Designing and Orchestrating Reproducible Experiments on Federated Networking Testbeds

Thierry Rakotoarivelo\textsuperscript{a}, Guillaume Jourjon\textsuperscript{a}, Max Ott\textsuperscript{a}

\textsuperscript{a}NICTA, 13 Garden Street, Eveleigh, NSW, Australia

Abstract

In addition to theoretical analysis and simulations, the evaluation of new networking technologies in a real-life context and scale is critical to their global adoption and deployment. Federations of experimental platforms (aka testbeds) offer a controlled and cost-effective solution to perform such an evaluation. Most recent efforts in that area focused on building those facilities and providing experimenters with tools to allow the discovery and provisioning of their shared resources. Thus many challenges remain in order to support the complete experiment life cycle in a federated environment.

We propose OMF-F, a framework which allows the definition of networking experiments and their execution over shared resources provided by different federated administrative domains. OMF-F provides a domain-specific language enabling rich event-based experiment descriptions. It defines a specific resource model and protocol, which together with its publish-subscribe messaging system allows automatic experiment orchestrations at large scale. OMF-F further provides interfaces to operate with existing resource discovery and provisioning tools for federated testbeds.

Our contributions in this paper are threefold. First we provide detailed descriptions of OMF-F’s design, its architecture, and its involved entities. Then, we present a quantitative evaluation of its underlying messaging and event-handling systems. Finally, we discuss two real examples of OMF-F deployed and used on federated domains to define and execute experiments.
1. Introduction

Networking technologies have had a profound impact on societies and economies, with more than a quarter of the world now regularly using the Internet [1]. While these technologies are based on often solid theoretical frameworks, the intrinsic nature of large networks and the limitation in fully characterising an entire system require extensive simulation and ultimately evaluation at sufficient scale in real-world settings. Until recently the cost and effort associated with building these experimental facilities, also referred to as testbeds, limited their scale and the number of promising research ideas which could progress to the experimental validation phase.

Unfortunately, the problem is not only one of scale, but also one of resource diversity and rapid technology progress. Any testbed built with bleeding-edge equipment will quickly look dated and needs frequent refreshes. In addition, most research groups focus on specific domains which translates to testbeds with a similarly narrow focus which in turn limits its potential audience beyond that group.

There are essentially two approaches to addressing this problem. The first one is to pool the resources of a community and build a small number of large-scale shared facilities. The second approach is to federate a large number of independent testbeds to allow experimenters to assemble larger ‘virtual’ testbeds from many smaller, physical ones. The two are actually complimentary as most experimental investigations progress through different stages of scale and fidelity, where the majority of the experiments can be served by the smaller testbeds.

In fact, this is what happened in the network research community over the last few years. Initiatives, such as GENI\(^1\) in the US and FIRE\(^2\) in Europe (through projects like OneLab\(^3\)) have been developing federation frameworks to integrate existing testbeds and facilitate the addition of new ones with new capabilities. This has led to substantial increase in the number and diversity of available resources. However, the number of users and research outcomes which have been enabled by these large facilities is still lagging.

We observe that most of the effort went into building those facilities, and much less so towards tools to support the experimenter. This was often a deliberate decision, as there is little consensus on how researchers want to use these testbeds. Thus, architectures like SFA\(^2\) adopted the “hour glass” principle of the Internet, and defined a “small waist” on which many different user tools can be built. While we agree with this principle, we believe that its current application is too narrow and too much focused on the provisioning of resources directly provided by the individual testbeds.

We will argue in this paper for a small set of architectural principals and mechanisms to support the experiment life cycle in a federated environment. This in-

---

1\http://www.geni.net
2\http://cordis.europa.eu/fp7/ict/fire/
3\http://onelab.eu
cludes a generic way to describe all resources required for an experiment, the orchestration of the experiment over its life time, as well as the provenance of all produced measurements.

Most of the existing federation solutions provide unified mechanisms to discover and provision resources offered by the testbeds. However, this does not readily extend to resources provided by the experimenters. An example of such resource would be a software service, e.g., a novel content delivery network (CDN) service [3], which may be made available for other researchers to experiment with, effectively running an experiment on top of another one. Discovering, provisioning, and controlling them in the same manner would greatly simplify the tools used to manage experiments. Although the existing mechanisms can be extended to support this, the absence of such use cases in almost all the relevant architecture discussions is concerning.

