and therefore continuous polling was required to access the sensor data, with useless bandwidth waste.

These problems and some of the users/experimenter requierements led the developers to design a new, second-cycle architecture, that solved the majority of these issues. The idea was to have an scalable system with a generic authentication/authorization architecture able to support user access in a more transparent way and, at the same time, provide an asynchronous pub/sub interface to the service oriented experimenters.

3.2.2 Second Cycle Architecture

Fig. 3.4 shows second cycle architecture. The pillars that sustain its design are:

1. Establish a clear separation of the roles inside the architecture
   (a) SmartSantander Core
   (b) Infrastructure or information providers.
   (c) Experimenters or observers, sometimes referred as service providers.
   (d) External entities that are not controlled by SmartSantander, like information repositories.

2. Homogenize and unify as best as possible the access interfaces to core and measurements, fleeing from ad-hoc interface development.

3. Simplify SmartSantander city infrastructure management.

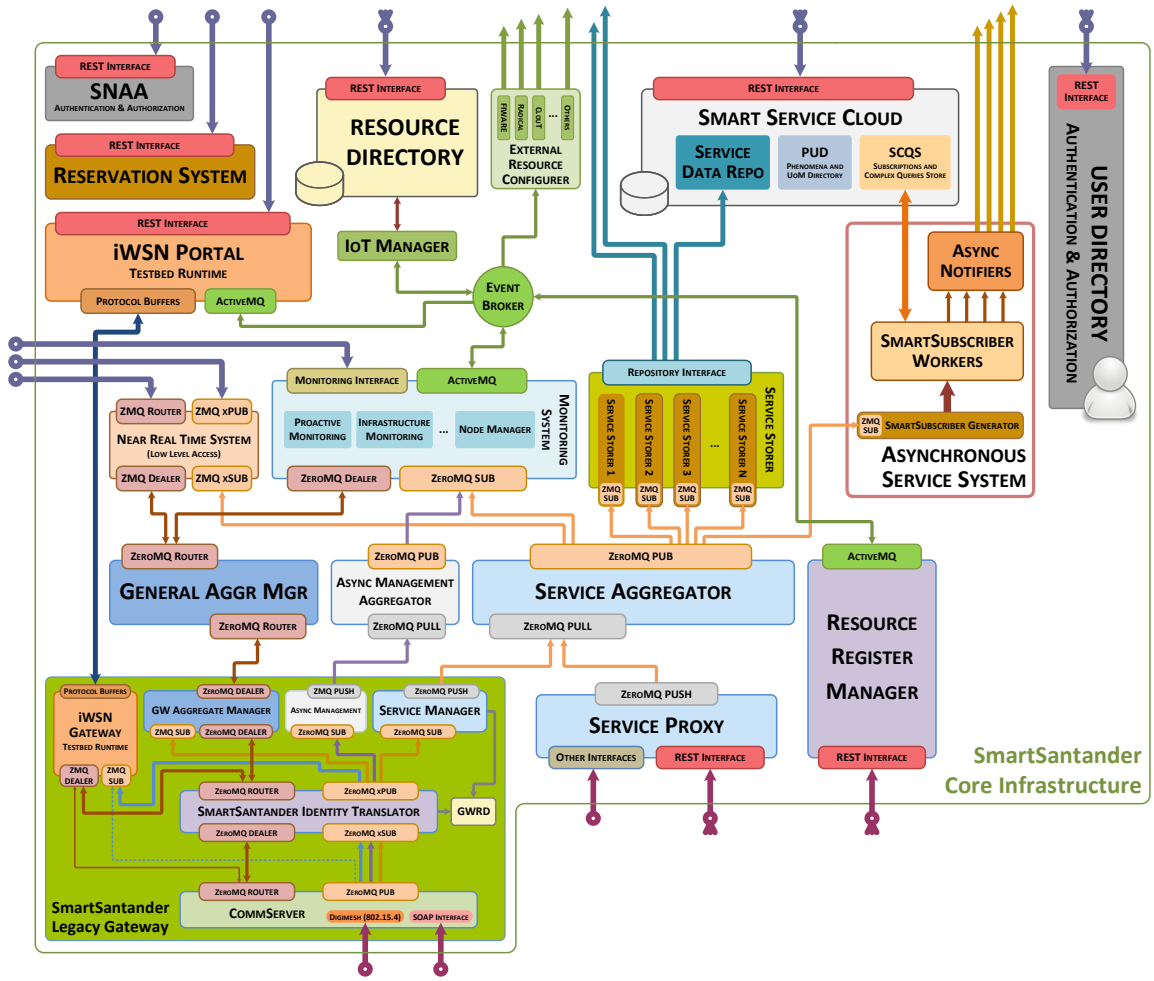


Fig. 3.4 SmartSantander Second Cycle architecture.

4. Identify and recognize who is using the system at any time, while at the same time providing a layer of security to the public interfaces of SmartSantander.

5. Introduce a real-time notification system (the asynchronous service layer).

Therefore the second architecture simplifies and regularizes the system architecture, but nonetheless its complexity remains high. Many deliverables would be required to fully describe SmartSantander v2. However, those are not available in common place because most of those developments were done as part of the involment in different projects.

3.3 Service Experimentation Layer [SEL]

Our concern lies mostly in the Service Experimentation Layer [SEL], therefore we now analyse in details its structure; then, we introduce one of the most important addition that the second architecture brought, i.e. the Asynchronous Notification Service Layer [ANSEL], which is the component we contributed to develop.

3.3.1 Underlying Concepts

Observation and Measurement Model

SmartSantander is a large and heterogeneous platform, especially in terms of devices. This made the development team apt for the standardization of a flexible data format, that was generic enough to allow the inclusion of new, diverse devices in the future.

All the information generated by SmartSantander *resources* is stored inside the system as single *measurements* or in batch as *observations*. The system stores both the data types in JSON format (see Listing 3.1 and Listing 3.2), and handles them like chunks of information. Moreover, once an observation or a measurement is inserted into the system, it cannot be modified in any way, nor even deleted.

Listing 3.1: Measurement JSON format.

```
{
    "urn" : "urn:x-iot:smartsantander:u7jcfa:fixed:10013",
    "timestamp":"2014-04-30T11:41:01.123+02:00",
    "location":
    {
            "coordinates": [-3.8643275, 43.4664959],
            "type":"Point"
    },
    "phenomenon":"temperature:ambient",
    "value":23.01,
    "uom":"degreeCelsius"
}
```

Listing 3.2: Observation JSON format.

```
{
    "urn" : "urn:x-iot:smartsantander:u7jcfa:fixed:10013",
    "timestamp":"2014-04-30T11:41:01.123+02:00",
    "location":
    {
```

```
6               "coordinates": [-3.8643275, 43.4664959],
7               "type":"Point"
8       },
9       "measurements":
10      [
11          {
12              "phenomenon":"temperature:ambient",
13              "value":23.01,
14              "uom":"degreeCelsius"
15          }, {
16              "phenomenon":"relativeHumidity",
17              "value":56,
18              "uom":"percent"
19          }
20      ]
21  }
```

If we compare these data formats to the data packets that are exchanged in a typical communication system, we can split the data format in *header* and *payload*. The **header** contains three fields, *URN, timestamp, location*, which are provided by and linked to the specific resource that generated this information. In details:

- the URN is the Unique Resource Number that was assigned to the resource at the time of its registration into SmartSantander framework, and can be split in multiple sub-domains as like IP addresses are.
- the timestamp employs ISO 8601 combined format, i.e. per example "2015-09-30T01:28:23+00:00".
- the location must be formatted in GeoJSON format, as this allows to include virtual sensors inside the system (e.g. we can identify the sensors in a specific area, let it be a circle of radius 500m, with a single virtual sensor).

The **payload** contains the actual information. In case this field contains multiple measurements, therefore if this is an observation, the keyword "measurements" must be included in this field. Per each measurements, three parameters are signifying:

- the specific measurement *phenomenon*, from those included in the official SmartSantander phenomenon directory.
- the specific Unit of Measurement [*UoM*], that must be listed under SmartSantander's UoM directory.

- the specific measurement value, either raw or pre-processed, as specified by the UoM definition.

Subscription and Complex Query

One of the concern of big IoT platforms, and SmartSantander is not an exception, is how to offer the experimenter a proper way to filter the information to fit their requirements without heavily impacting usability. To satisfy all level of users, though, a trade off between simplicity and usability must be reached.

The optimal trade off was found in the use of REST APIs, that allowed both neophyte users to navigate the resources using a web browser and advanced users to fine-grain design their requests. In addition, the descriptive model is designed to answer the "what-where-when" questions: *what* information does the experimenter need? From which location, i.e. *where*, should the data be collected? Which time period does the experimenter need information on, i.e. *when* were the data generated?.

In this context, Subscriptions and Complex Queries are the fundamental medium with which users can express their requests of data to SmartSantander framework. The two formats share many similarities, and the distinction is made solely on the basis of their purpose: complex queries are linked to the Synchronous part of SEL, while subscriptions are used exclusively by the Asynchronous part of SEL. In REST terms, each subscription or complex query generates new persistent resources inside the system, which can be managed by its owner and formatted as JSON objects. Their description follows the "what-where-when" model we introduced beforehand.
The **<what> criterion** provides filtering capability for three different levels:

1. Among all different measurements, select just those measuring phenomenon P.

2. Among all measurements of phenomenon P, select only those expressed with unit of measurement U.

3. Among all measurements of phenomenon P with unit of measurement U, select those with value V.

Valid values of P and U can be extracted from the SmartSantander PUD service (Phenomenon and UoM Directory), while different operations are supported to specify the desired valid values. The <what> structure is in Listing 3.3. Subsciptions to SmartSantander can be made through the IoT API, by using a RESTful client such as Postman [56] to send a POST request.

Listing 3.3: Complex query <what> criterion.

```
{
    "phenomenon" : <phenomenon>,
    "filter" : {
        "uom" : <uom>,
        "value" : {
            // AND is applied

            "_eq" : <string> or number,
            "_ne" : <string> or number,
            "_in" : [<string> or number],
            "_nin" : [<string> or number],

            // operators for strings
            "_regexp" : <regexp>

            // operators for numbers
            "_gte" : <number>,
            "_gt" : <number>,
            "_lte" : <number>,
            "_lt" : <number>
        }
    }
}
```

The **<where> criterion** allows a user to specify an observation/measurement source. This origin can be expressed in two different ways: either as a set of resource URNs or as a geographical area described in GeoJSON format. See Listing 3.4 for additional <where> format details.

Listing 3.4: Complex query <where> criterion.

```
{
    "phenomenon" : <phenomenon>,
    "filter" : {
        "uom" : <uom>,
        "value" : {
            // AND is applied

            "_eq" : <string> or number,
            "_ne" : <string> or number,
            "_in" : [<string> or number],
```

```
11              "_nin" : [<string> or number],
12
13              // operators for strings
14              "_regexp" : <regexp>
15
16              // operators for numbers
17              "_gte" : <number>,
18              "_gt" : <number>,
19              "_lte" : <number>,
20              "_lt" : <number>
21          }
22      }
23  }
```

The **<when> criterion** is used solely when an experimenter wants to restrict the period of the data he is retrieving. This criterion is mostly used when accessing historical data, but can also be used to cover some special use cases. Time constrains following ISO 8601 are supported, but if a fixed quantity is inserted, it indicates the number of last observations (or measurements) that are shown.

3.3.2 Low-level Details

Fig. 3.5 depicts the SEL in its entire high level structure. From this, we can differentiate several subsystems:

- Security sub-system, which is responsible for the authentication, authorization and management of the different users in the system. Its main component is the User Directory.

- Resource sub-system, which handles resource description information. Its main components are the Resource Register Manager, which is the infrastructure provider entry point; the event broker which acts as the central resource oriented data exchange hub; and the Resource Directory which is the actual data repository offering storage capabilities. Finally, there is another component, known as IoT Manager, which enables interoperability between different domains.

- Glossary sub-system, essentially composed by the Phenomena and UoM directory (PUD). Its mission is the definition of the accepted vocabulary associated to the different sensor parameters that SmartSantander SEL platform supports.

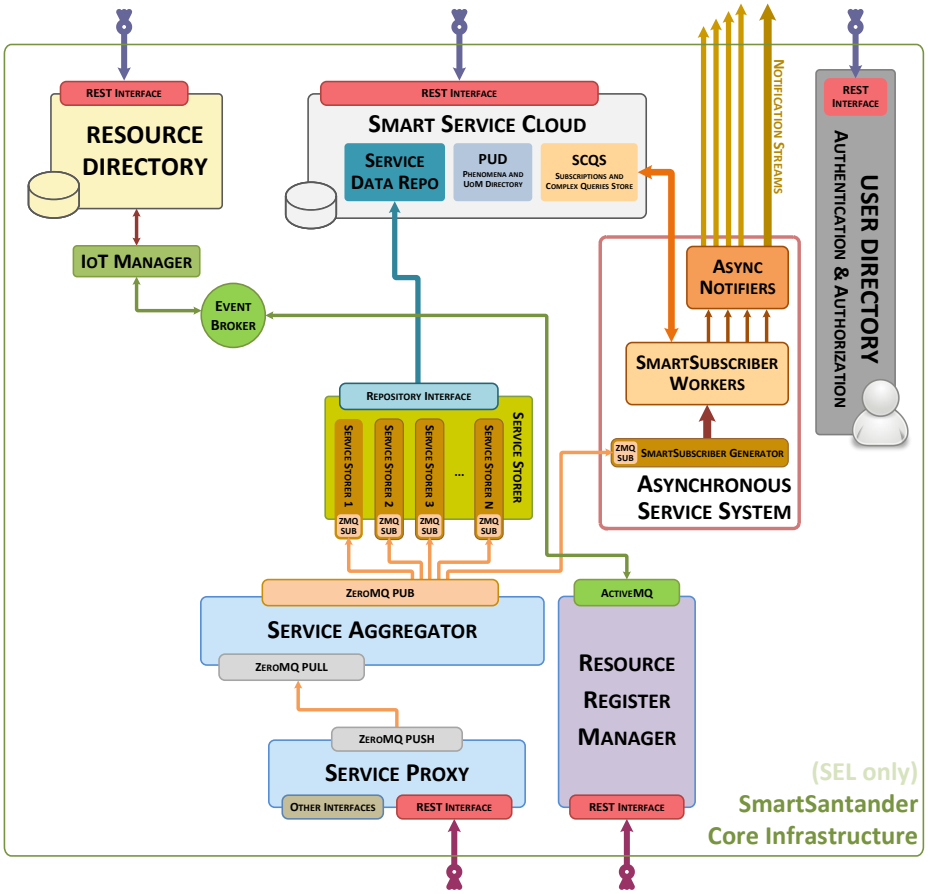


Fig. 3.5 SmartSantander Service Experimentation Layer [SEL].

- Information sub-system, which is in charge of adapting and storing observations to be used both in the synchronous and asynchronous service system. As the focus of this article is to dig into this sub-system, next subsections include detailed aspects of its different components.

All those sub-systems are exposed to the outside world through a REST API, which is known as SmartSantander IoT API, and which will be described in the next section of this chapter.

Service Proxy

Service Proxy is the entry point for any infrastructure provider that desires to insert any kind of sensor data in SmartSantander SEL platform. Information injection is exposed as part of the IoT API, and can be done issuing a POST request to a specific endpoint (e.g. http://api.smartsantander.eu) including one or more observations. The

proxy filters out any push from hosts without proper authorization or including a capability (phenomenon/UoM couple) not listed in the resource description, as well as it patches any measurement or observation lacking a significant field inside the header (e.g. without timestamp or location).

Service Proxy can be easily scaled by replicating its structure on another dedicated machine and properly configuring the load balancing. It uses a ZeroMQ push/pull service to send the observations to the next ring of the chain, which is the Service Aggregator.

Service Aggregator

Service Aggregator serves as broker of all the observations generated in SmartSantander. Technically, it works as a router, which publishes all the incoming observations on the different topics employing a ZeroMQ pub/sub mechanism. It doesn't process observations in any way, it just wraps them in a packet, assigns a serial number to avoid duplication on consumers subscribed to different topics, and forwards them.

The Service Aggregator forwards messages to both the synchronous and asynchronous part of SmartSantander SEL.

3.3.3 Synchronous Service System

Service Storer

The component is composed by a set of different modules, each of them in charge of forwarding service data to a specific internal or external backend repository. The module in charge of storing observations on the SmartSantander Service Data Repository is known as SDR Storer. FIWare IDAS/DCA platform is an example of an external repository receiving information from a Service Storer module.

Service Data Repository

The Service Data Repository (SDR) is, as it name suggests, the storage element for all the service information generated inside SmartSantander. The SDR Storer continuously writes data inside the SDR, hence its architecture allows for efficient concurrent read/write operations. Moreover data inside SDR is compressed to save space, and replication allows for higher availability.

Complex Query Store

This component is the data repository to hold complex queries, and it is part of the Subscriptions and Complex Queries Store [SCQS]. Read and Write operations are performed through the SmartSantander IoT API.

3.3.4 Asynchronous Notification Service Layer [ANSEL]

SmartSubscriber Generator

The function of this component is to gather all the observations from the Service Aggregator in a similar way as SDR Storer does, and generate background tasks to be executed by the different SmartSubscriber Workers. Those tasks are stored using RQ queues.

Those workers are in charge of evaluating all the active subscriptions in the User Subscriptions Cache (USC) against a single observation. In this sense, after applying authorization rules, SmartSubscriber workers push the resulting notifications on the different notification channels. There is one different notification channel for each supported notification technology (e.g: RestHooks, OML, websockets...). Both USC and Notification channels are based on a Redis deployment.

Async Notifiers

Async Notifiers are the modules emitting the notifications to the end users based on the subscription target. Each notifier handles a different technology, and they can be horizontally scaled by replication if needed. Async notifiers can use a loopback mechanism to inform the system if a notification fails because the other endpoint is not listening and disable the subscription.

Subscription Store

This component is the data repository which holds all the subscriptions, and it is part of the Subscriptions and Complex Queries Store (SCQS). Read and Write operations are performed through the SmartSantander IoT API. Active subscriptions are cached in the USC to be consumed by the SmartSubscriber Workers. From a technical perspective, it follows the same approach as the complex queries store.

3.3.5 Examples

Listing 3.5: Example of *JSON* ANSEL subscription submitted to the system.

```
{
    "target": {
        "technology": "oml",
        "parameters": {
            "address": "192.168.100.134",
            "port": "3003",
            "experiment": "omltest",
            "application": "omltestapp"
        }
    },
    "query": {
        "what": {
            "format": "measurement",
            "_allOf": [
                {
                    "phenomenon": "temperature:ambient",
                    "filter": {
                        "uom": "degreeCelsius",
                        "value": {
                            "_gt": 5
                        }
                    }
                }
            ]
        },
        "where":{
            "_allOf": [
                {
                    "area": {
                        "type": "Circle",
                        "coordinates": [
                            43.462403,
                            -3.810011
                        ],
                        "radius": 0.5,
                        "properties": {
                            "radius_units": "km"
                        }
                    }
                }
```

```
41              ]
42          }
43      }
44  }
```

Listing 3.6: Example of *JSON* ANSEL simple measurement.

```
1  {
2      "urn":"urn:x-iot:smartsantander:u7jcfa:fixed:10013",
3      "timestamp":"2014-04-30T11:41:01.123+02:00",
4      "location":
5      {
6          "coordinates":[-3.8643275,43.4664959],
7          "type":"Point"
8      },
9      "phenomenon":"temperature:ambient",
10     "value":23.01,
11     "uom":"degreeCelsius"
12 }
```

Listing 3.7: Example of *JSON* ANSEL simple observation.

```
1  {
2      "urn":"urn:x-iot:smartsantander:u7jcfa:fixed:10013",
3      "timestamp":"2014-04-30T11:41:01.123+02:00",
4      "location":
5      {
6          "coordinates":[-3.8643275,43.4664959],
7          "type":"Point"
8      },
9      "measurements":
10     [
11         {
12             "phenomenon":"temperature:ambient",
13             "value":23.01,
14             "uom":"degreeCelsius"
15         }, {
16             "phenomenon": "relativeHumidity",
17             "value":56,
18             "uom":"percent"
19         }
20     ]
21 }
```

Chapter 4

Fed4FIRE Federation

In this section we introduce the structure of the Fed4FIRE federation, as well as the interfaces and protocols our contribution aimed to implement. As a starter, we briefly describe the large scale project behind Fed4FIRE, i.e. the Future Internet Research and Experimentation Initiative [FIRE], then follow up with Fed4FIRE Project structure and details. Later on, at the beginning of Chapter 5 we shall cover how what we described in Chapter 3 and here in Chapter 4 are linked together.

4.1 FIRE Initiative

Fig. 4.1 FIRE logo.

FIRE is one of the largest FP7 ICT group of projects that the EU finances. The budget for the first wave, in 2008, was 40M EUR, which was later extended by a second wave in 2010 (50M EUR), a third in 2011, and a fourth wave under FP7 ICT Call 8 (25M EUR), in 2012. Eventually, FIRE participated as well to FP7 ICT Call 10 in 2013, receiving additional 19M EUR funding, for a total of 144M EUR of financing in 5 years

time [57]. Currently, FIRE's offering includes 12 facility projects: CONFINE, CREW, Fed4FIRE, FELIX-EU, FESTIVAL, FIESTA, FLEX, MONROE, ORGANICITY, RAWFIE, SUNRISE, WISHFUL, plus one open access complete project, OneLab. Additional informations can be found in [58].

The aim of FIRE is to provide an uniform, self-sustained, large-scale experimentation site, over which researchers, companies and universities may experiment with multiple types of scalable networks and multiple devices topologies. Hence, the main aim of FIRE is to support users, so that they don't need to build their own experimentation site, which could be too resource-consuming for most of them, and may be hard to build on a large scale. Moreover, most architectures resemble one another, therefore multiple users that implement the exact same architecture would waste their time and resources. FIRE's idea is to block users from reinventing the wheel, by providing an efficient, consolidated framework.

Though, as we amply pointed out in Chapter 1, sustainability and visibility of the testbed and the federation itself is another big concern. These aims are connected, though, as valid support enhances visibility. That visibility allows higher support from the community and the institutions, and finally higher sustainability leaves space for improved support to the end user. This virtuous cycle is exactly what FIRE strives for.

The facility project we are interested in is Fed4FIRE, which is FIRE's own take into testbed federations, therefore we leave out of the matter any detailed information regarding FIRE, and point the eager reader to the official documentation [59].

4.2 Project Overview

Fig. 4.2 Fed4FIRE logo.

Fed4FIRE stands for "Federation for FIRE" and is the federative project that aims to link most of FIRE resources and obtain all the benefits we listed in Chapter 2.

Following the official site description, "Fed4FIRE is an Integrating Project under the European Union's Seventh Framework Program (FP7) addressing the work program topic Future Internet Research and Experimentation. It started the 1^{st} October 2012 and will run for 48 months, until 30 September 2016." The EU contributions to the project were 7.747.997 EUR, while the total cost of the project was 11.067.773 EUR [60].

Currently, the project is running at full throttle, with multiple conferences being held/planned in the following months [61]. Scholar documentation is scarce, but conference proceedings and public deliverables provide enough material for a detailed description of the project.

4.2.1 Federative Architecture

Fed4FIRE architecture is composed of multiple parts, but is mainly split into three domains: the Experimenter side, the Federation side, and the Testbed side. Each side has its own role, and should provide specific services and APIs. The Experimenter and Testbed side are external to the Federation, and should adapt to the Federator interfaces and employ its APIs, while the Federator should suffice to the needs of the other two parties with a sufficiently rich portfolio of APIs.

Fig. 4.3 is the empty template that highlights this separation of roles. The three sides are in the three columns, while the rows represent the typology of the service. As it can be seen, there is strict relationship between the service type and the role that should provide the service. Note that the fourth column states what proprietary tools may be implemented from entities external to the federation.

4.2.2 Experiment Lifecycle

Fed4FIRE provides a pre-defined experiment lifecycle, which gives directions on which tools the experimenter can employ.

Function	Description
[SFA] Resource discovery (1)	Discovery of the facility resources,(e.g. uniform,resource description model).

[SFA] Resource requirements (2)	Specification of the resources required during the experiment, including compute, network, storage and software libraries. E.g., 5 compute nodes, 100Mbps network links, specific network topology, 1 TB storage node, 1 IMS server, 5 measurement agents.
[SFA] Resource reservation (3)	How can you reserve the resources? Examples: (a) no hard reservation or best-effort (use of a calendar that is loosely linked to the facility), (b) hard reservation (once reserved, you have guaranteed resource availability). Other options: one should reserve sufficient time in advance or one can do instant reservations.
[SFA] Resource provisioning (4)	**Direct**: Instantiation of specific resources directly (responsibility of the experimenter to select individual resources). **Orchestrated**: Instantiation of resources through a functional component, orchestrating resource provisioning (e.g., OpenNebula or PII orchestration engine) to decide which resources fit best with the experimenter's requirements. E.g., the experimenter requests 10 dual-core machines with video screens and 5 temperature sensors.
[OMFv6] Experiment control (5)	Control of the experimentation facility resources and experimenter scripts during experiment execution (e.g. via the OMF experiment controller orSSH). This could be predefined interactions and commands to be executed on resources (events at startup or during experiment workflow). Examples are: startup or shutdown of compute nodes, change in wireless transmission frequency, instantiation of software components during the experiment and breaking a link at a certain time in the experiment. Realtime interactions that depend on unpredictable events during the execution of the experiment are also considered.

[OML] Monitoring (6)	**Resources**: Monitoring of the infrastructure health and resource usage (e.g., CPU, RAM, network, experiment status and availability of nodes). **Experiment**: Monitoring of user-defined experimentation metrics (e.g., service metrics, application parameters) of the system under test. Examples include: number of simultaneous download sessions, number of refused service requests, packet loss and spectrum analysis.
Permanent storage (data experiment descriptor) (7)	Storage of the experiment descriptor or experimentation data beyond the experiment lifetime (e.g., disk images and NFS)

Table 4.1 Experiment Lifecycle

Now we tackle the technologies employed by the federation. For each technology, we also provide the way in which it is integrated into Fed4FIRE.

4.3 Federative Technology

Here in this section we overview those technologies that F4F implements, and that are useful to the aims of this thesis. Specifically, we detail two technologies, that deal with measurement collection and distribution (OML) and resource reservation, provisioning and management (SFA). The documentation for all those technologies not tackled in this section can be found on the official F4F website [62].

4.3.1 Open Measurement Library [OML]

OML is a tool developed together with the ORBIT framework by NICTA [63], Australia's largest ICT research organization. The official documentation is composed by an online website [64] - which also contains a wiki - and four main publications [65][66][67][68].

The tool provides an easy interface to data and measurement collection, and allows any experimenter to specify both the kind of data it requires and the physical/digital location where to store the collected data. There are other more common alternatives to this protocol, like Rest Hooks, WebSockets, ZeroMQ, AMQP or similiar, but the interest on OML has lately risen, as it has a good number of aces up its sleeve.

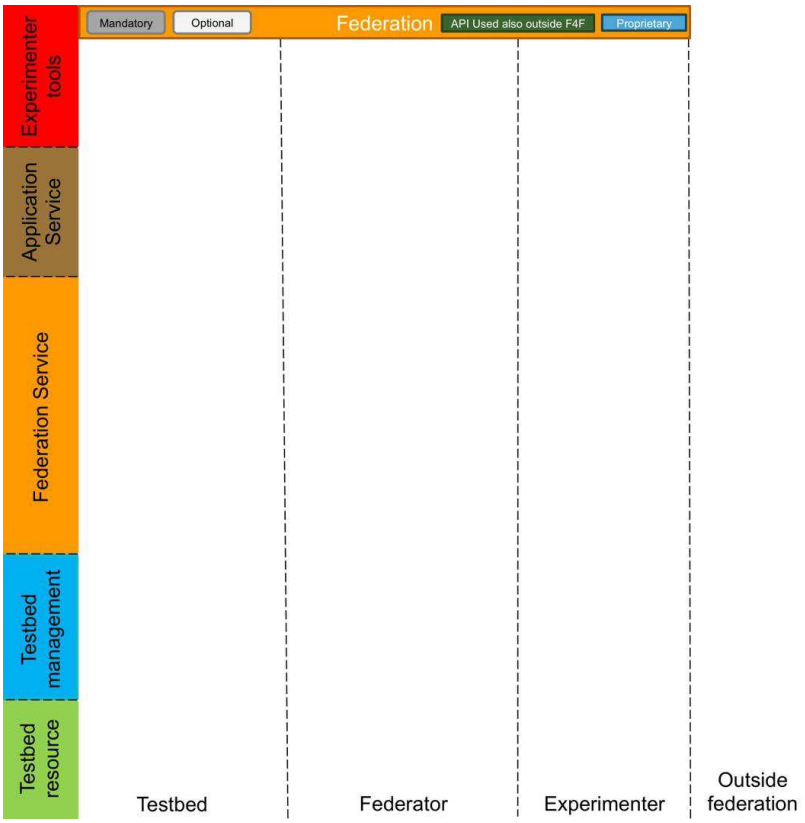


Fig. 4.3 Fed4FIRE architecture subdivision.

Fig. 4.4 OML logo.

The aim of NICTA was to patch some of the issues that may arise in a mobile sensor network when dealing with measurement collection. Lack of control network, sporadic connectivity, single connection interfaces or even bandwidth overload, may affect heavily or even compromise data collection from these devices. Moreover, most of the time there is not a standardized location nor way where to collect measurement: data could be sent to different collection sites based on the type, or different databases could be employed to store all the data types available. This is good practice for data separation and management, but abundance of sensors on modern devices render this

approach suboptimal, due to data location fragmentation, or, worst, high overhead to manage the different locations and the different measurement types.

OML comes at hand exactly in this cases, as it mainly solves the latter: it allows to store all the experiment data in a single place, and minimizes the collection overhead. The trick to this is that OML automates most of the functions, working as a separate application-level connection. OML respects the client-server model, therefore applications are split into an OML server and one or multiple OML clients. Each client keeps a connection with the OML server, and forwards measurement to that single server automatically, which cares later about storing data in multiple tables inside an SQL or NoSQL database (currently SQLite3 and PostgreSQL are fully supported). This automation makes OML an excellent tool for the end user, which required to provide only a handful of details to use OML services.

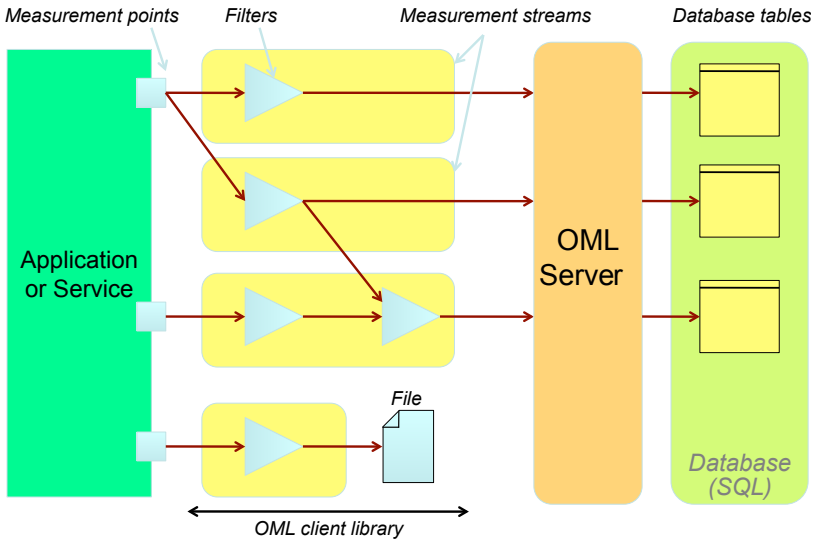


Fig. 4.5 OML data delivery scheme.

The OML delivery scheme can be seen in Fig. 4.5. To reduce the network overhead, OML introduces the concept of "measurement point", which are logical channels that accept a single format of measurement. Additional filters may be employed to provide further separation between measurement points, based on specific contents, but these are optional. The importance is that, during the set-up of the server, or at *any moment during runtime*, the client is able to cast new measure points on the server, which are referred to by name. Then, once the client requires to send different kind of measurements, it simply keeps the same connection active with the server and **injects** the measurements on the different measure points. As a mere data storage handler, OML is also very flexible, as measurement points format can be specified by the client

accordingly to standard digital data types (*byte, integer, long integer, float, double, string, etc*). Formally, OML measures "anything" that can be stored into digital data, for it is uses the basic digital data types to store measurements.

This versatility results significantly useful also in the other way round, using a single client and multiple servers. Let us analyze the particular case of an experiment collection site that employs a subscription system for specific measurements, which are injected using the OML framework. There, the collection site can employ a single OML client, and the subscribers that yearn to collect measurements have each one its own OML server, already configured with the data format, and its own database where to store data. In this scenario, OML works as a charm, as the client only needs to be notified of 1) the measurement format and 2) the OML server's address of each subscriber to that measurement format. Hence, it just needs to multicast the measurement to the respective servers, as the servers have already been configured to receive that kind of measurement.

This approach is similar to our contribution, with the difference that the collection site - indeed SmartSantander - allows for 1) great subscription granularity - it is possible to subscribe to single measurements, single groups of devices or even single devices -, and more than that 2) provides different level of access to each user, therefore forming multicast sets keeping track of both user privileges, single device subscriptions and measurement format would waste all the utility gained by multicast. Therefore our contribution was to implement a parallel unicast system, which employs multiple OML clients to unicast the measurements to the subscribers OML servers. The parallel application was devised using an hybrid threaded and asynchronous system, where each OML client is a separate thread, which is kept alive depending on the bandwidth requirement and subscriber diversity. Then, both the notification system and the clients employ an asynchronous pipeline to perform parallel operations, be it notifying the clients or sending measurements to multiple subscribers while waiting for each transmission to be completed.

From the above considerations, it is easy to notice that the tool developed by NICTA hits the "simplicity/power" sweet spot, providing easy to implement tools that allow for high flexibility and - as well - high performances [68]. The built-in library is lightweight, flexible and fast, as it is written in C code, and implementation in other languages are available (an official release for Python is available - *OML4Py* - as well as for ruby - *OML4R* - while third party tools are available for Java/Javascript - *OML4J/JS*). In addition, OML is a stand-alone application, and can be integrated with any other existing architecture. As a technical example representative of these

features, the integration of OML with our hybrid asynchronous and threaded framework employed Python OML libraries.

Fed4FIRE supports OML and OML-enabled testbeds. The architecture for OML support is in Fig. 4.6, where we can see OML-related modules as well as backend servers.

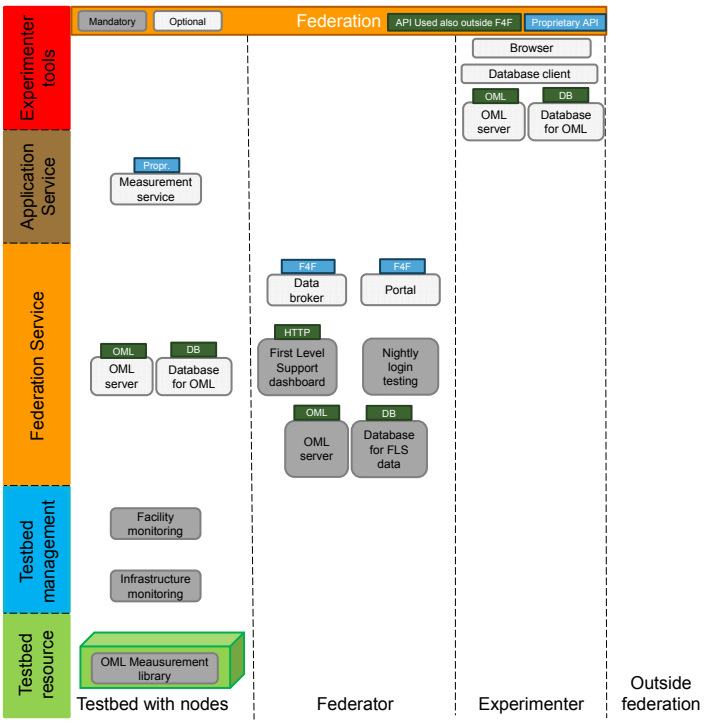


Fig. 4.6 Fed4FIRE architecture subdivision, monitoring and measurement architecture.

In the guide [69] provided on Fed4FIRE Documentation, which is based on a tutorial by NICTA, it is shown how to create an OML Wrapper (ruby script) around an existing binary application (iPerf). Further information for those interested into using OML together with Fed4FIRE can be found there.

4.3.2 Slice-based Federation Architecture [SFA]

The Slice-based Federation Architecture [SFA] is the final form of what grew from the resource control framework employed by the GENI [40] initiative, thanks to the experiences made with PlanetLab [42], Emulab [70] and VINI [71].

It is now a stand-alone architecture, that is not sponsored whatsoever by GENI or other testbed federations, and is designed to support the aforementioned testbeds as well as a broader range of designs. This statement resides on the main document we use as

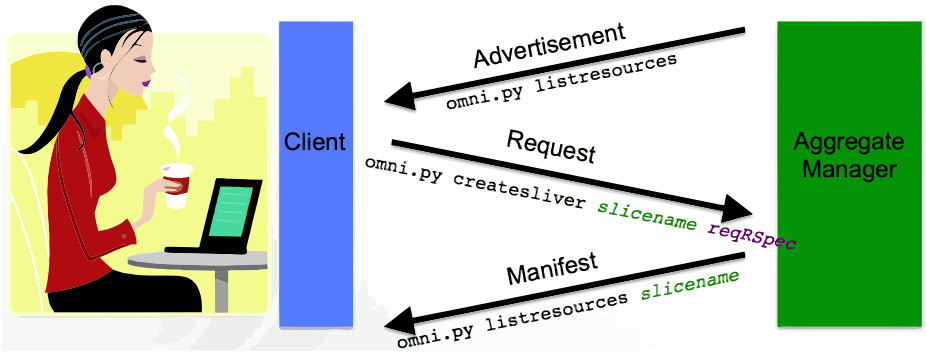


Fig. 4.7 SFA example.

reference, which is GENI SFA AM API v3 Documentation [72]. A slightly less detailed but more practical explanation of the architecture can be found in CONFINE [73] Wiki [74], provided one omits those informations that are solely related to CONFINE SFA implementation.

Formally, SFA provides a minimal interface to enable a federation of slice-based network substrates - testbeds with different technologies and belonging to different administrators - to inter-operate, while granting the control of the resources to their owners. This allows researchers to combine resources available in different testbeds, increasing the scale and diversity of their experiments.

SFA is based on a set of high-level concepts that define the actors and the resources that interact on the testbed, as well as defining an architecture with its interfaces and main data types to facilitate the federation of testbeds.

We start the overview of these concepts in the other way round with respect to the reference documentation. First, we define how the resources are abstracted into what we are going to call *abstract resources*:

- **Components**, the minimal aggregation of physical resources that can be managed (e.g. in SmartSantander: a single sensor). A collection of components is sometimes then considered as a single **aggregate**.

- **Slivers**, which are the portion of such resources let to the researchers (e.g. in SmartSantander: the complete experimentation layer resources, while the service and control ones are kept private).

- **Slices**, whose purpose depends on the perspective in play. From an experimenter point of view, slices are substrate-wide network of computing and communication slivers, upon which experiments or network services may run. Slices are requested by experimenters, but assigned by operators. Therefore, from an operator's

perspective, they are the primary abstraction for accounting and accountability, as slices *consume* the resources they are granted. SFA defines three stages in a slice life-cycle:

(i) *register*: at this point the slice exists only in name;

(ii) *instantiate*: the slice is instantiated in the required components, being granted of a set of resources; and

(iii) *activate*: the slice becomes active and runs code on behalf of the researcher.