Most researchers currently rely on custom scripts and ad hoc tools to define and orchestrate an experiment over a direct secure connection (ssh) to the resources. However, these scripts and tools are often closely integrated with a specific experiment realisation, and thus often do not allow its reproduction in a different context or cater for other type of experiments.

The other element missing in most federation architectures is the support for instrumentation and measurements (I&M). This is often assumed to be part of the user tools, or segregated in a measurement plane. We will argue that I&M needs to be managed in the same manner as other resources. This raises issues, such as the harmonisation of resource naming and properties to ensure sound provenance tracking of produced measurements, as well as authorisation, as collecting some measurements may require extended privileges (e.g., a spectrum analyser in a shared wireless testbed). Finally, maintaining long-running experiments requires ongoing measurements to detect and recover from resource failure.

To address these challenges we have developed OMF-F, a generic framework to allow the definition and orchestration of experiments using shared (already provisioned) resources from different federated testbeds. OMF-F is based on OMF [4], which was originally developed for single testbed deployments. OMF-F makes three novel contributions compared to existing experiment orchestration solutions. First, it provides a domain-specific language based on an event-based execution model to fully describe even complex experiments, such as ones involving cars or phones moving out of range of all control networks. OMF-F also defines a generic resource model and concise interaction protocol, which allows third parties to contribute new resources as well as develop new tools and mechanisms to control an experiment. Finally, it has a distributed communication infrastructure supporting the scalable orchestration of thousands of distributed and potentially disconnected resources.

In addition to these novel features, OMF-F easily integrates with systems from other complementary testbed frameworks. Indeed, it interfaces with SFA to allow researchers to discover and provision resources to be used in their experiments. It is also compatible with measurement resources developed within instrumentation
and monitoring initiatives. OMF-F can also be paired with a web-based interface to act as a digital laboratory notebook and capture study cycles involving multiple experiments, through the use of additional wiki, versioning and statistical analysis capabilities.

Section 3 describes our objective and discusses the user and design requirements for OMF-F. From these specifications, we propose a resource model and an architecture for OMF-F in section 4. This architecture is composed of distributed entities and systems to support their interactions, which we further detail in the remainder of that section. In section 5, we focus on the event-based model used to describe experiments and some of the related capabilities. Most parts of OMF-F have been either deployed in existing testbeds or demonstrated at past events, such as the GENI Engineering Conferences (GEC). In section 6, we evaluate the performance of some of the key aspects of OMF-F when used on these existing testbeds. We then present some examples of real experiment demonstrators highlighting specific OMF-F features in section 7. Finally, we conclude this paper and discuss future work directions in Section 8.

2. Background and Related Work

2.1. OMF

OMF was originally developed to operate the ORBIT [6] wireless testbed. It has been extended in the recent years to support more types of resources and additional experimental features [4]. OMF has a dual purpose. First, it provides services to help testbed operators efficiently manage their resources, such as remote bootstrapping or disk image saving and loading. Second, it provides experimenters with software tools to describe experiments and automatically run them on a testbed. In this contribution, we build on the original OMF and significantly modify its core design to allow its operation across large scale distributed and federated testbeds.

2.2. Resource Discovery and Provisioning in Federated Testbeds

The Slice-based Federation Architecture (SFA) [2] is a framework which federates many GENI and FIRE testbeds. It defines the notion of a slice, which is a container abstraction for all the resources in a given experiment. Researchers are associated to slices and can use the SFA tools to discover, provision and include resources in their slice. SFA uses a hierarchical naming scheme to refer to entities and implements a distributed architectures to enable their interactions. It handles authentication and authorisation through the use of PKI-like certificates and credentials that are trackable along the trust chains between the users and institutions involved in the federation.

http://groups.geni.net/geni/wiki/GIMI
ProtoGENI [7] is another GENI-funded framework, which implements a variant of SFA for resource discovery and provisioning. It implements a SFA Clearinghouse entity acting as a centralised certificate repository and a registry for other entities (e.g. users, resources) within the federation. ProtoGENI also introduces a Slice Authority (SA) entity for each federated institution. Researchers affiliated with an institution use its SA to create, modify, destroy their slices. ProtoGENI currently federates many testbeds over multiple GENI-affiliated institutions in the USA.