With the exception of slivers, each abstract resource requires then the presence of a dedicated manager (e.g. **Component Manager [CM] or Aggregate Manager [AM]** for components/aggregates and **Slice Management [SM]** for slices), that exports a well-defined remotely accessible interface, and defines the operations available to user-experiments. CM usually runs on components, while if the component can't host a CM or if the abstraction requires an AM, then they are usually proxy CM/AM and are hosted on separate machines.

Then, the managers and the resources need to be registered to a certain directory to take visibility. These directories are managed by relative *authorities*, that file under the other main concept of SFA, that of *actors*:

- **Management Authority [MA]**, which is responsible for the maintenance of physical components.
- **Slice Authority [SA]**, which manages slices, naming and registering them while giving users access and control to their slices.
- **Users**, who are people playing one or more role in the facility.

Every entity we have defined must then be addressable:

- **Global identifier [GID]** is an identifier assigned to components, slices, services and every principal participating in the system. Specifically a GID is a certificate that binds together a public key, a UUID and a period of time during while the GID is valid.
- **Human readable name [HRN]** for an object identifies the sequence of authorities that are responsible for the object. Authorities have a level-based hierarchy, and HRNs are formed as a list of the authority name, in descending order of level, followed by the resource own name. A simple dot provides separation between names, like in the following: *top-level-auth.sub-level-auth.bottom-level-auth.name*.

- **Registry** maps HRNs to GIDs, as well as records others domain-specific information about the corresponding object, such as the URI at which the object's manager can be reached, an IP or hardware address for the machine on which the object is implemented, the name and postal address of the organization that hosts the object, and so on.

In addition to GIDs, the SFA defines four key data types:

- **Resource specification [RSpec]** is used to describe the resources on the system. Therefore, for a component it will describe the set of resources it possesses and constraints and dependencies on the allocation of those resources; whereas for slices it will describe the set of slivers it consists of and their characteristics. The RSpec is an XML document, and its definition depends on the testbed, since every testbed will have its own type of resources and requirements. An example of a working RSpec for GENI framework is in Listing 4.1.

Listing 4.1: RSpec example to select a single random node from a testbed.

```
<?xml version="1.0" encoding="UTF-8"?>
<rspec xmlns="http://www.protogeni.net/resources/rspec/2"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.protogeni.net/resources/
          rspec/2
                           http://www.protogeni.net/resources/
                              rspec/2/request.xsd"
       type="request" >
  <node client_id="my-node"
        exclusive="true">
    <sliver_type name="raw-pc" />
  </node>
</rspec>
```

- **Ticket** is a promise signed by an aggregate manager, giving an entity the right to allocate the resources that are being granted.

- **Credential**, on the other hand, is a grant of a set of rights and privileges associated with a particular principal.

In [72] are reported all the interfaces needed to further comprehend the interaction between the different parts of the SFA. These are not fundamental to our needs, but an eager reader can find there all the necessary material. Once all the previous abstractions

have been performed on the existing infrastructures, hence both the federation and the testbeds have implemented the API correctly, all the blocks that SFA is composed of are ready.

Finally, the interaction between actors and abstract resources is shown in Fig. 4.8.

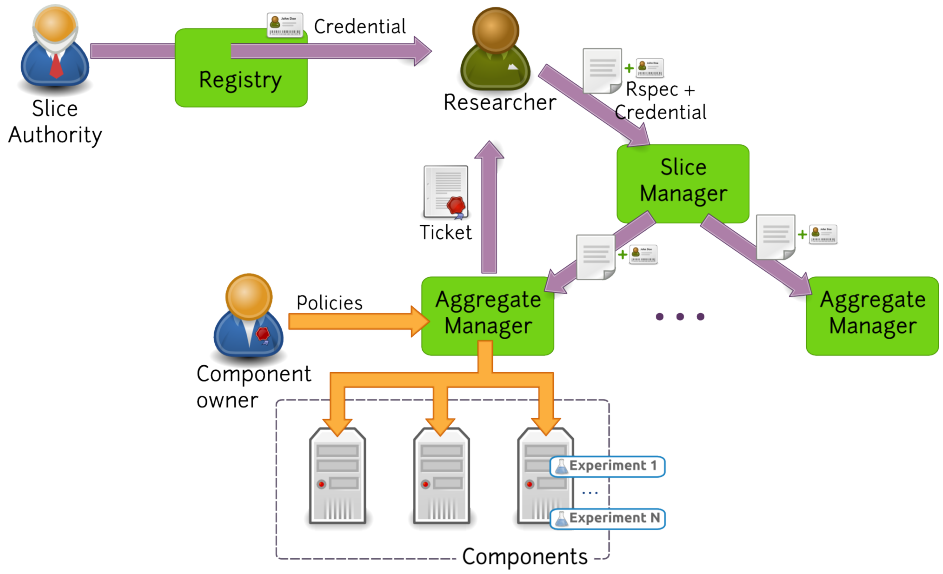


Fig. 4.8 SFA actors, authorities and abstract resources interaction.

The workflow to inact an experiment is the following:

1. Researchers need to first acquire valid credentials from the SA, which grants them specific access rights to the federated resources. Depending on these rights, the allowed specifics of the reservation may change.

2. Certificates allow researchers to correctly submit their custom Rspec to the SM. As we previously said, Rspec definition depends on each testbed and federation, because the SFA simply defines basic interfaces, therefore GENI suggests to use predefined tools such as Flack [75], or to modify existing examples.

3. The SM then parses the Rspec and forwards the request to the target AM, which grants an experimentation ticket to the researcher, with predefined expiration date and rights of use. Together with the ticket, the AM instances the requested slice, which is, from now up to the expiration date, reserved.

4. The experiment must agree with the final policy of use, which is always defined by the component owner. Policies may determine maximum durations for the experiment as well as restrictions on the diffusion of the collected data to the general public.

5. Once the researcher submits a valid experiment, the AM disseminates the already instanced components with the experiment code, and the experiment is started, and the researcher's focus may shift without any issue on the monitoring tools provided by the federation.

6. When either the ticket expires or the experiment is concluded, and in case the researcher didn't refresh its subscription, the slice is automatically erased and the resources are released for future uses.

This concludes the formal introduction to the SFA architecture. Any other implementation detail is out of our concern, for now. Fed4FIRE integrates SFA modules in a specific architecture, which can be seen in Fig. 4.9, that describes the resource discovery, reservation and provisioning architecture. In this case, the role distribution is the following:

- The *Testbed* shall provide the physical resources and their interfaces, like an SSH server or a service on top of them. In order to be able to manage them in a federated way, it is mandatory to provide a compliant Aggregate Manager [AM] (see next). Finally, if willing to do so, it can also offer additional federation services that match the Federator ones (member and slice authorities).

- The *Federator* must provide a registration portal, a documentation center, member and slice authorities (see next), as well as directories for authorities, services and AMs.

- The *Experimenter* should be equipped with a compliant browser, together with stand-alone tools (if needed, like SSH client, FTP client, etc.).

All these and additional information can be found in [76].

As a review, we recall that the SFA framework aims to provide a flexible background for component-based architectures, that - due to dependencies, privileges and a reservation system - may assume a hard-to-deal-with, non-linear structure. This is performed through a series of abstraction, that ease the way in which resources are handled throughout the federation. In addition, the SFA framework also allows to secure the communication between different layers of the architecture, by means of certification authorities, credentials, and firm policies. It is not the only solution viable to build large federation, be it clear, but its good features made it popular in the field. Additionally, large freedom of implementation is granted by the architecture, and the presence of many existing, open source implementations of this architecture helped its

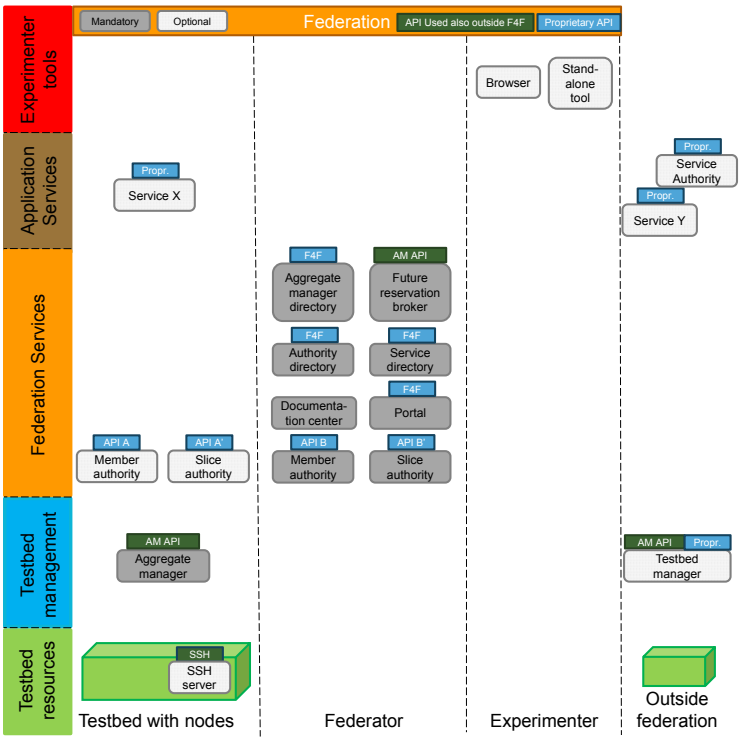


Fig. 4.9 Fed4FIRE architecture for discovery, reservation and provisioning

diffusion. The one we employed for our contribution was developed under the FiTeagle Project [77], and we chose it as it granted us the largest degree of freedom and ease of use, while being an almost stand-alone application.

4.3.3 Orbit Management Framework [OMFv6]

OMF [78] is a Testbed Control, Measurement and Management Framework, and, in its latest release, Version 6, it is one of the supported experiment control tools in Fed4FIRE. As well as OML, OMF was originally developed for the ORBIT wireless testbed at Winlab, Rutgers University. Since 2007, OMF has been actively extended to operate on testbeds with many different type of network and resource technologies. It is now deployed and used on different testbeds in Australia, Europe, and in the U.S. OMF is currently being extended further to support exciting new features and technologies.

It is a powerful tool, that from the experimenter's point of view, provides a set of tools to describe and instrument an experiment, execute it and collect its results, while from the testbed operator's point of view, provides a set of services to efficiently manage and operate the testbed resources.

OMF client and server application comes as standalone, as its measurement counterpart OML, and the framework exists out of 3 parts:

- Every node runs a Resource Controller (RC)
- There is one Experiment Controller (EC) from where the experiment control script runs
- There are PubSub servers which send around the FRCP messages between the EC and RCs

Experiment description can be performed through OMF, using the OMF Experiment Description Language (OEDL) [79], which is based on Ruby, but provides its own set of experiment-oriented commands and statements.

An OEDL experiment description is composed of 2 main parts:

- A first part where we declare the resources that we will use in the experiment, such as applications or computing nodes, and some configurations that we would like to apply to these resources.
- A second part where we define the events that we would like to re-act to during the experiment's execution, and the associated tasks we would like to execute when these events occur.

For additional informations, refer to the main documentation of OMF. Fundamentally, all testbeds within the federation should support FRCP-based experiment control, therefore also OMF; further information on the state of the implementation within each testbed can be recovered from Fed4FIRE documentation.

4.4 Federative Tools

Here next we introduce in details just the tools that are fundamental for our contribution. The federation provides an user-friendly interfaces between experimenters and facilities (jFed), as well as slice reservation tools for SFA-compliant testbeds (SFI). In addition, it provides experiment control tools (OMFv6), as well as instrumentation tools (OML). We showcase these tools in order.

4.4.1 jFed

First of all, Fed4FIRE provides the users with a Graphical User Interface [GUI] and Command Line Interface [CLI] tool, jFed [80], which provides most of the basic functionalities.

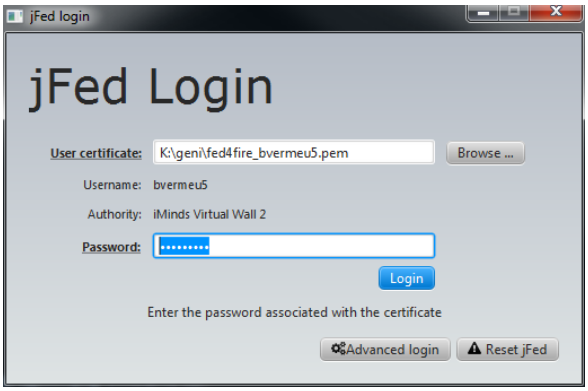


Fig. 4.10 jFed login step, with standard authorization.

jFed was developed by iMinds Association [81] and is a general tool that can be employed by any other authority, provided it is given the credentials by iMinds. To use the tool one must have registered a free account and obtained an user certificate (*.pem*) from the iMinds certification authority. The login screen is in Fig. 4.10.

The tool is written in Java 1.7/1.8, and therefore it is able to run on most machines. It is subdivided into three components, independent from one another, as the target users are different for each of them:

1. **jFed Experimenter GUI and CLI**, which allow end-users to provision and manage experiments; it is a very powerful tool, that allows experimenters to design with incredible depth their experiments. The GUI (see screenshot in Fig. 4.11) provides a slick design tool to instantiate resources, personalize them, design the network, and interact with all the resources via integrated ssh terminals. A large selection of firmwares gives great freedom to the experimenter, who may also, if needed, provide custom firmware, start-up scripts, and control multiple machines at once via practical multi-machine code injection.

2. **jFed Probe GUI and CLI**, which assist testbed developers in testing their API implementations; the tools allows to choose the testbed to poll, and provides a complete list of all the standardized commands than can be sent. It contains an HTTP and XML parser, so that replies can be visualized and analyzed locally on the tool. A screenshot of the GUI is in Fig. 4.12.

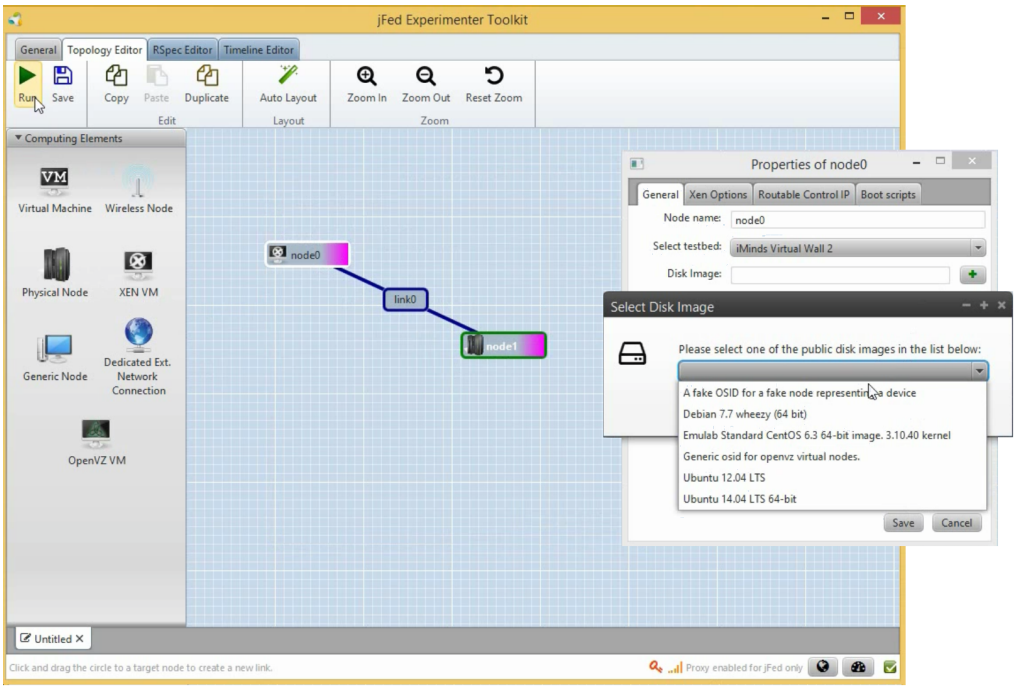


Fig. 4.11 jFed Experimenter GUI, network design tool.

3. **jFed Automated tester GUI and CLI**, perform extensive full-automated tests of the testbed APIs and testbeds, in which the complete work-flow of an experiment is followed. Nightly tests are available, and testbed monitoring in Fed4FIRE is performed using also this tool.

4.4.2 Other SFA Tools

Fed4FIRE provides several additional SFA interfaces:

- **SFI**, or Slice Facility Interface, is a command line SFA implemented in python as part of the (freely available) PlanetLab implementation. It provides the functionality to create, update and display a slice. SFI also supports resource discovery, reservation and provisioning. It can also be used to release resources, and to start and stop a slice.

- **OMNI** is a GENI command line tool for discovering and provisioning of resources at GENI Aggregate Managers (AMs) via the GENI AM API. The Omni client also communicates with Clearinghouses (also known as Control Frameworks or CFs) to create slices, and enumerate available GENI AMs. A Clearinghouse is a framework of resources that provides users with GENI accounts (credentials). Users can use these credentials to reserve resources in GENI AMs. Any AM API

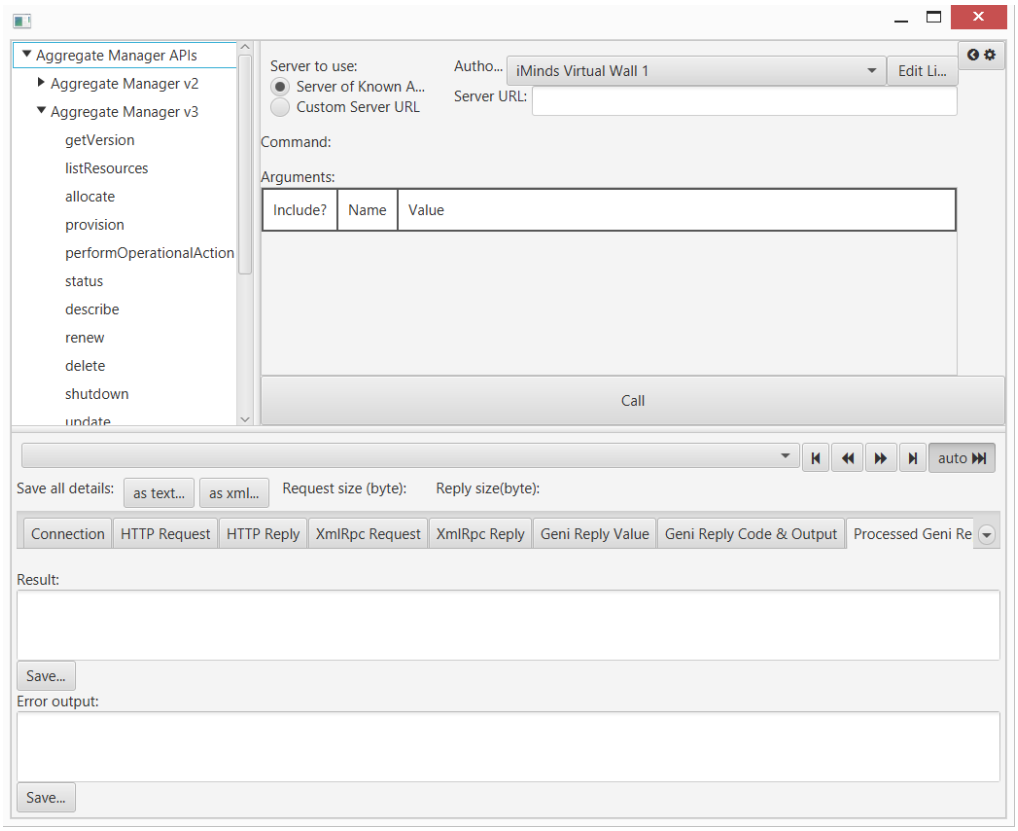


Fig. 4.12 jFed Probe GUI, SFA API tester section.

compliant aggregate should work with Omni. These include SFA, ProtoGENI, OpenFlow and GCF.

- **Flack**, which was cited beforehand, is a visual client for SFA based aggregate managers, such as the ProtoGENI and federated GENI aggregate managers. Flack covers the main functionalities of authentication, discovery and resource provisioning over SFA compliant testbeds. It also allows the experimenter to create, open and update slices.

4.4.3 YourEPM

In the latest version of Fed4FIRE, service orchestration is also implemented based on the "Your Experiment Process Model" [YourEPM] tool, which is designed to provide high level service orchestration for experimenters, based on open standards such as BPMN and BPEL. Detailed informations are available in [82].

YourEPM provides to the end user a practical GUI, that allows to automatically access the services offered by each testbed and advertized in the Service Directory. The

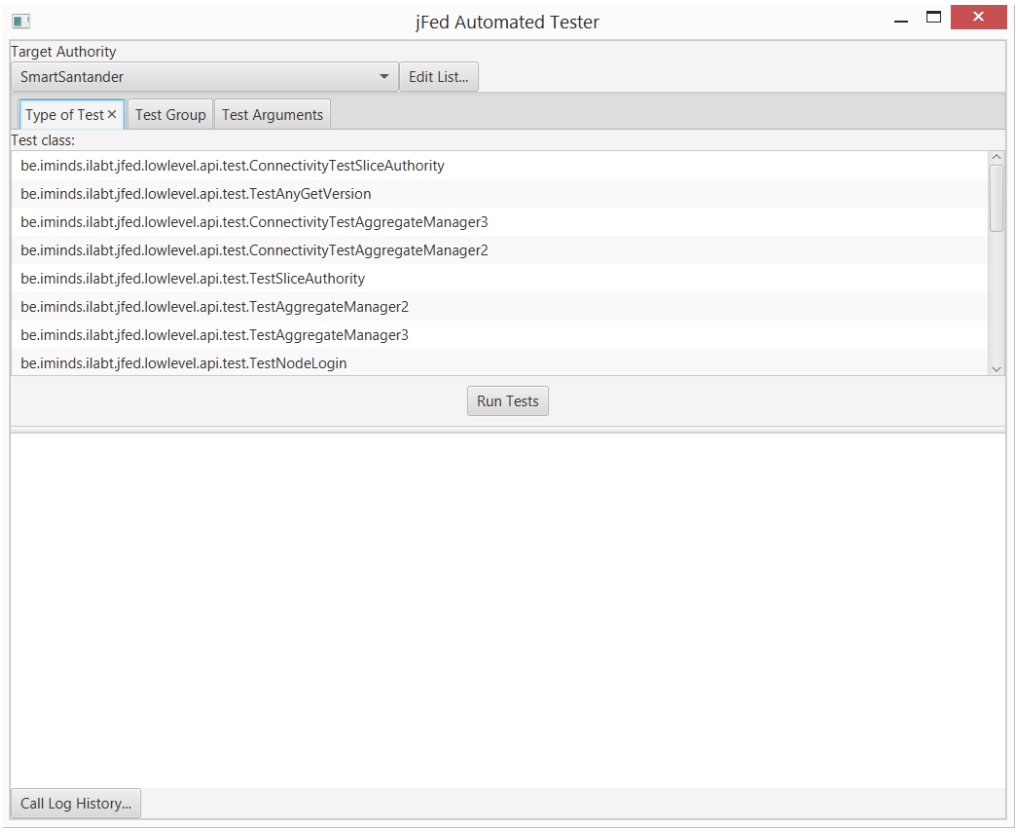


Fig. 4.13 jFed Automatic Tester GUI, SmartSantander section.

communication with the service from the tool is done using general wrappers to specific technologies (i.e. REST, SFA), and testbeds that desire to support YourEPM just need to provide a description of the service API, so the tool can invoke it automatically.

An experiment run using YourEPM is in Fig. 4.14.

4.5 Federative Approaches

Fed4FIRE allows for three different types of integration to the federation, *associated*, *light federation* and *advanced federation*.

- **Associated**: this is the basic level, and just requires to be listed under Fed4FIRE site and provide basic documentation on the testbed.
- **Light Federation**: this level is achieved by testbeds whose resources **cannot** be controlled with Fed4FIRE federated tools (portal, jFed, YourEPM), but provide access through their own tools by using Fed4FIRE compatible credentials. In details, the testbed must accept Fed4FIRE PKC12 certificate, and the API for

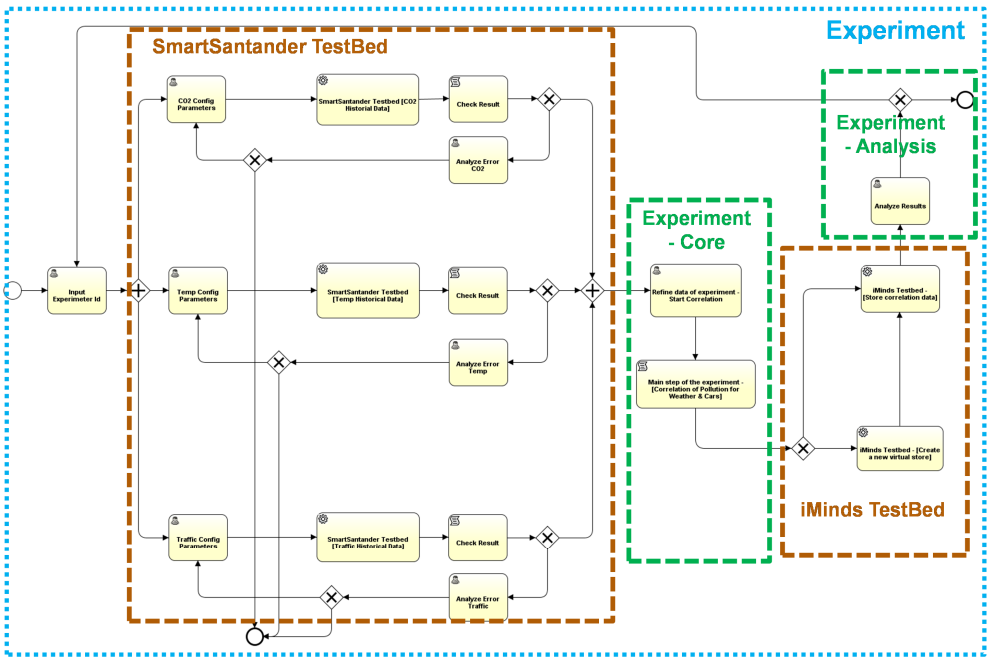


Fig. 4.14 Experiment run using YourEPM tool.

resource description, discovery, reservation and provision must be supported as well. Each testbed must then provide detailed documentation and public IPv4 or IPv6 addresses to the service endpoints.

- **Advanced Federation**: the advanced level is achieved by testbeds whose resources **can** be controlled with Fed4FIRE federated tools. Fed4FIRE Portal, jFed and any other SFA compatible client can be used to control resources from those testbeds that provide a compatible SFA AM. In details, adding over Light Federation requirements, Advanced Federation requires that the testbed supports all AM v2/v3 functionalities, and that resource description, discovery and provisioning for the AM are implemented as well. Among the other details, which can be found in [82], the testbed shall provide a public IPv4 address for AMs, specific IPv4 or IPv6 for SSH login (direct or using proxy), and, finally, at least one administrator must be registered to fed4fire-experimenters google group.

Currently, Fed4FIRE presents 21 associated testbeds, 11 of which are also advanced federated testbeds, 2 are light federated only testbeds (SmartSantander and FIONA), while the remaining 10 testbeds are in progress to switch to advanced architecture. As we said, SmartSantander is currently one of the two testbeds being only light-federated, and is now working to become an advanced federation as well.

Chapter 5

Technical Contribution

In this chapter we finally overview the main topic of this thesis, i.e. the contributions towards the federation of SmartSantander asynchronous service layer into Fed4FIRE context.

Currently, SmartSantander is a Light Federation, but migration to Advanced Federations is planned for Cycle 3. Our contribution helps the federation of F4F in two specific aspects: first, it enables an optional feature, which is OML support towards experimenters; second, we developed an SFA wrapper that allows SmartSantander to support AM v2/v3 functionalities, i.e. introduces in the framework an Advanced Federation features. As the two components are not strictly intertwined and can work in an independent way, we will analyze the benefits each of them offer to the final solution. A peek into the whole SmartSantander asynchronous system architecture is provided in Fig. 5.1.

5.1 Subjacent Technology

As the careful reader may have noticed, we have delayed the description of some background computer-science technology, which is required to fully understand the implementation. We start the analysis of the contribution from here.

Python Version and Modules

It may seem trivial, but for the sake of argument we need to specify which version and modules of Python we have used.

Python was used in its release 3.4.3 [83], installed using *pyenv* [84] and hosting the code on a virtual environment through the *Virtualenv* module [85]. The additional

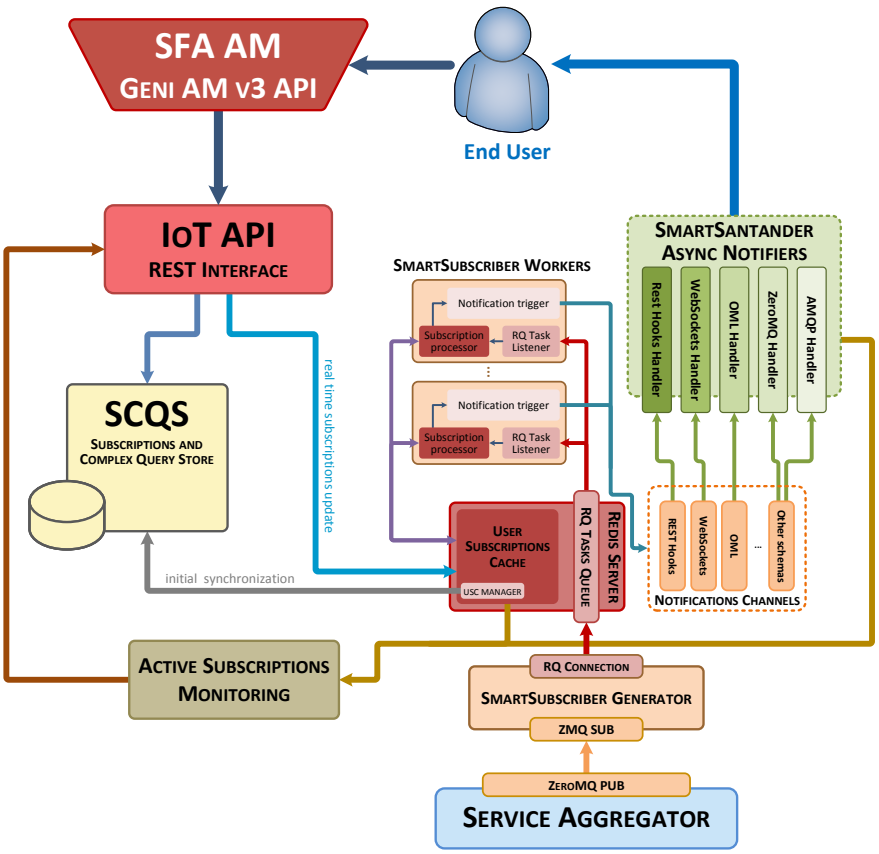


Fig. 5.1 Complete system structure of SmartSantander asynchronous service layer.

Fig. 5.2 Python Logo.

modules employed during development were *asyncio* [86] as Asynchronous I/O library, *oml4py* [87] as OML client library, and *Redis/aioREDIS* [88][89] as Redis client library. In absence of an asynchronous OML library, it was necessary to wrap in an asynchronous routine all OML client calls using asyncio module.

Asynchronous Programming Model [APM]

The Asynchronous Programming Model [APM] is a common model to perform multiple operations in a *pseudo-parallel* fashion. We say pseudo because, in theory, we have a single worker taht executes only a single operation at once, therefore no parallelism is present in the architecture. In practice, through, once the worker has started executing

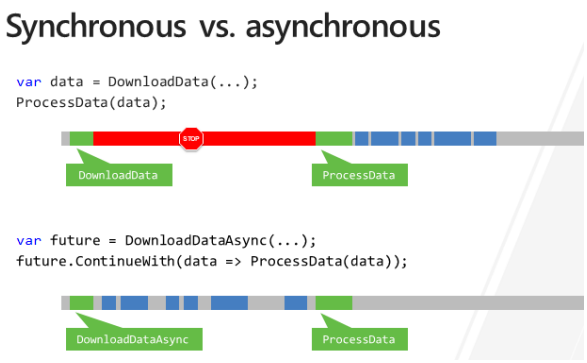


Fig. 5.3 Synchronous vs Asynchronous programming diagram.

an operation, the APM schedules the execution of the remaining operations in the dead-times that the current operation presents, hence saving time with respect to waiting for each operation to end before starting the next one, i.e. in a *synchronous* fashion. See Fig. 5.3 for a quick example.

Instead, the difference between APM and standard parallel architecture (multiple workers that execute multiple functions at the same time) is more subtle. First of all, we point out that the purpose of the two architectures is different: where **parallel** architectures aim to multiply system **performance** by performing multiple operations at once, **asynchronous** systems try on the other hand to provide **responsiveness**, removing dead times by the equation.

Moreover, the benefits given by asynchronous systems depends on who is responsible for the latency. If it is a computational latency, i.e. a function takes time in computing something, then either the asynchronous system employs parallel threads, or no improve is possible. On the other hand, if we are dealing with network operations, and hence with latencies that do not depend on the current architecture, then asynchronous operations may be for the best.

An excellent guide to APM was written by Dave Peticola for Twisted [90], a renown Python framework for APM. In the guide, it is clearly stated that deep understanding of the APM model as well as proficiency with asynchronous systems require time and experience. Although, the payback could be significant: in systems which are affected by external large latency, or especially in event-based systems, efficient use of asynchronous operations may result in improved performances even with respect to purely parallel architectures [91].

To exploit at best the good points of both models, we chose to develop our contribution using an hybrid parallel (threaded) and asynchronous paradigm, so that high performance as well as high responsiveness could be achievable.

SQL and NoSQL Database Model

SQL, or *Structured Query Language*, is a special-purpose programming language designed for managing data held in a *relational database management system* (RDBMS), or for stream processing in a *relational data stream management system* (RDSMS). Relational databases organization is based on the relational model of data, as proposed by E.F. Codd in 1970 [92]. This model opposes to the standard organization of databases, which were usually hierarchical, and organizes data into one or more tables (or "relations") of rows and columns, with a unique key for each row. The model simplicity was one of the reasons why relational databases became so common worldwide.

SQL was one of the first languages that were designed with the relational model in mind. It was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce before 1974 [93] under the name of SEQUEL, or *Structured English QUEry Language*, and it was designed to manipulate and retrieve data stored in IBM's original quasi-relational 1970s database management system (System R). Later on, the one-day-to-be Oracle Corporation developed the first commercially available implementation of SQL, *Oracle V2*, and there began SQL's pandemic diffusion. We say pandemic because, for the past 30 years, SQL-compliant databases have been the stakeholder in the database market, with a small share of market belonging to the so-called **NoSQL** databases.

The rationale behind NoSQL databases is that "not everything fits comfortably inside a relational database", and therefore, in specific cases, ad-hoc structures may be more efficient that standard SQL-compliant databases. This, even though it has always been true, was of small importance to the sector in general, because available data was large but *not as much so that SQL performances would degrade to unreasonable levels.*

Yet, the latest years saw an exponential explosion of data traffic and data storage needs: let us just think that the standard hard-disk capacity was only around 10 MB as of 1985 [94], while now we are probably running out of space very quickly even on 128 GB drives. This increase was so steep that it gave birth to a brand new discipline, that is renown with the general name of Big Data, as we introduced in Chapter 1, and deals with efficient handling of huge chunks of data.

Therefore the market is now striving for optimal performances, and SQL-compliant databases often fall behind NoSQL ones, which are many and different, and usually optimized for specific tasks. An overview of SQL vs NoSQL is given in [95].

Actually, to fit our requirements, we chose to employ a NoSQL database known as Redis, which is detailed in the next section.

Redis Server/Client Model

Fig. 5.4 Redis Logo.

Citing from the official website, "Redis is an open source, BSD licensed, advanced **key-value cache** and **store**. It is often referred to as a **data structure server**, whose keys can contain strings, hashes, lists, sets, sorted sets, bitmaps and hyperloglogs" [96]. The name Redis stands for REmote DIctionary Server. According to the monthly ranking by DB-Engines.com, as of September 2015, Redis is the most popular key-value database [97].

Redis is a feature-rich software with high performance, as it works with an in-memory dataset, made persistent via either a log or a periodic dump to disk. One of the most useful features it offers is the ability to perform *atomic operations* on server keys, from remote, without the need to first read the value and then store it once again. This feature is fundamental for parallel operations, as these atomic operations can be performed in an asynchronous fashion by the Redis itself, without the risk that different users may incur into deprecated data. Another fundamental feature is an optional expiration for keys, therefore it is perfect to host a time-dependent subscription system.

It is a versatile NoSQL database, and its libraries have been translated into an abundance of programming languages, therefore it is easily integrable with existing software.

Technological Outskirt

Here we list very briefly all those technologies that could not be known to the reader but whose knowledge does not compromise the understanding of our contribution.

- **ZeroMQ** [98]: In less than a hundred words, ZeroMQ (also known as ØMQ, 0MQ, or Zmq) is a concurrency framework, dressed up as embeddable network library, that provides sockets that carry atomic messages across various transports like in-process, inter-process, TCP, and multicast. It is feature-rich, and it fits perfectly ANSEL needs it can operate asynchronously, and is supported by most architectures.

- **REST paradigm** [99]: is a software architecture style for building scalable web services. RESTful systems typically communicate over HTTP, in the same fashion as web browser communicate with remote servers (GET, POST, PUT, DELETE, etc). The APIs implemented in SmartSantander allow hence to interact with the complete system using web-like commands.

In the following sections we procede to detail our contributions, providing every time enough details to understand where the contribution inserts into the existing architecture, and how it was designed.

5.2 Integration of OML in SmartSantander

5.2.1 Asychronous I/O OML Notifier and Channel [AION+C]

We contributed to ANSEL developing the Async OML Notifier [AION] and the separate OML Notification Channel [ONC], which is detailed here next. All the software was written in Python 3.4.3, and is made to run stand alone on a separate machine connected to SmS network.

AION+C Version 1

The first version of the contributed system can be seen in Fig. 5.5. It is composed just by the two main blocks, the AION (above) and the ONC (below). In this version we focused on the implementation of the asynchronous system, using asyncio to schedule asynchronous operations, as well as to wrap OML in an asynchronous container. The features missing in this version - like the threaded part of the system - are implemented in version 2.

The ONC is composed by a task queue (TQ) and by an Asynchronous Redis (aioREDIS) client/server pair. The subscription processor, the task queue and the Redis server are placeholders for SmartSantander components, and simply generate/store notifications that the asynchronous Redis client pops, parses, and forwards to the single OML worker (OML Inject) inside the AION.

The ONC's aioREDIS client is responsible for popping notifications from the notifications queue using BRPOP Redis command, and forwarding them to the OML worker. The BRPOP command is fundamental, as it is a "blocking" command, i.e. tells the Redis to wait as long as it finds a new notification in the database entry. Therefore we don't need to implement a polling system, and system complexity lowers.

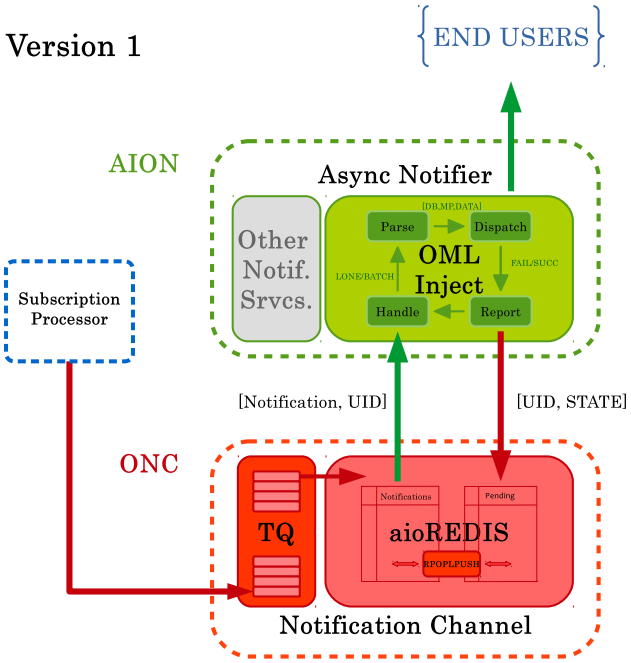


Fig. 5.5 Implementation of AION+C into SmartSantander ANSEL, Version 1.