Federation frameworks were also developed within the FIRE initiatives. For example, the Panlab project proposed a TEAGLE-based framework [8, 9] to allow the discovery and provision of resources on both PanLab and PlanetLab testbeds [10]. It relies on a Panlab-specific SFA interface, which interacts with the SFA-managed PlanetLab resources. Other contributions such as Top-Hat [11] federate different measurement infrastructures to provide researcher with valuable information during the discovery and selection of resources for their experiments.

While these contributions are essential to find and prepare resources from various institutions to be used in an experiment, they do not address the issue of describing and performing the experiment itself.

### 2.3. Experiment Definition and Orchestration

The GENI User Shell (GUSH) [12] allows GENI users to describe the execution of their experiments in an XML document. The user’s GUSH controller interprets that document and sends corresponding commands to a GUSH client running on each resources involved in the experiments. GUSH relies on direct unicast connections between the controller and clients and the availability of ssh. As such, it does not support experimental cases involving disconnections as found in mobile wireless meshes, or delay tolerant networks.

The Network Experimentation Programming Interface (NEPI) [13] provides tools to describe experiments as a box-model workflows. As previously a controller interprets this description, and sends related commands to a single entity per testbed responsible for executing them on all its resources. NEPI allows the orchestration of experiments in simulated, emulated, or testbed environments and their combinations. However, it currently supports a limited number of these environments.

Emulab [14] extends the ns-2<sup>5</sup> domain-specific language to allow users to define experiments to be performed on Emulab testbeds. This experiment orchestration system has many features such as detailed control of link emulation between resources, synchronisation barriers, or remote software install. In addition, it also has some tracing capabilities to collect link measurements. However, this system does not appear to be able to orchestrate experiments involving resources from different federated testbeds.

<sup>5</sup>http://nsnam.isi.edu/nsnam/
3. Objective, Scope, and Requirements

3.1. Objective

The goal of our OMF-F system is to enable the description and automatic orchestration of reproducible experiments involving resources from multiple testbeds under different administrative domains. Thus OMF-F aims at enabling the experimental scenario illustrated in Figure 1. Some researchers working on a study want to perform some experiments, for which they require resources (e.g. R1) from different testbeds or aggregates (e.g. A) provided by distinct administrative domains. These resources are defined in a ‘resource description’, and included in a ‘virtual aggregate’ (e.g. X) or slice [2, 8]. Users define their experiments in ‘experiment descriptions’ (e.g. Exp1), which are used to direct their execution over the resources within the virtual aggregate.

3.2. Scope

We loosely define federation as an agreement between participating entities representing experimenters and testbed providers on how to share resources and conduct experiments. In more technical terms, federations will be defined by allocation and usage policies as well as a mutually agreed set of APIs and mechanism for experimenters to discover, reserve, provision, and interact with the resources provided to the federation. Ideally, the users would interact with the federated aggregates as if they were a single one. Figure 2 shows a potential set of functional components to achieve such a unified view.
The Discovery and Provisioning component (D&P) holds the list of capabilities offered by the federation (such as available resources), and the list of entities involved in it (such as institutions, users). It accepts user requests for resources and handles reservations if required. It may offer advanced features such as constraint-based selection, e.g. at least $x$ resources of type $y$. This component also prepares resources for their use in experiments. This provisioning phase may involve different tasks depending on the resource’s type, e.g. creating a virtual machine, turning on a physical device, or loading an OS image. Section 2.2 present some of the systems implementing this functional component.