Next is the AION. As of now, it contains a single OML worker, and the block named "Other Notification Services" is nothing but a placeholder for future workers. The worker is designed to perform 4 operations:

1. *Handle*, in which brand new or failed notification request are receivedl, analized, and, if there is anything wrong, discarded. In case everything is good, they are inserted in a queue, that is consumed as quick as possible by the

2. *Parse* step, where correct requests are parsed, the recipient is extracted, and the measurement is formatted following OML standards. Then

3. *Dispatch* tries to asynchronously send the notification at the specified recipient. To do so, it needs to open a new connection with the OML server, set up on it a measurement point [MP], and send either the measurement or the batch of measurements in a measurement stream [MS], like we said in Section 4.3.1. Then, he asynchronously waits for the result of the operation, that is next handled at the

4. *Report* step, where successes break the cycle, while failures are recorded. If the number of times a specific recipient failed to receive a notification is smaller than N (default: N = 3), then the cycle starts over again. Otherwise, the N-th

successive failure triggers a "failed" notification on the aioREDIS, which stores it in a "Pending" table. The aioREDIS is then in charge of signaling all the pending notifications, whenever possible, to the USC. The USC then decides whether to remove the subscription or to fall back for a random time before retrying to send the subscription once again. We note that this feature was only planned and not implemented at this time, while it was redesigned and implemented in Version 2.

Performance during this phase of development were out of concern, but, nonetheless, the system could generate and fetch data consistently. Most importantly, the focus was on whether the OML worker could perform all its operations asynchronously. As this was the case, we set on moving to the second development phase, where we redesigned the system to envisage using multiple OML workers, simplify the OML Inject routine, add missing features and, most importantly, to integrate the system within SmartSantander's architecture.

AION+C Version 2

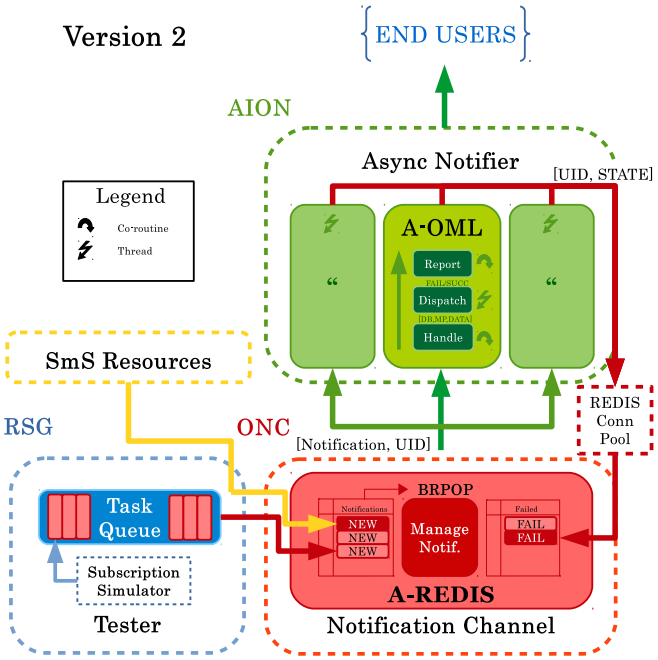


Fig. 5.6 Implementation of AION+C into SmartSantander ANSEL, Version 2.

The second architecture can be seen in Fig. 5.6, and is visibly more polished. In addition to the AION and the ONC, now a third and fourth block are present. The

Random Subscription Generator (RSG) block has been separated from the ONC, and now interacts with the latter using dedicate interfaces. These interfaces are shared with the SmartSantander system, the fourth block, which is now connected to the ONC. The block is an abstraction for the complete ANSEL, therefore we leave out the details already introduced.

The ONC has therefore changed now, even though it still presents two separate lists, one for the new subscriptions, the other for failure notifications.

What has changed the most is the AION. Now, the workers follow a different, simplified routine:

- *Handle*: is the function called by the ONC whenever new notifications are available. Here, the worker verifies that the subscription is correct and valid, i.e. it checks that the notification channel is OML, and that the provided parameters - host IP address and names - are consistent. An example of notification that the ONC receives is in Listing 5.1.

Listing 5.1: Example of ANSEL notification received by the OML worker.

```
{
    "target": {
        "technology": "oml",
        "parameters": {
            "application": "omltestapp",
            "address": "193.190.127.238",
            "port": "3003",
            "experiment": "omltest"
        }
    },
    "notification": {
        "timestamp": "2015-08-26T17:55:33.396806+02:00",
        "urn": "urn:x-iot:smartsantander:u7jcfa:t286",
        "measurements": [
            {
                "uom": "percent",
                "value": 70,
                "phenomenon": "batteryLevel"
            },
            {
                "uom": "degreeCelsius",
                "value": 24.64,
                "phenomenon": "temperature:ambient"
            },
```

```
25              {
26                  "uom": "lux",
27                  "value": 7759.28,
28                  "phenomenon": "illuminance"
29              }
30          ],
31          "location": {
32              "type": "Point",
33              "coordinates": [
34                  -3.7983,
35                  43.46354
36              ]
37          }
38      }
39 }
```

To send the data in OML format we first parse the notification, extract *host, database information* and *measurements*, convert them to *string* or *integer* in accordance with SmartSantander's specification, subdivide them into *namedtuples* (Python's named arrays) for ease of transportation, and finally define an ad-hoc OML measurement point through which the data will be sent. The Python code that does all this is in Listing 5.2.

Listing 5.2: Python snippet which handles the OML-formatted measurement.

```
1 # ad-hoc tuples
2 db = namedtuple('db','name dbname host')
3 mp = namedtuple('mp','name format')
4
5 @asyncio.coroutine
6 def handle_request(self, request):
7     # parse the xml request
8     req = self.parse_request(request)
9     # define host
10    host_def = "tcp:" + str(req.params['address']) + ":" + str
          (req.params['port'])
11    # define database with host
12    db_def = db(name = str(req.params['application']), dbname
          = str(req.params['experiment'], host = str(host_def)))
13    # define the measurement point (MP) to use
14    mp_def = mp(name = 'single_measurement', format = 'urn:
          string timestamp:string m_uom:string m_value:string
          m_phenomenon:string loc_geojson:string')
```

```
15      # harvest the data
16      msr_data = req.notification
17      # schedule the injection (use futures)
18      future = asyncio.Future()
19      # inject
20      asyncio.async(self.inj_OML(db_def, mp_def, msr_data,future
            ))
21      # set-up callback
22      future.add_done_callback(self.return_outcome)
```

- *Dispatch*: the dispatch operation has changed itself. Instead of injecting simply asynchronously, now there is the possibility to create new threads to dispatch notifications to different OML servers at the same time. This is for best practice of traffic control, so that we can flee delays. Moreover having separate fluxes allows us to retry to send the notification multiple times in a row, without the need to undergo the whole routine as before. The sending still is asynchronous, though, so that whenever the result of the sending operation is available, it is passed onto the next and last step.

- *Report*: it is the last asynchronous function of the pipeline, where the final report is sent back to the asynchronous Redis server inside the ONC. Like before, in case of success nothing is reported, whereas in case of multiple failures, the UID of the notification is pushed back to the Redis through a Redis connection pool, so that Redis connections are not wasted.

In case of congestion, multiple workers can be spawned easily, and they act independently one another without resource access collision, as Redis connections are queued and served successively by the server. We planned the introduction of an autonomous system that spawned workers basing on the queue behavior, but its need was later on bypassed by the third software version, which we now introduce.

AION+C Version 3

Some performance concerns - shown later on - made us develop a third version of the module, more focused on application level throughput. The v3 architecture stems from that of v2, and is just a global overhaul, therefore no new component is present. The architecture now exploits in a better way the parallel/asynchronous features of the code, especially the parallel side: instead of spawning multiple workers to serve additional traffic, we decided to use a re-designed single worker approach. Now, the

worker is able to spawn new threads freely in which to run OML routines, which causes the performance to be bound not by the number of workers used, but by the system architecture, especially the maximum number of socket connections allowed by the OML library. We underestimated the value to hundreds of connections, to flee collision possibilities, although sometimes the system does indeed experience seldom collisions, which calls for some fine-tuning. The new and final architecture can be seen in Fig. 5.7.

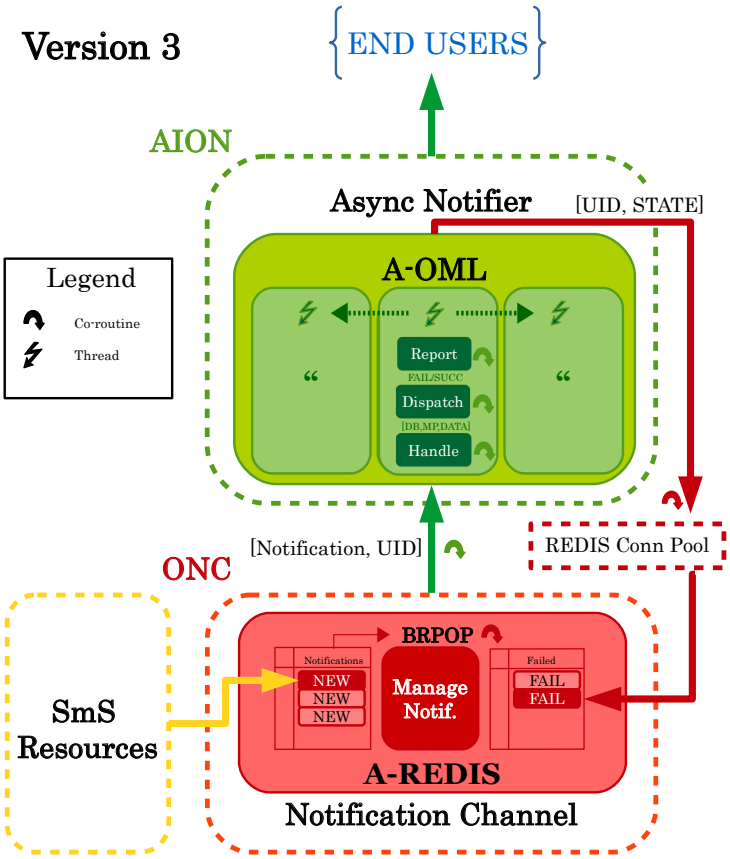


Fig. 5.7 Implementation of AION+C into SmartSantander ANSEL, Version 3.

5.2.2 Demo

Once completed, we set up a small demo of the software, where a virtual user subscribes to SmartSantander via the OML channel and receives the required measurements. The demo goes through the following steps. Each step is linked to a screenshot.

1. **Set-up and run the AION+C, Fig. 5.8**: run the developed software on a virtual machine inside SmartSantander, providing information on the Redis server where new notifications are stored.

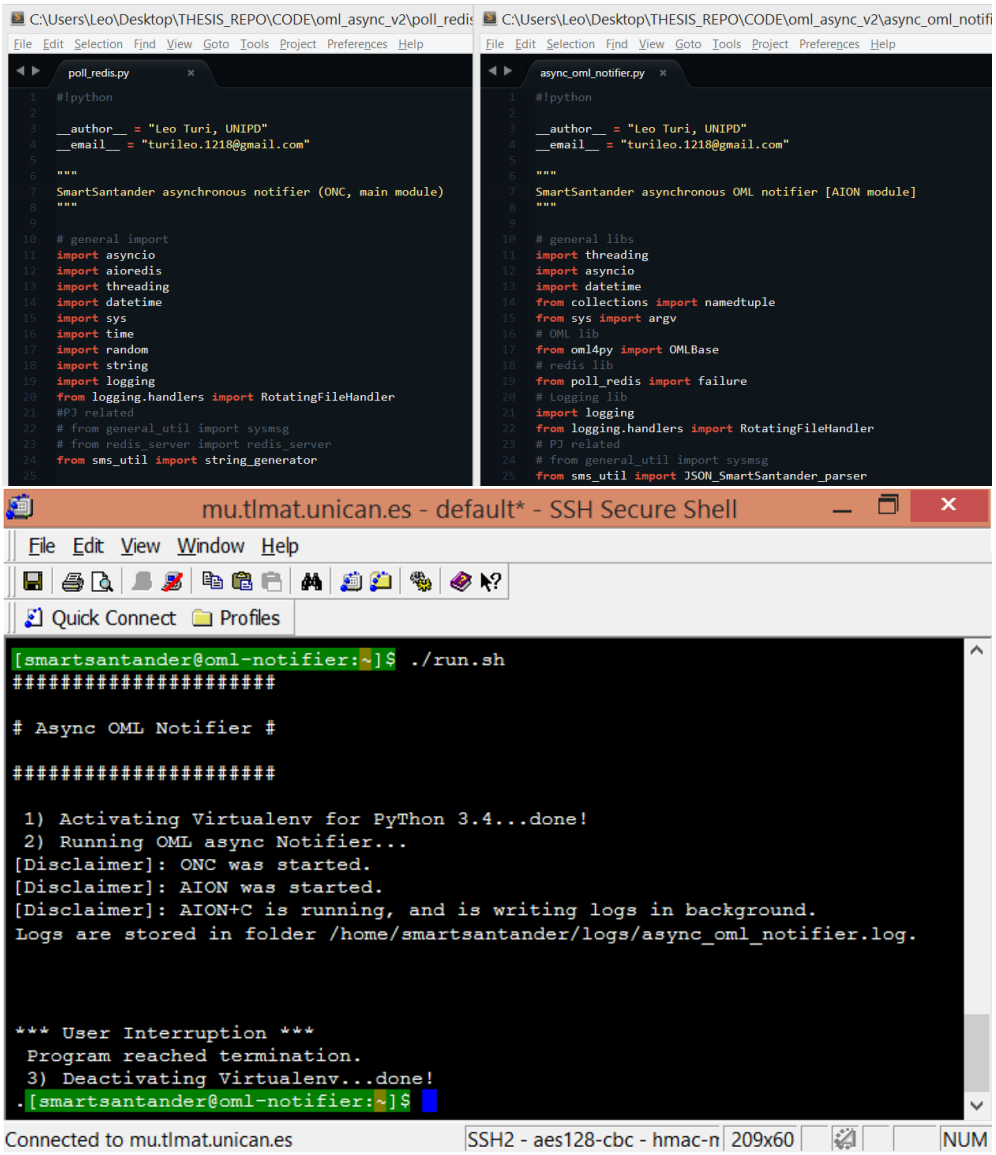


Fig. 5.8 Step 1: Run the AION+C.

2. **Create the virtual user, Fig. 5.9**: we reserve a virtual resource in Fed4FIRE using jFed software, with an OML server that we configure on the fly. The machine has a routable IP address, meaning it is assigned in a subnet which passes though iMinds proxy and reaches up to the jFed client.

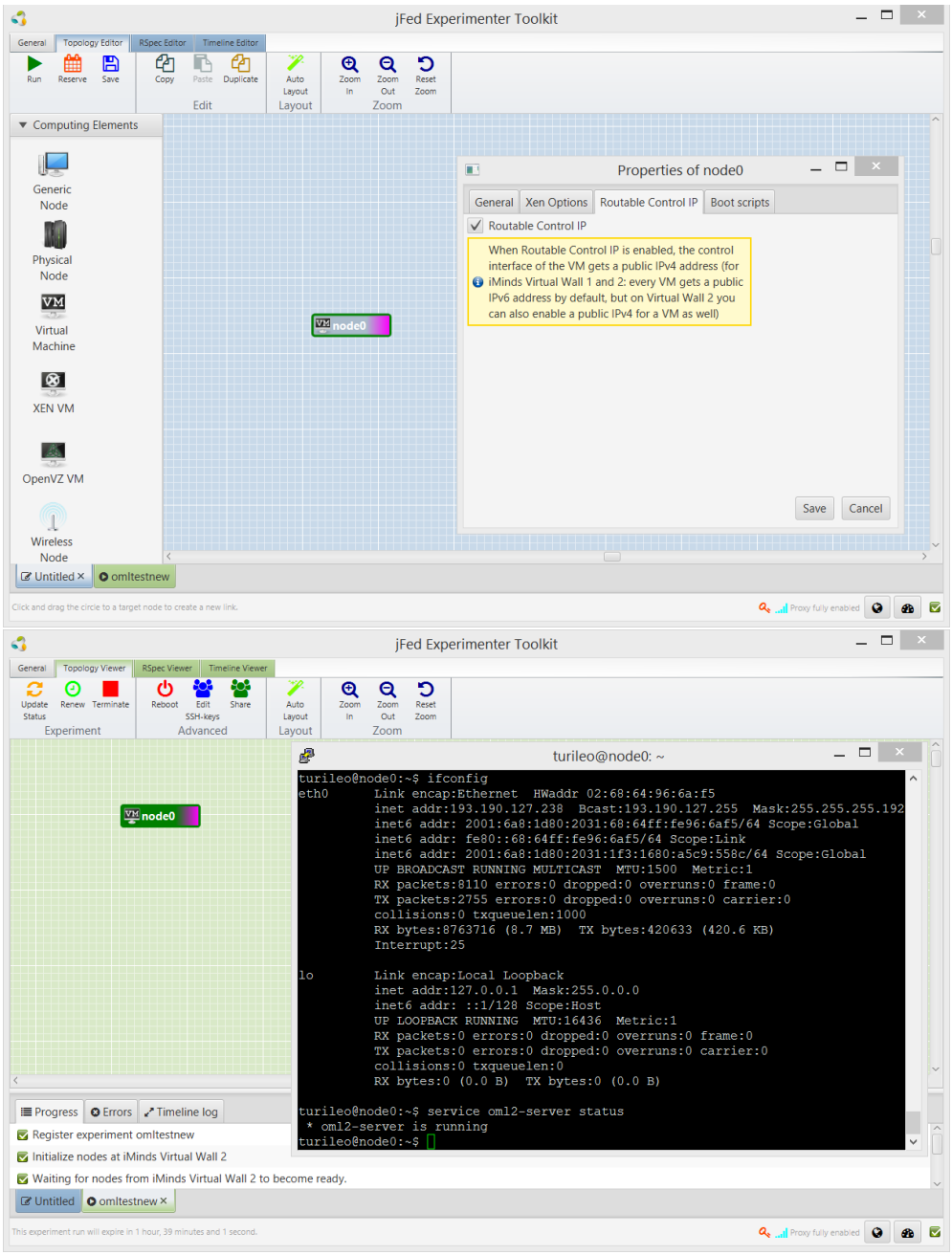


Fig. 5.9 Step 2: Create the virtual user.

3. **Subscribe the user to SmartSantander, Fig. 5.10**: through the RESTful client Postman, we subscribe the user to SmartSantander ANSEL, on the OML channel. The subscription is general, but in the demo we chose to receive all the temperature measurements from the nodes in a specific circular area.

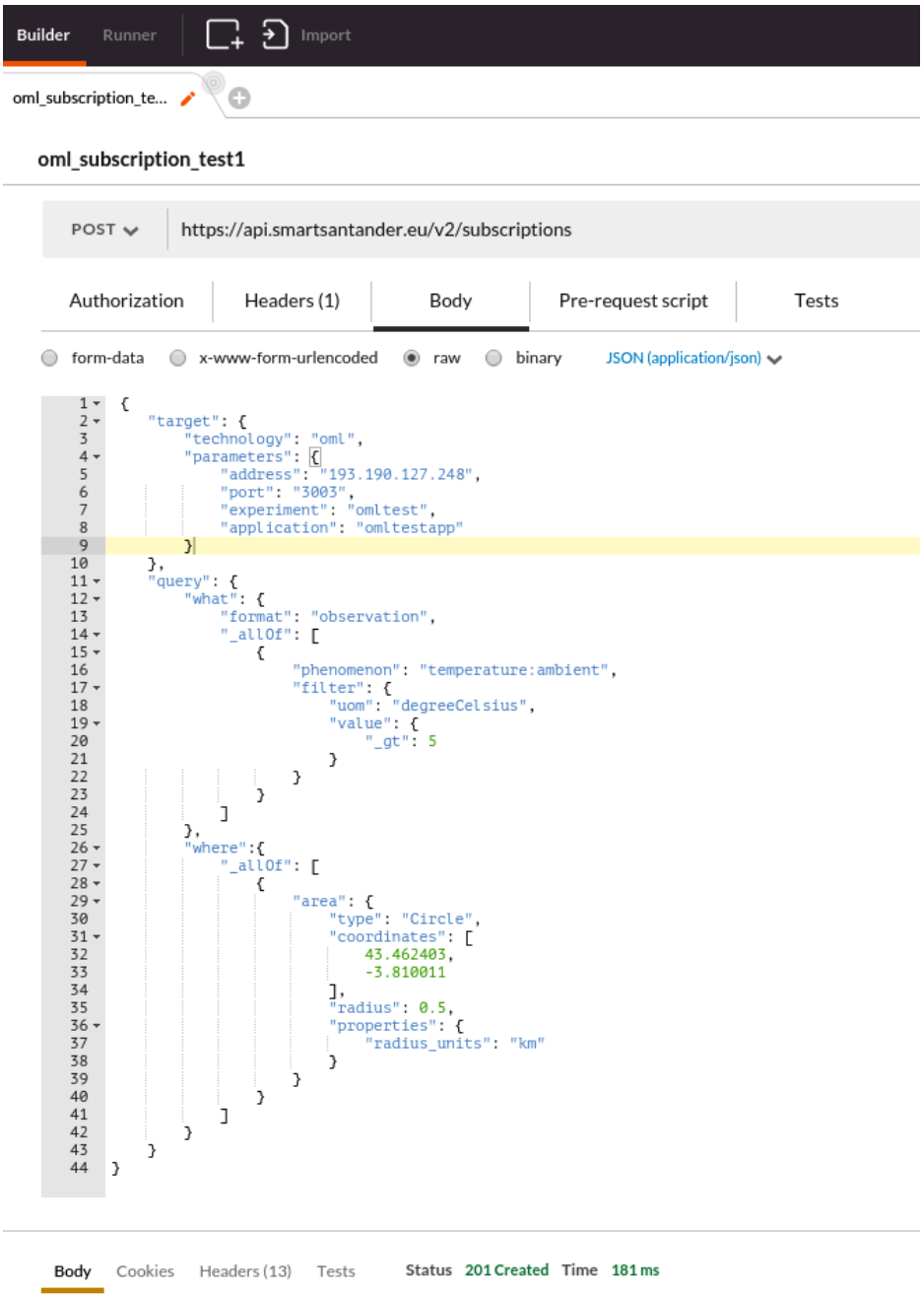


Fig. 5.10 Step 3: Subscribe the user to SmartSantander.

4. **Receive and visualize the measurements, Fig. 5.11**: once the measurements are available, the AION+C forwards them to the subscribed user, and we are able to see them filling up the database linked to the OML server application. The back-end database was downloaded from the virtual machine and read from local, after a sufficient time had passed.

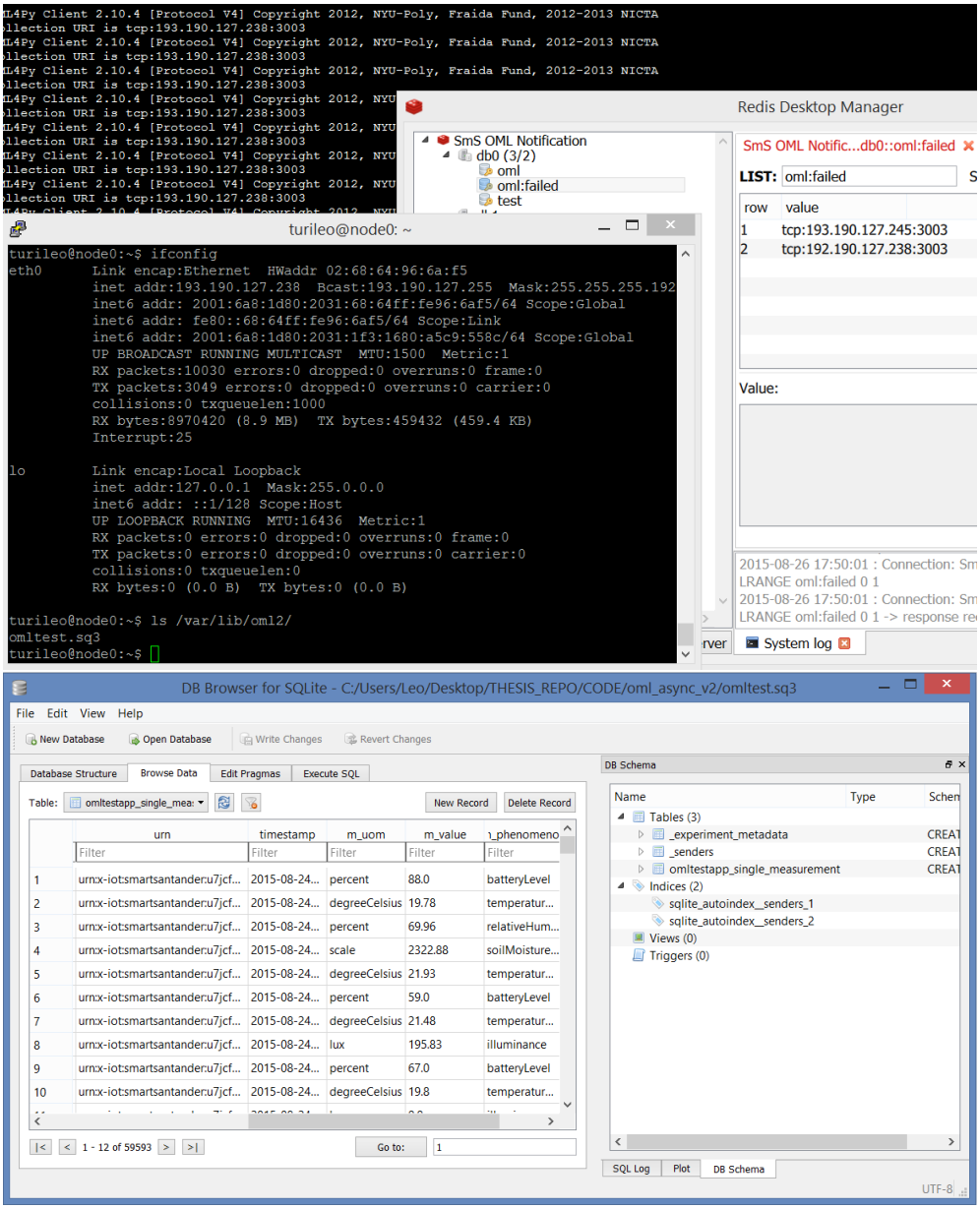


Fig. 5.11 Step 4: Receive and visualize the measurements.

5. **Erase the virtual user, Fig. 5.12**: once we have what we need, we close the experiment using jFed GUI, so that the Federation resources are released correctly for future uses.

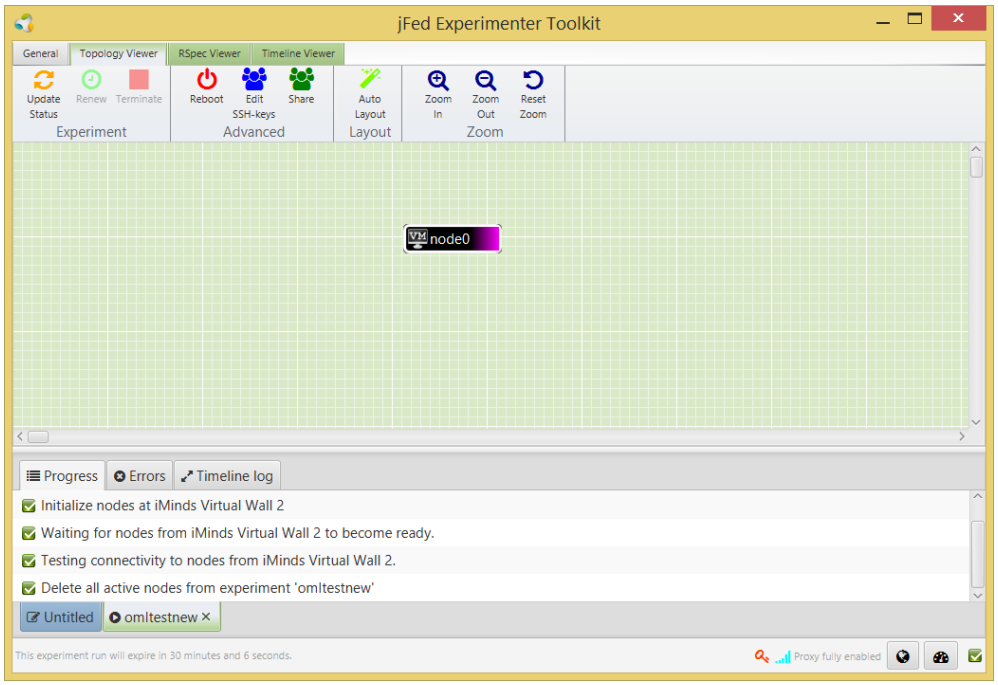


Fig. 5.12 Step 5: Erase the virtual user.

5.2.3 System Performance

Even though the development focused mainly on *impromptu* measurement delivery, and not on performance, we kept an eye on this parameter all throughout the development process, and, since v2 was completed, we run extensive performance evaluations. We note that, being SmartSantander a real-world experimentation site, the experiments are hardly replicable.

Performance Metrics

Due to system design, we base the performance evaluation on two metrics: the notification queue size, and the application-level throughput, in therms of operations per second [op/s]. The former was easy to measure, as Redis *llen()* command enables to quickly read a list size. The second was a bit harder to measure: in AION+C v2 we could measure it by keeping a counter of the OML injections performed by the application, and different workers had different counters; in v3, though, the parallel

architecture doesn't allow us to easily keep counters, and forced us to keep a global counter inside the Redis database to keep track of how many injections were performed. For the sake of investigation, we checked whether the two methods give around the same throughput measures, and they actually do.

General Experiment Setup

The experiments we ran follow the same workflow that presented, but the simulation set-up changes every time. In general, the subscriptions follow the style in Listing 5.3, i.e. we subscribe a single user to receive *all* the observations that are generated at runtime in SmartSantander. Each experiment deals with one or multiple subscriptions, and have from zero to five OML workers that are switched on/off at different times, depending on the purpose of the experiment. A total of 5 experiments were run, and the results are reported hereafter.

Listing 5.3: Asynchronous subscription that captures all the available observations.

```
{
    "target": {
        "technology": "oml",
        "parameters": {
          "address": "193.190.127.250",
          "port": "3003",
          "experiment": "omltest",
          "application": "omltestapp"
        }
      },
      "query": {
        "what": {
          "format": "observation",
          "_allOf": [
            {
              "phenomenon": "*"
            }
          ]
        }
    }
}
```

Case # 0: Background Incoming Notifications Traffic

Experiment zero was run with a single subscription, and zero workers, and is purpose is to highlight the following: experiments are hardly replicable. The main reason behind this is the way the queue grows, i.e. the background incoming notification traffic. We ran several experiments of small duration to try and characterize it, but eventually found that the experiments seemed incoherent at most. This is due to the real-world nature of SmartSantander: there are both fixed and mobile sensors, connectivity is not always available, sensing is irregular, notifications are injected in the system when possible, and some sensors (e.g. smartphones) enter and exit the system almost at random. Therefore, we know for sure that rates are everything but constant, and delays equally so. This was known a priori, but we found practical proofs while experimenting: the notification queue at times grows constant, at times undergoes sudden acceleations with no particular reason, at varying time windows and intervals.

Fig. 5.13 shows a single experiment, which ran through a whole night, and gives a pretty average description of background incoming notifications rate. The growth rate is almost constant around 7 notifications per second, but the rate oscillates between 5 and 12, with sudden burst up to 40 notifications per second. These bursts may last minutes, and some of the following experiments were carried out while the system underwent bursts of notifications. Whenever this occurs, we point it out inside the case analysis.

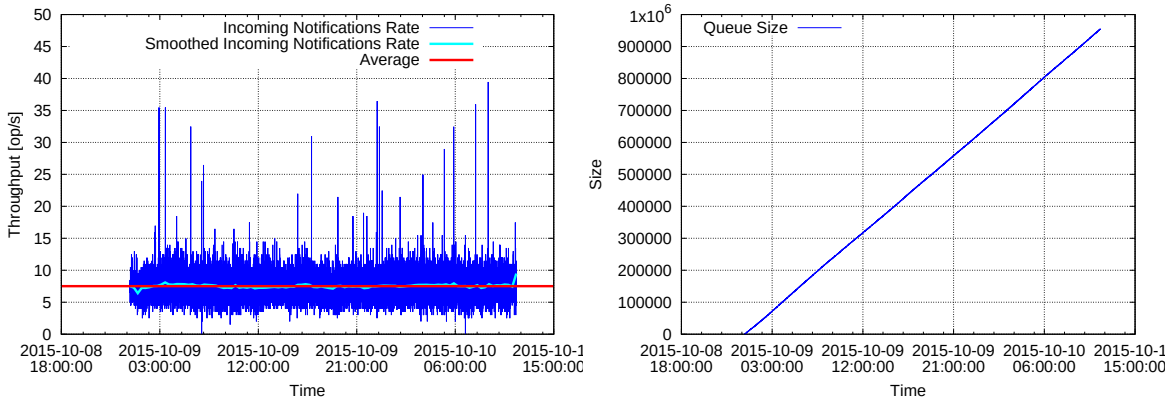


Fig. 5.13 Case #0: Background incoming notifications traffic.

Cases # 1/2: Single-user scenario (v2)

The first performance-evaluation experiment was run under v2, and it was designed as follows: we generated a single user, and submitted a subscription to collect *all*

the observations (not *measurements*) that were available at runtime. Every time the system updates the failure list, we collected the total number of delivered notifications, the length of the notification list present in the Redis server, and computed the *oml delivery throughput* and the *queue length*. The experiments ran for several minutes each, to provide us enough data to analyze. The results of this first performance test on the v2 architecture are in Fig. 5.14, Fig. 5.15, where the user was respectively online and offline.

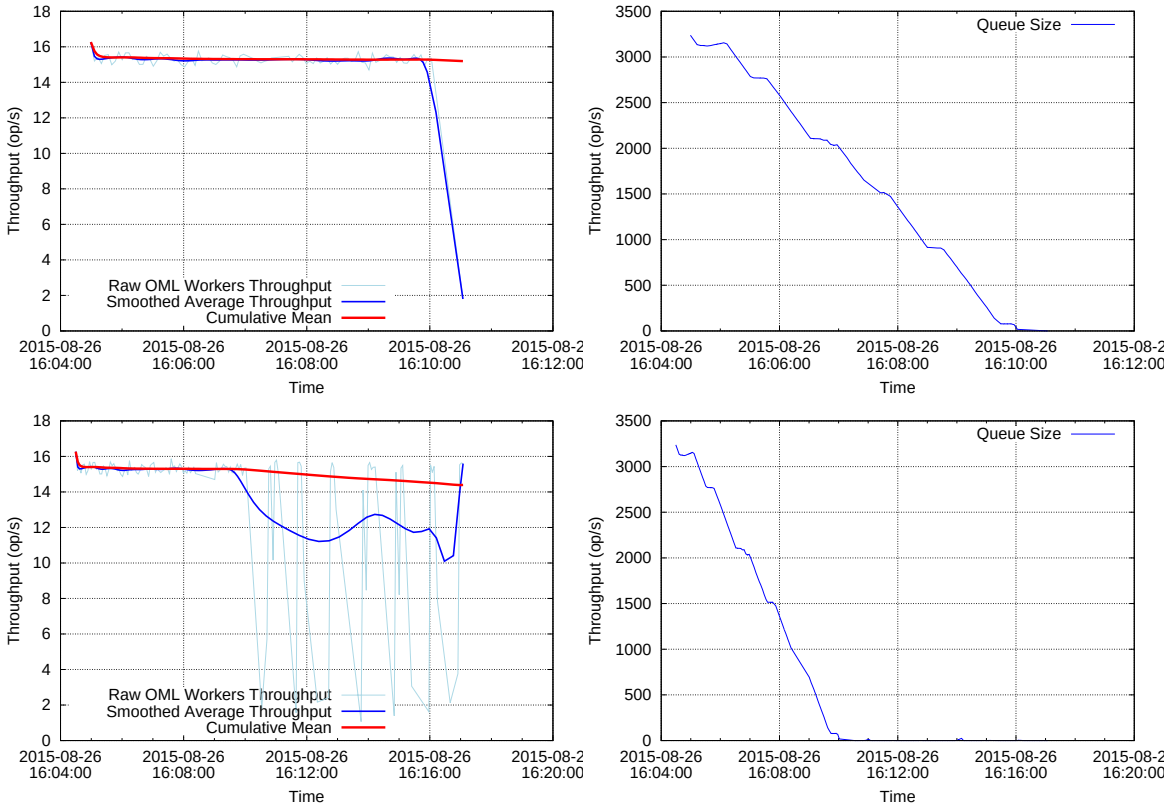


Fig. 5.14 Above, the system going at full regime. Delivery throughput is steady around 15 op/s, and the wavy behavior of the queue is caused by notifications being sent burstly by the sensors. Below, we let the system run while the queue had already been consumed, and we see that they system delivers the notifications as soon as they arrive, keeping the queue almost empty.

Cases # 3/4 Multi-user scenario (v2)

The second time we ran a multi-user experiment, with 5 different terminals as in Fig. 5.18. In the worst case scenario, all the users require *every* observation that arrives at the mainframe, therefore we sent 5 subscriptions for 5 different users, with the said

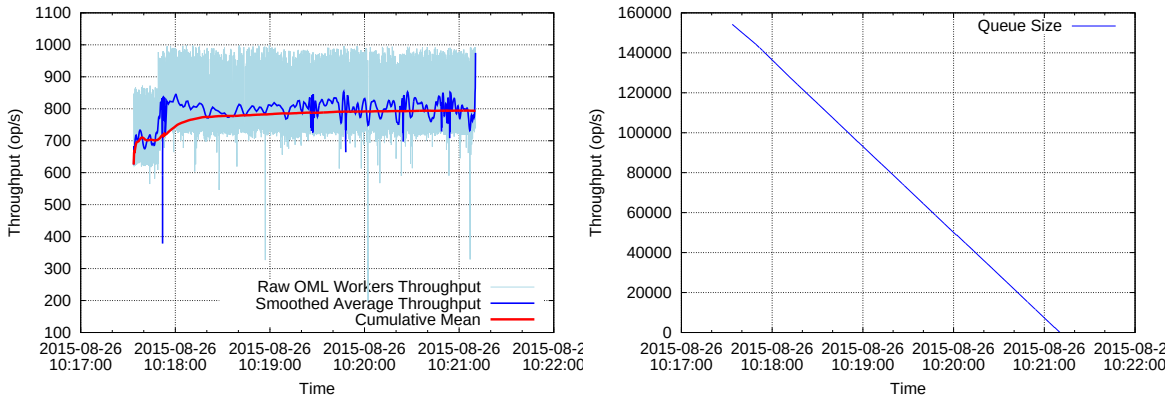


Fig. 5.15 When the user is offline, the AION soon notices this - in 1-2 failed deliveries - hence informs the ONC, and skips all the notifications directed towards the offline user. Queue drops linearly as communication with the Redis server is almost istantaneous.

requirement. The scenario is unlikely to happen, luckily: as we see in Fig. 5.16, the strain on a single OML worker is too high, and the queue soon starts to fill up.