The Experiment Execution component (EE) provides experimenters with domain-specific constructs to describe their experiments. It processes these descriptions and ensures the provisioning of the resources and the overall experiment orchestration over the experiment’s life-time. This may include tasks, such as setting values of resource properties, performing tasks at certain time, or collecting specific measurements from or through resources. Thus the EE is critical in enabling reproducible experiments at large scales with resources from different aggregates. Indeed, a high level description allows unambiguous reproduction of experiments on similar or different resources (e.g. how does an ad hoc routing protocol perform over different types of wireless technologies). While an automated orchestration facilitates its execution on large number of distributed resources, as it frees the user from dealing with resource heterogeneity and large scale communication. Only a few contributions have focused on the EE component in federated environments. Indeed, the systems presented in section 2.3 have limited capabilities or do not support federations. OMF-F aims at filling that gap and focuses specifically on this EE function.

The UI component is the interface between the users and either or both the D&P and the EE components. It can be a text based user-shell [12], or a web-based software [7], or a set of web services for the recording of complete study cycles from experiment design to result analysis and curation [5].

3.3. User and Design Requirements

From an experimenter’s perspective, a framework supporting the entire experiment life-cycle should have the following characteristics.

*Ease of Use.* The abstractions and APIs to describe experiments and initiate their execution should have a low learning curve, and be backed-up by comprehensive documentation and tutorials.

*Scalable Complexity.* The description and execution of an experiment should not become more complex as the number of involved entities grows.

*Observable.* The user should be able to monitor and collect information from all involved resources and the experiment itself.

*Support Study Cycles.* An experiment is usually part of a larger study involving multiple design, execution, result analysis iterations. The framework should support these consecutive iterations, for example through interfaces with tools such as
versioning systems or statistical analysis software.

In addition to these user-oriented specifications, the design of such a framework should also adhere to the following testbed operator-oriented requirements.

**Generic.** It should be modular and offer high-level abstractions to support a large spectrum of resource types, e.g. core network & mobile personal devices, wireless sensors, cloud-hosted services.

**Secure.** It should provide secure authentication for involved entities (e.g. experimenters, resources) to support accountability, and a secure authorisation scheme to confirm the rights of entities requesting actions from other ones (e.g. user requesting the monitoring of a wireless channel).

**Scalable & Distributed.** It should scale to a large number of aggregates, resources, and concurrent experiments; and not rely on a central entity, which may become a failure or performance bottleneck, or source of conflict.

**Open.** Its design and implementation should be open source to promote its dissemination and contributions from third parties (e.g. support for new types of resources).

**Adopt, Adapt, Develop.** It should adopt standard technologies, potentially adapting them to meet its objectives, before developing new ones.

4. The OMF-F Architecture

4.1. Overview

![OMF-F Architecture Overview](image)

Figure 3: OMF-F Architecture Overview.

Figure 3 presents an overview of OMF-F architecture, where several entities interact to enable the federation scenario described in the previous Section 3. At the centre of this architecture is a distributed publish-and-subscribe messaging system (pub/sub), which is realised by a set of Peering Servers (PS). OMF-F use a topic-based messaging pattern where resources or groups of resources are represented by topics. Communication among all entities is achieved by publishing and subscribing to the respective topics.
Any resource is associated with a Resource Controller (RC). The RC is the proxy for own or more barebone resources. The RC subscribes to the topics associated with its resources and translates messages it receives from other entities to resource specific interactions. The RC normally also includes a policy component which first checks the validity of an incoming request.

An experimenter describes her experiment using a domain-specific language, and passes this description to an Experiment Controller (EC). This EC interprets the experiment description and uses the pub/sub system to send requests to the involved RCs and to receive reports from them on the experiment’s progress. These RCs instruct the resources to execute the tasks within these requests and relays their outcomes. If a resource is instrumented, it may collect filtered measurements as instructed by the RC based on the experiment description. These data are sent to measurement entities using the OMF Measurement Library (OML) [15], which can process them and store or forward them to other OML components. These OML entities are themselves resources, which understand the same communication protocol as the RCs and thus can also be organised and controlled via the EC and the experiment description.

The experimenter may use additional software (Vis & A) such as the LabWiki [5] to visualise the experiment’s progress and analyse its collected measurements. The remaining of this section details the main components of this architecture, and how they meet the requirements from Section 3.3.