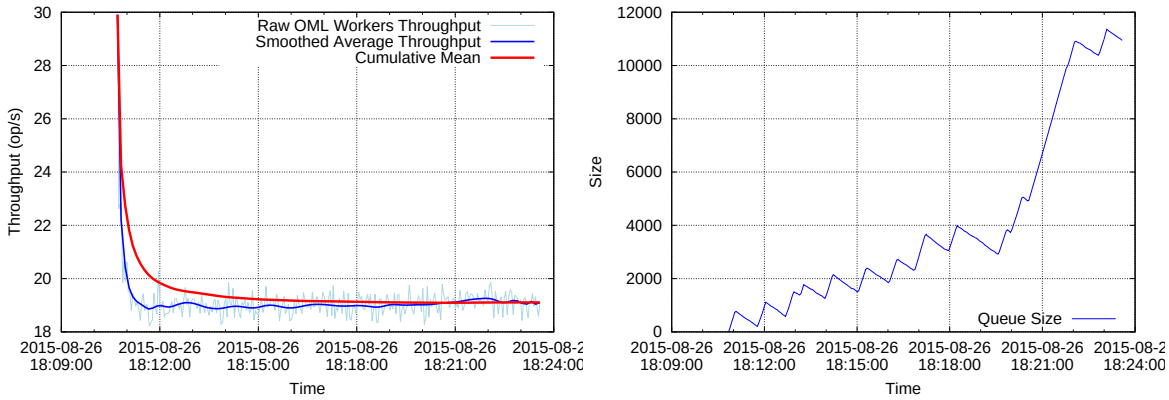


Fig. 5.16 Due to current system design, 5 users with the same subscription generate 5 times the initial traffic. We see that the OML worker undergoes a performance improvement of 25% thanks to the heavy load and the asynchronous architecture. Though it is not sufficient to serve all the requests, and the queue explodes soon, keeping a staircase behavior due to uneven input traffic.

The stair-case behavior of the queue is due to inconsistencies in the measurement flow. In this case, the incoming measurement flux has 5 times the size of the previous experiment, because the system is designed to spawn a single notification per each subscription, even though the notification recipient are the same. Therefore, we need more than one OML workers to keep up with the incoming stream.

Actually, congestion is solved when we spawn more than a single OML user, as we can see in Fig. 5.17. In this case, we initially launched 5 workers, and during

this time the queue decreased quickly. Then, at around 15:11 we started putting to sleep the workers, one every few seconds, with a consequent stair-case decrease of the aggregate throughput. Eventually, since 15:16, only a single OML worker was still active, hence we fell back the the previous congestion scenario. It is interesting to note that throughput variance is higher the more workers there are. This is probably due to queues in the access to the socket and the Redis server, but never due to collisions, as OML streams are parallel.

In the specific experiment, we also see that 5 workers are more than enough to dispatch all the incoming traffic, and, instead, that also 2 workers are able to serve this traffic. The OML throughput results to be steady around 19 op/s, therefore incoming notification rate is about 30 observations per second, an average value. When we replicated the experiment at a month distance, though, the average incoming notification speed was only 7 observations per second, which could have been served easily by a single worker. These inconsistencies are hard to deal with, and, as we see next, v3 architecture is more adapt to serve the system. Actually, incoming notification rate may also increase several times.

These considerations, as well as some underlying changes in the platform that have been made only recently, convinced us to develop another version of the module, which cares more about delivery performance.

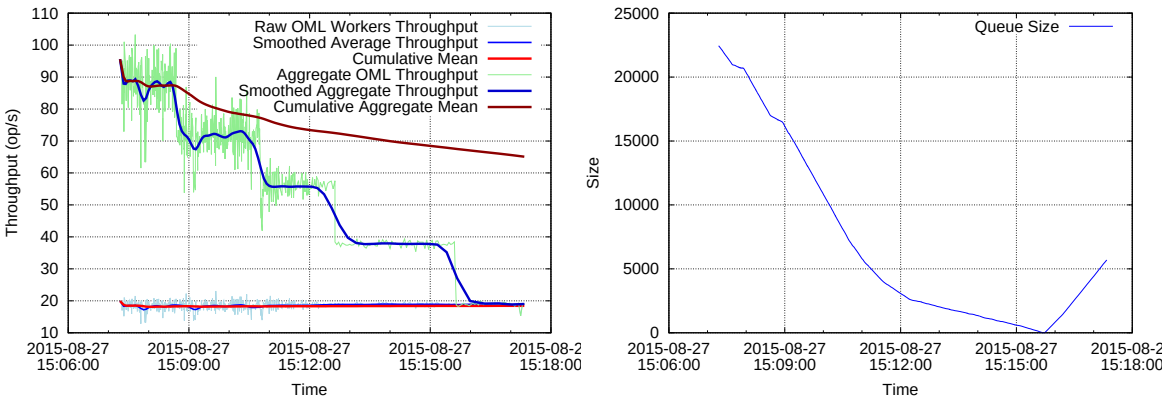


Fig. 5.17 Initially we have 5 active workers, which perform at around 18-19 op/s, as we can see from the average and aggregate OML throughput on the left plot. Then, the workers are deactivated one after the other, at different times between 15:12 and 15:16. After 15:16 we are back in the previous scenario, with a single OML worker not being enough to keep up with the volume of notification that arrive.

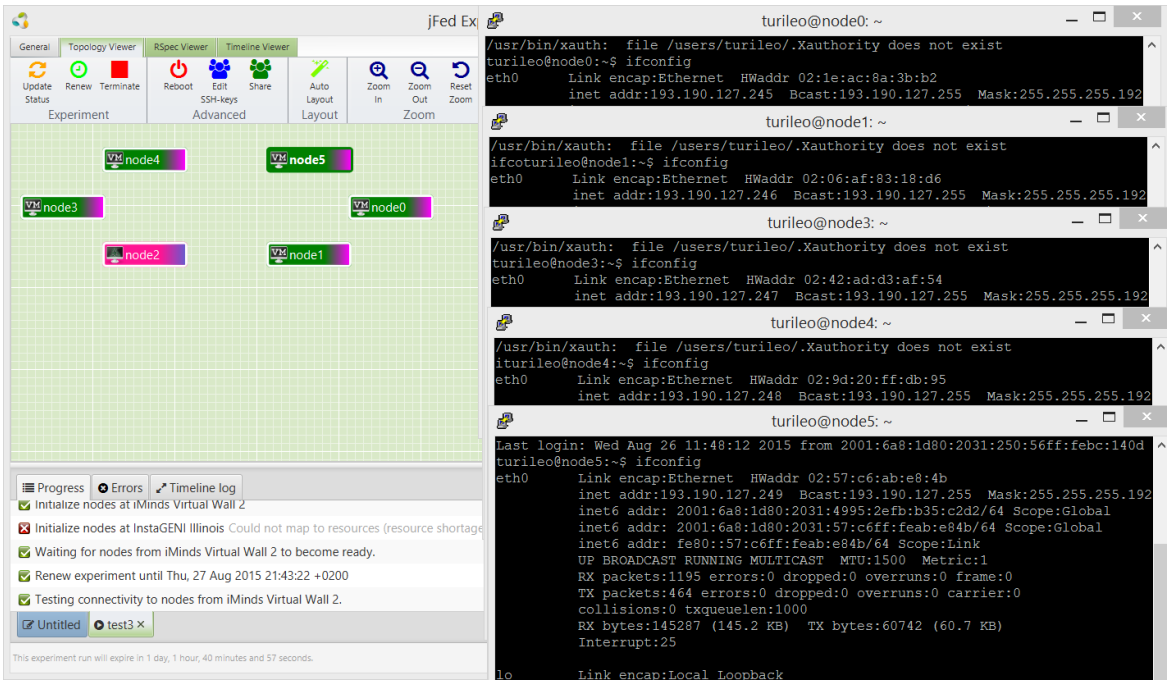


Fig. 5.18 Case # 3/4: Performance evaluation for AION+C, multi-user scenario. System set-up: 6 machines were reserved using jFed, but only 5 were available.

Case # 5: Optimized multi-user scenario (v3)

The fifth experiment was run using the latest v3 module architecture, and it dealt with the same 5-subscription multi-user scenario as the third and fourth experiments. The results are visible in Fig. 5.19. On the right, the throughput actually refers to the OML worker only, while the speed at which the queue decreases is affected by that at which new observations are generated. In the specific experiment, average fill-up speed was of around 180 op/s, which is a higher w.r.t. the previous cases. It is also relatively high, though, as some devices may send up to 500 or 1000 observations per second, but only for short periods of time. During our tests we never incurred in this case, but the information was given by former developers of SmartSantander.

We see that this third architecture is, in any case, more than able to serve high traffic, which is a significant point, and marks a miliar stone in its development. Further improvements will fine tune the code and how the system accesses to the sockets, as seldom collisions still happen while injecting parallel OML fluxes.

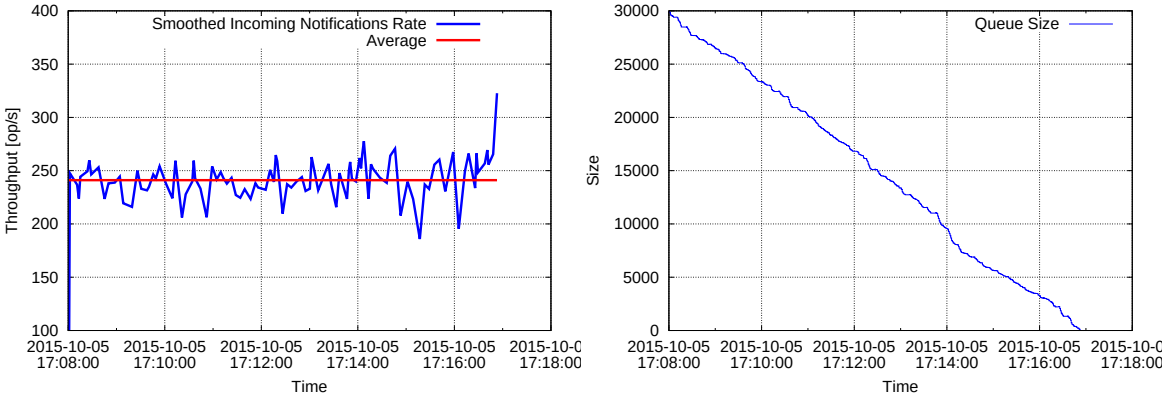


Fig. 5.19 Throughput evaluation: v3 (250 op/s) is around 15 times higher than v2 (15-16 op/s). Now, even with a single worker the high traffic queue drops at significant speed, as it can be seen on the right.

5.3 Integration of SFA in SmartSantander

In this section we tackle the second part of our contribution, i.e. the implementation of SFA API into SmartSantander framework. Our purpose in this phase was to integrate a basic version of an SFA AM into SmartSantander, so that all the resource management and reservation could have been performed using solely an SFA-enabled tool, without the need to access SmartSantander framework via other channels, e.g. REST API.

Unfortunately, time constrains didn't allow us to fully complete this part, so this section includes the theoretical approach on *how* it would be possible to integrate such technology into SmartSantander framework.

5.3.1 The Idea

The approach we chose to adopt in order to implement an SFA AM is based on FITeagle platform [77]. As FITeagle is an integrated system, though, we were not able to decouple the SFA API from the main core of the software - even after speaking with FITeagle developers - due to heavy dependencies. Therefore, we chose to run FITeagle system on one of SmartSantander VMs, and to use an ad-hoc *Resource Adapter*, which are FITeagle standard extensions, as a proxy between SmartSantander and FITeagle. Then, we employ a tool like jFed, which offers SFA functionalities, to communicate with FITeagle Adapter and perform resource discovery and reservation on SmartSantander resources, using the so-called IoT API REST Interface. This satisfies our aims, and allows us to perform all the necessary operations on SmartSantander framework using jFed tool, without needing to directly access the IoT API from remote using REST

clients. This approach ease the experimentation process for SFA trained users because SmartSantander asynchronous subscriptions are generated by the system based on SFA calls behind the scenes and they don't need to know the SmartSantander subscription model. However, when using this interface the experimenter is only taking advantage of a subset of the whole functionality SmartSantander asynchronous system can offer.

We start the explanation of the system from the inner layers, therefore we first introduce FITeagle itself.

5.3.2 FITeagle

Fig. 5.20 FITeagle logo.

FITeagle is "A Semantic Resource Management Framework", or, in more details, it is a "semantic- and micro-services-based resource management toolkit for federated infrastructures, such as testbeds." [77]. It is a software which was born within the FIRE and GENI context, and shares with them the idea of interconnecting the existing testbeds into autonomous, self sustained interconnected entities. Whereas FIRE and GENI are environments that nurture the birth of new federations, FITeagle sets as a powerful tool to *help* federations organize and been born. It is based on FIRE-Teagle [100], and its main aim is to help the interconnection of testbeds by providing services to describe, register, orchestrate, interconnect, provision and monitor heterogeneous resources (such as physical and virtual computation-, storage- and networking- resources and services) across participating provider and user domains. The tool is *semantic* in which it employs *ontologies* as a method of abstraction. It was build with the idea in mind of "an architecture that could be extensible enough to abstract from resources and federation mechanisms, while at the same time support in principle all aspects of the experiment life-cycle" [101], which is one of its main points.

Three are the positive features of FITeagle we identified: first, it is open-source; second, it runs in a Docker-based WildFly environment; third, it is extensible by means

of the above-said *resource adapters*. Where the first positive feature is crystal clear, being developed using WildFly [102] (previously known as *JBOSS*) means that it is based on the open-software state-of-the-art for reliable, dynamic Java (J2EE) servers. It is quick, responsive, extensible, and secure. Moreover, the introduction of Docker [103] makes the software deployable anywhere, independently from the environment, and makes it achieve maximum compatibility.

The third point is that the software was developed with extensibility in mind: *resource adapters* are easy to build, and allow the software to use existing resources like system plug-ins. We dedicate a separate section on how to build them, but now we point out that extensibility is very important for an open software: in a similiar way to federation's virtuous cycle, ease of use helps visibility, which helps the software to diffuse.

Last but not least, a very good feature of FITeagle is its implementation of SFA: it is versatile, slim, and provides compatibility with the GENI Aggregate Manger API Version 3. It is compliant with the specifications and also provides its own authority, which gives a starting point for those testbeds or federations which yearn to implement it.

FITeagle is open-source and deployable on local machines with a very practical bootstrap, `bash <(curl -fsSkL fiteagle.org/sfa)`. During our tests, we installed a copy on a local machine and positively verified that it was compliant with GENI AM API v3 using its own authority.

SmartSantander Adapter and ANSEL Integration

The schematics we foreseen to implement are represented in Fig. 5.21. Essentially we need to implement a FITeagle resource adapter as a proxy for the different operations SmartSantander asynchronous subscription management platform can offer via the IoT API. The technology to be used when implementing this kind of adapters is Java.

Adapters represent an abstraction layer between FITeagle internal data representation and the outer world. Fundamentally, adapters commute calls to FITeagle framework, which can be done through different user interfaces and then translated into a common ontology based model inside FITeagle core, to the different testbed management platforms. In our case, user interactions will be done through the SFA interface, and once transformed into the internal FITeagle model, the different Java classes implemented in the adapter will convert the different operations requested by the user into the equivalent calls to the SmartSantander IoT REST API.

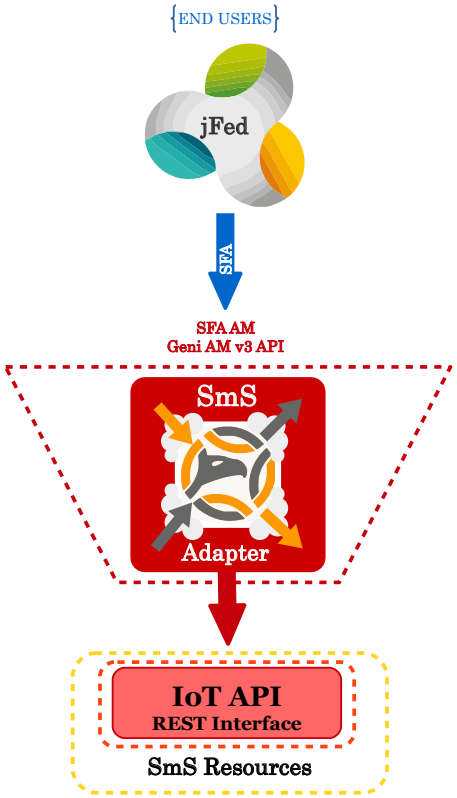


Fig. 5.21 System design to integrate SmartSantander Adapter into ANSEL.

FITeagle team developed an application skeleton, called MotorAdapter [104], which offers a practical example of adapter implementation. Let the MotorAdapter be a "Garage", which allows unspecialized users to access the "Motor" resources, which come with their ad-hoc definition. FITeagle, in this case, plays the role of the "Mechanic", that does all the dirty job behind the garage door. Communication between the user and the Mechanic happens in common (semantic) language, while all the operations the mechanic performs on the Motors are obfuscated to the user, but they are performed through the functionalities offered by the Garage.

As stated before, the resource adapter practical implementation has not been carried out as part of this thesis work. Time constrains, together with the fact that semantic related knowledge is required to understand FITeagle internals, has not allowed us to achieve a full experiment life-cycle integration following the advance federation model in Fed4FIRE. However, a very complete analysis and evaluation of a potential SmartSantander Adapter and how it should perform has been done, and will be used in the future implementation carried out by SmartSantander team.

Chapter 6

Conclusions and Future Works

During the realization of this work we've dealt with the concepts of Smart City and Federation of Testbeds. We employed three fundamental pieces of technology: SFA, a management framework which provides useful abstractions and interfaces to allow the federation of heterogeneous testbeds; jFed, a powerful tool developed by iMinds that employs SFA and works as interface between users and federations; and last OML, which is a reliable measurement delivery tool developed by NICTA, which uses the abstraction of measurement streams to simplify measurement delivery.

The software designed and developed during this work was made under the context of the European Fed4FIRE project, and all the developed material is currently deployed in production under SmartSantander environment.

In details, we developed a new module inside SmartSantander Asynchronous Notification Service Layer [ANSEL] that allows to send notifications using OMSP protocol, which is the standard behind OML technology. The configuration of the recipient was made using SmartSantander's subscription system, by itself a powerful and extensive tool. One of the challenges was to develop an efficient asynchronous architecture, and we performed thorough experiments to attest the good performances of our software.

Finally, we developed a partial implementation of an SFA Aggregate Manager on top of SmartSantander's IoT API, that allows an experienced SFA user to have transparent access to SmartSantander's sensor orbservations and measurements. By using this interface an experimenter is able to generate a SmartSantander asynchronous subscription and configure it to receive notifications using either OML or any other supported technology.

We foresee specific improvements to do on the platform and the contribution we designed, improvements that were not done during development due to lack of time or lack of tools.

First, the OML library we employed is OML version 2.10.4, which gives support to OMSP v4. OMSP v5 is already available, and even though support for this protocol is provided into **OML v2.11**, the library has not yet be converted to Python and it was only available in C during development. The most significant improvement that would come with OMSP v5 is the support for *vectors*, which would enable to store data in connected structures, instead of simple data types (e.g. location would be saved as a vector instead of two separate values for longitude and latitude).

Additionally, OMSP is by design an unsecured protocol. As part of its involvement in Fed4FIRE project NICTA is working on a new version, OML v2.12, which will provide encryption support. **Secure OML** will be designed to support authenticated channels and differentiated access control, in order to improve access control over the data delivery mechanism. Using Secure OML users would require to be authenticated to receive the measurements, therefore identity theft - which is now increadibly simple to accomplish - would be prevented. Moreover, Secure OML should also provide differentiated access control to storage backends, which is often required whenever different parties report and access the collected data through shared deployment. This improvement is expected to be released before end of 2015.

Another future work is what we could not complete due to time constraints, i.e. the implementation of SFA APIs into SmartSantander framework, that would represent a further step towards becoming a fully-federated framework in the Fed4FIRE context.

Interesting improvements with both the contributions could be provided by *ontologies* usage. Ontologies are a representational scheme that focuses on linking ideas and objects together with their properties and relations, according to a system of categories. Formally, they are an alternative to RSpec, that allows users to perform more "human-like" queries on the complete set of objects, by listing singular properties or relations between resources. They are a powerful tool, whose research is supported by both GENI and FIRE through the Open Multinet forum and several different projects, and both OML and SFA future is deeply intertwined with the idea of ontologies.

- For what regards OML, the development of a **Semantic OML** is another of Fed4FIRE future plans: an OML client/server model that supports ontologies, and where resources can be subscribed and delivered using semantic rules instead of single, specific descriptions. In a complete ontologic system, this protocol

would likely improve OML performances. A draft was developed in the OpenLab project, however only a proof-of-concept release was developed.

- In order to enhance slice definition and reservation capabilities, ontologies are planned to be integrated with SFA to generate **Semantic SFA descriptions**. In a somewhat similiar fashion as OML, by using semantic rules instead of specific ones, slices could be more flexible, and resource reservation through SFA could be easier also for the end user, which may use generic, human-like expressions to define slices, thickening the layer of obfuscation between the end user and the actual resources, with improves for both parts.

References

[1] "Testbed." https://en.wikipedia.org/wiki/Testbed. Definition of Testbed, Wikipedia, The Free Encyclopedia.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393 – 422, 2002.

[3] J. Loo, J. Mauri, and J. Ortiz, *Mobile Ad Hoc Networks: Current Status and Future Trends*. CRC Press, 2011.

[4] E. Temprana, E. Myslivets, B.-P. Kuo, L. Liu, V. Ataie, N. Alic, and S. Radic, "Overcoming kerr-induced capacity limit in optical fiber transmission," in *Science*, vol. 348, pp. 1445–1448, June 2015.

[5] J. Heidemann, M. Stojanovic, and M. Zorzi, "Underwater sensor networks: applications, advances and challenges," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 370, no. 1958, pp. 158–175, 2011.

[6] R. Merz, D. Wenger, D. Scanferla, and S. Mauron, "Performance of lte in a high-velocity environment: A measurement study," in *Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges*, AllThingsCellular '14, (New York, NY, USA), pp. 47–52, ACM, 2014.

[7] F. Campagnaro, F. Favaro, F. Guerra, V. Sanjuan Calzado, M. Zorzi, and P. Casari, "Simulation of multimodal optical and acoustic communications in underwater networks," *OCEANS 2015 MTS/IEEE Genova*, 2015.

[8] "Samsung Electronics Sets 5G Speed Record at 7.5Gbps, Over 30 Times Faster than 4G LTE." http://www.samsung.com/uk/news/local/samsung-electronics-sets-5g-speed-record-at-7-5gbps-over-30-times-faster-than-4g-lte. Copyright© 1995-2015 SAMSUNG.

[9] M. A. Assad, *A real-time laboratory testbed for evaluating localization performance of WIFI RFID technologies*. PhD thesis, Worcester Polytechnic Institute, 2007.

[10] T. Dreibholz, "The NorNet Project – A Research Testbed for Multi-Homed Systems." Invited Talk at Karlstads Universitet, Nov. 2012.

[11] J. T. Koskinen, J. Poutiainen, D. M. Schultz, S. Joffre, J. Koistinen, E. Saltikoff, E. Gregow, H. Turtiainen, W. F. Dabberdt, J. Damski, N. Eresmaa, S. Göke, O. Hyvärinen, L. Järvi, A. Karppinen, J. Kotro, T. Kuitunen, J. Kukkonen,

M. Kulmala, D. Moisseev, P. Nurmi, H. Pohjola, P. Pylkkö, T. Vesala, and Y. Viisanen, "The helsinki testbed: A mesoscale measurement, research, and service platform," *Bulletin of the American Meteorological Society*, vol. 92, pp. 325–342, Nov 2010.

[12] S. Bouckaert, W. Vandenberghe, B. Jooris, I. Moerman, and P. Demeester, "The w-ilab.t testbed," in *Testbeds and Research Infrastructures. Development of Networks and Communities* (T. Magedanz, A. Gavras, N. Thanh, and J. Chase, eds.), vol. 46 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 145–154, Springer Berlin Heidelberg, 2011.

[13] T. Miyachi, K.-i. Chinen, and Y. Shinoda, "Starbed and springos: Large-scale general purpose network testbed and supporting software," in *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, valuetools '06, (New York, NY, USA), ACM, 2006.

[14] C. Ameixieira, A. Cardote, F. Neves, R. Meireles, S. Sargento, L. Coelho, J. Afonso, B. Areias, E. Mota, R. Costa, R. Matos, and J. Barros, "Harbornet: a real-world testbed for vehicular networks," *Communications Magazine, IEEE*, vol. 52, pp. 108–114, September 2014.

[15] G. Werner-Allen, P. Swieskowski, and M. Welsh, "Motelab: A wireless sensor network testbed," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN '05, (Piscataway, NJ, USA), IEEE Press, 2005.

[16] J. Zhou, Z. Ji, M. Varshney, Z. Xu, Y. Yang, M. Marina, and R. Bagrodia, "Whynet: A hybrid testbed for large-scale, heterogeneous and adaptive wireless networks," in *Proceedings of the 1st International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, WiNTECH '06, (New York, NY, USA), pp. 111–112, ACM, 2006.

[17] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols," in *Wireless Communications and Networking Conference, 2005 IEEE*, vol. 3, pp. 1664–1669 Vol. 3, March 2005.

[18] R. Giffinger, C. Fertner, H. Kramar, R. Kalasek, N. Pichler-Milanovic, and E. Meijers, "Smart cities-ranking of european medium-sized cities," tech. rep., Vienna University of Technology, 2007.

[19] M. Deakin and H. A. Waer, "From intelligent to smart cities," *Intelligent Buildings International*, vol. 3, no. 3, pp. 140–152, 2011.

[20] *dg.o '13: Proceedings of the 14th Annual International Conference on Digital Government Research*, (New York, NY, USA), ACM, 2013.

[21] "Smarter Cities." http://www.ibm.com/smarterplanet/us/en/smarter_cities/overview/. IBM.

[22] "Smart Cities Report." http://www.iso.org/iso/smart_cities_report-jtc1.pdf. ISO/IEC JTC 1, 2015.

[23] "Internet of Things By The Numbers: Market Estimates And Forecasts." http://www.forbes.com/sites/gilpress/2014/08/22/internet-of-things-by-the-numbers-market-estimates-and-forecasts/. Gil Press, Forbes, published on 8/22/2014 at 1:17PM.

[24] "OWASP Internet of Things Top Ten Project." https://www.owasp.org/index.php/OWASP_Internet_of_Things_Top_Ten_Project. Open Web Application Security Project (OWASP).

[25] K. Ren, P. Samarati, M. Gruteser, P. Ning, and Y. Liu, "Guest editorial special issue on security for iot: The state of the art," *Internet of Things Journal, IEEE*, vol. 1, pp. 369–371, Oct 2014.

[26] Z.-K. Zhang, M. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, "Iot security: Ongoing challenges and research opportunities," in *Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on*, pp. 230–234, Nov 2014.

[27] "The interconnected city: improving the quality of life of citizens." https://www.thalesgroup.com/sites/default/files/asset/document/12pagesA4-SmartCity-GB.pdf. Thales Group, https://www.thalesgroup.com/en.

[28] A. McGovern and A. Geisert, *Aesop's Fables*. Apple Classics, Scholastic, 1990. "The Bundle of Sticks", pp.62-63.

[29] "Dictionary of military and associated terms," July 9 2015. S.v. "interoperability.", http://www.thefreedictionary.com/interoperability.

[30] E. Seidel, G. Allen, A. Merzky, and J. Nabrzyski, "Gridlab—a grid application toolkit and testbed," *Future Generation Computer Systems*, vol. 18, no. 8, pp. 1143 – 1153, 2002. Grid Computing: Towards a New Computing Infrastructure.

[31] F. Donno, V. Ciaschini, D. Rebatto, L. Vaccarossa, and M. Verlato, "The worldgrid transatlantic testbed: a successful example of grid interoperability across EU and U.S. domains," *CoRR*, vol. cs.DC/0306045, 2003.

[32] I. Baldine, Y. Xin, A. Mandal, P. Ruth, C. Heermann, and J. Chase, "Exogeni: A multi-domain infrastructure-as-a-service testbed.," in *TRIDENTCOM* (T. Korakis, M. Zink, and M. Ott, eds.), vol. 44 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 97–113, Springer, 2012.

[33] "Federation." http://www.oxforddictionaries.com/definition/english/federation. Definition of "federation". Oxford Dictionaries.

[34] F. Hermenier and R. Ricci, "How to build a better testbed: Lessons from a decade of network experiments on Emulab," in *Proceedings of the 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom)*, June 2012.

[35] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda, "Indriya: A low-cost, 3d wireless sensor network testbed.," in *TRIDENTCOM* (T. Korakis, H. Li, P. Tran-Gia, and H.-S. Park, eds.), vol. 90 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 302–316, Springer, 2011.

[36] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. M. Parulkar, "Can the production network be the testbed?," in *OSDI*, vol. 10, pp. 1–6, 2010.

[37] "D2.3 – First Sustainability Plan." http://www.fed4fire.eu/fileadmin/documents/public_deliverables/D2-3_Fed4FIRE_First_sustainability_plan.pdf. Fed4FIRE Public Deliverables.

[38] "What is ZigBee?." http://www.zigbee.org/. ZigBee Alliance.

[39] A. Gavras, A. Karila, S. Fdida, M. May, and M. Potts, "Future internet research and experimentation: The fire initiative," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 89–92, July 2007.

[40] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "Geni: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, no. 0, pp. 5 – 23, 2014. Special issue on Future Internet Testbeds – Part I.

[41] A. Willner, S. Albrecht, S. Covaci, F. Schreiner, T. Magedanz, S. Avessta, C. Scognamiglio, S. Fdida, and U. Bub, "Fantaastic: Sustainable management of future internet testbed federations," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–4, May 2014.

[42] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: An overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 3–12, July 2003.

[43] "TESTMAN: Testbed Federation Technology for Interconnection of different types of Testbeds." http://ec.europa.eu/information_society/newsroom/cf/dae/document.cfm?action=display&doc_id=7671. Testman project page, http://testman.naist.wide.ad.jp/, developed under WIDE Project http://www.wide.ad.jp/index.html.

[44] T. Wauters, B. Vermeulen, W. Vandenberghe, P. Demeester, S. Taylor, L. Baron, M. Smirnov, Y. Al-Hazmi, A. Willner, M. Sawyer, *et al.*, "Federation of internet experimentation facilities: architecture and implementation," in *European Conference on Networks and Communications (EuCNC 2014)*, pp. 1–5, 2014.

[45] J. Mirkovic and T. Benzel, "Deterlab testbed for cybersecurity research and education," *Journal of Computing Sciences in Colleges*, vol. 28, no. 4, pp. 163–163, 2013.

[46] "ProtoGENI." http://protogeni.net/.

[47] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis, and D. Pfisterer, "Smartsantander: Iot experimentation over a smart city testbed," *Computer Networks*, vol. 61, no. 0, pp. 217 – 238, 2014. Special issue on Future Internet Testbeds – Part I.

[48] "Business Aspects of the Internet of Things, Seminar of advanced topics, FS 2011, Florian Michahelles ." http://www.im.ethz.ch/education/FS11/iotsem11_proceedings.pdf. ETH.

[49] "SmartSantander Experimental Test Facilities." http://www.smartsantander.eu/index.php/testbeds/item/131-testbeds. SmartSantander Homepage, http://www.smartsantander.eu/.

[50] T. Ojala, "Open urban testbed for ubiquitous computing," in *Communications and Mobile Computing (CMC), 2010 International Conference on*, vol. 1, pp. 442–447, April 2010.

[51] R. Murty, G. Mainland, I. Rose, A. Chowdhury, A. Gosain, J. Bers, and M. Welsh, "Citysense: An urban-scale wireless sensor network and testbed," in *Technologies for Homeland Security, 2008 IEEE Conference on*, pp. 583–588, May 2008.

[52] G. Coulson, B. Porter, I. Chatzigiannakis, C. Koninis, S. Fischer, D. Pfisterer, D. Bimschas, T. Braun, P. Hurni, M. Anwander, G. Wagenknecht, S. P. Fekete, A. Kröller, and T. Baumgartner, "Flexible experimentation in wireless sensor networks," *Commun. ACM*, vol. 55, pp. 82–90, Jan. 2012.

[53] "Campus to become 'living lab' for the IoT." http://www.smart2zero.com/en/campus-to-become-living-lab-for-the-iot.html?cmp_id=7&news_id=10005982&vID=1860&from_mail=1#.VaZF-fmy48Y. Smart 2.0 Website.

[54] "SmartSantander Project." http://cordis.europa.eu/fp7/ict/fire/docs/fp7-factsheets/smartsantander_en.pdf. CORDIS, SmartSantander webpage http://cordis.europa.eu/project/rcn/95933_en.html.

[55] "WP1: Reference Model." http://smartsantander.eu/downloads/Deliverables/D13-Third-Cycle-Architecture-Specification-Final.pdf. SmartSantander Public Deliverables, http://smartsantander.eu/index.php/material/public-deliverables.

[56] "Postman, supercharge your API workflow." https://www.getpostman.com/. Postman.

[57] "FIRE in FP7 ICT Calls." http://cordis.europa.eu/fp7/ict/fire/calls/calls-info_en.html. CORDIS, FIRE webpage http://cordis.europa.eu/fp7/ict/fire/.

[58] "FIRE Offering - Current and Past." http://www.ict-fire.eu/home/fire-offering.html. FIRE ICT Project, http://www.ict-fire.eu/home.html.

[59] "What is Fire?." http://www.ict-fire.eu/getting-started/what-is-fire.html. FIRE ICT Project, http://www.ict-fire.eu/home.html.

[60] "Fed4FIRE Project." http://cordis.europa.eu/project/rcn/105823_en.pdf. CORDIS, Fed4FIRE webpage http://cordis.europa.eu/project/rcn/105823_en.html.

[61] "Fed4FIRE - Events." http://www.fed4fire.eu/events/. Fed4FIRE Project Homepage, http://www.fed4fire.eu/.

[62] "Fed4FIRE Home." http://www.fed4fire.eu/. Fed4FIRE Project Homepage.

[63] "NICTA." http://www.nicta.com.au/. National ICT Australia.

[64] "OML - Orbit Measurement Library." https://mytestbed.net/attachments/download/305/OML.pdf. mytestbed.net reference page for OML project, https://mytestbed.net/projects/oml.

[65] M. Singh, M. Ott, I. Seskar, and P. Kama, "ORBIT measurements framework and library (OML): Motivations, design, implementation, and features," in *TridentCom 2005, 1st International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities* (J. Aracil, S. Kalyanaraman, and K. Mase, eds.), (Piscataway, NJ, USA), pp. 146–152, IEEE Computer Society, Feb. 2005.

[66] J. White, *OML User Manual*. Nicta, Alexandria, NSW, Australia, Oct. 2009.

[67] J. White, G. Jourjon, T. Rakotoarivelo, and M. Ott, "Measurement architectures for network experiments with disconnected mobile nodes," in *TridentCom 2010, 6th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities* (A. Gavras, N. Huu Thanh, and J. Chase, eds.), Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, (Heidelberg, Germany), ICST, Springer-Verlag Berlin, May 2010.

[68] O. Mehani, G. Jourjon, and T. Rakotoarivelo, "A method for the characterisation of observer effects and its application to OML," May 2012.

[69] "Instrument your experiment with OML." http://doc.fed4fire.eu/oml.html. Fed4FIRE Documentation, http://doc.fed4fire.eu/.

[70] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, (Boston, MA), pp. 255–270, USENIX Association, Dec. 2002.

[71] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In vini veritas: Realistic and controlled network experimentation," in *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, (New York, NY, USA), pp. 3–14, ACM, 2006.

[72] "GENI SFA AM API v3." http://groups.geni.net/geni/wiki/GAPI_AM_API_V3. GENI Wiki, http://groups.geni.net/geni/wiki/.

[73] A. Neumann, I. Vilata, X. Leon, P. Garcia, L. Navarro, and E. Lopez, "Community-lab: Architecture of a community networking testbed for the future internet," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*, pp. 620–627, Oct 2012.

[74] "Slice-based Federation Architecture." https://wiki.confine-project.eu/sfa:start. CONFINE Wiki, https://wiki.confine-project.eu/start.

[75] "Flack, resource management tool." http://protogeni.net/flack. ProtoGENI, http://protogeni.net/.

[76] "D2.4 – Second federation architecture." http://www.fed4fire.eu/fileadmin/documents/public_deliverables/D2-4_Second_federation_architecture_Fed4FIRE_318389.pdf. Fed4FIRE Deliverable, http://www.fed4fire.eu/deliverables/.

[77] "FITeagle, A Semantic Resource Management Framework." http://fiteagle.github.io/.

[78] "OMF Overview." http://mytestbed.net/projects/omf/wiki/OMF_Main_Page. OMF Documentation.

[79] "OEDL." http://mytestbed.net/projects/omf6/wiki/OEDLOMF6. OMF Documentation.

[80] "jFed." http://www.fed4fire.eu/events/. iMinds.

[81] "iMinds Home." https://www.iminds.be/en. iMinds Website.

[82] "D2.7 – Third federation architecture." *not available online*, note = Fed4FIRE Deliverable, http://www.fed4fire.eu/deliverables/.

[83] "Python Release 3.4.3 Specifications." https://www.python.org/downloads/release/python-343/. Python Official, https://www.python.org/.

[84] "pyenv Project." https://github.com/yyuu/pyenv. GitHub Repository, https://github.com/.

[85] "VirtualEnv Project." https://virtualenv.pypa.io/en/latest/. Python Packaging Authority, https://www.pypa.io/en/latest/.

[86] "AsyncIO Documentation." https://docs.python.org/3/library/asyncio.html#module-asyncio. The Python Standard Library, https://docs.python.org/3/library/.

[87] "OML4Py Project." https://github.com/mytestbed/oml4py. GitHub Repository, https://github.com/.

[88] "Redis for Python." https://github.com/andymccurdy/redis-py. GitHub Repository, https://github.com/.

[89] "Asynchronous Redis for Python." https://github.com/aio-libs/aioredis. GitHub Repository, https://github.com/.

[90] "Twisted." https://en.wikipedia.org/wiki/Twisted_(software). Wikipedia, https://en.wikipedia.org/wiki/Main_Page.

[91] "Synchronous, Async and Parallel Programming Performance in Windows Azure." http://udooz.net/blog/2012/04/synchronous-async-and-parallel-programming-performance-in-azure/. Udooz Tech Blog, http://udooz.net/blog/.

[92] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[93] D. D. Chamberlin and R. F. Boyce, "Sequel: A structured english query language," in *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, pp. 249–264, ACM, 1974.

[94] "History of Hard Disk Drives." https://en.wikipedia.org/wiki/History_of_hard_disk_drives. Wikipedia, https://en.wikipedia.org/wiki/Main_Page.

[95] M. Stonebraker, "SQL Databases V. NoSQL Databases," *Commun. ACM*, vol. 53, pp. 10–11, Apr. 2010.

[96] "Introduction to Redis." http://redis.io/topics/introduction. Redis Official Website, http://redis.io/.

[97] "DB-Engines Ranking of Key-value Stores." http://db-engines.com/en/ranking/key-value+store. DB-Engines.com, http://db-engines.com/en/.

[98] "ZeroMQ Guide." http://zguide.zeromq.org/page:all. ZeroMQ website, http://zeromq.org/.

[99] "Representational state transfer." https://en.wikipedia.org/wiki/Representational_state_transfer. Wikipedia, https://en.wikipedia.org/wiki/Main_Page.

[100] "PanLab Teagle Main Page." http://www.fire-teagle.org/.

[101] A. Willner and T. Magedanz, "Firma: A future internet resource management architecture," in *Teletraffic Congress (ITC), 2014 26th International*, pp. 1–4, IEEE, 2014.

[102] "What is WildFly?." http://wildfly.org/about/.

[103] "About the Docker Hub." https://docs.docker.com/docker-hub/. Docker Hub Website, https://hub.docker.com/.

[104] "Motor Adapter Git Page." https://github.com/FITeagle/adapters/tree/master/motor.