4.2. Resource Model and Life-Cycle

Our first design decision is to consider every entity participating in an experiment as a resource, independent on who is providing it. A resource has a set of properties and an associated life-cycle (Figure 4). It communicates with other resources through well defined messages.

A new resource is created by an existing resource receiving a create message. It is initially in the inactive state, and may transition to the active state either immediately or at some later stage.

Given the large variety of resources there is no support for ’action’ commands, such as ’start’ of ’doX’. Instead, resources are requested, through a configure message, to adjust their internal operations, so that the observable properties reach a certain value. In other words, we request the outcome and leave the ’how’ to the resource. Some resources may only accept configuration requests in the inactive state, and some properties may only be set at creation time, as part of the create message. The request message asks a component to report on its status via
an *inform* message. Finally, a component is discarded when it receives a *release*
message.

Creating new resources by sending it to an existing one establishes a clear policy context in which we can decide if the request is valid or not. This results in a parent-child between the creator and the created. As a consequence and to maintain a proper accountability chain, a parent can only cease to exist when all its descendants have been released as well. In our current implementation, every resource maintains a list of its children and when receiving a *release* message it will forward it to its children as well. The initial release request will only succeed if all descendants have released themselves.

To support scalability we also introduce a *group* resource which maintains a set of other resources. In practice, this type of resource is simply a group messaging mechanism represented by a pub/sub topic. All members of a group are requested to also subscribe to the respective group topic and process the received messages accordingly. Given the one-way communication pattern of pub/sub there is no additional semantics associated with group resources. For instance, there is no implied guarantee that a message sent to a group will be received by all its members. In fact, that guarantee does not even exist for individual resources.

This resource model is an original feature of OMF-F compared to other orchestration framework [12, 14]. It is embodied in the RC and addresses our requirements for a generic, open, and observable framework.

The life-cycle model can be extended with sub-states and transitions for any given type of resources. For example, a mobile robot resource might have the sub-states *active/moving-forward* or *active/rotating*. In fact, the resource model is currently an optional part of the architecture as it is not reflected in the "standardised" components, such as the messaging protocol. There are no explicit messages to request a specific change in the life-cycle and it may only be observable if a resource provides it through a property. However, a well defined and broadly applied model simplifies the development of resource adapters.

In contrast, the communication protocol to interact with a resource is defined in an open document\(^6\), which allows third parties to develop their own RC or EC implementation to interact with RCs or ECs from other providers. Finally, the *request/inform* pair of messages allow the monitoring of the properties and states of any resource.

### 4.3. Resource Naming and Description

Each resource in OMF-F has an identifier of the form: `resID@aggregateID`, where `aggregateID` is unique, and `resID` is unique within an aggregate\(^7\).

OMF-F provides a domain-specific language to describe in a Topology the list of resources associated with a slice. This language is an extension of the orig-

\(^6\)http://mytestbed.net/projects/omf/wiki/ArchitecturalFoundation2ProtocolInteractions

\(^7\)The identifier of an aggregate resource is by convention: `aggregateID@aggregateID`
inal OMF language presented in [4]. Listing 1 shows a simple topology example, where two PC-based resources from different aggregates are connected via a L2 link provided by a resource in another aggregate. The user learned from the resource discovery phase, that the aggregate norbit.nicta.com.au offers a PC resource yellowb3, which has one interface connected to a L2 network provided by the aggregate foo.provider.net. This aggregate offers a resource l2controller to configure its provided L2 links. The user creates the resource description in Listing 1 with the reservation period and any other requirements captured in key/value parameters, and sends it to each aggregate. Although this resource discovery and provisioning phase is outside the scope of OMF-F as mentioned earlier, we will provide later in this section some details on how OMF-F can easily interface with tools implementing that phase (such as the SFA).

Other provisioning or orchestration frameworks use a similar resource description such as the RSpec [7, 12]. The OMF-F topology can be completely mapped into an RSpec and has the benefit of using the same language as the experiment description, thus facilitating the dynamic addition of resources within running experiments. This naming and description scheme can be applied to any type of resources and is based on an existing language, thus it meets our generic and adopt/adapt design requirements.

4.4. Publish and Subscribe Messaging

We implemented the OMF-F pub/sub messaging system using the XMPP protocol [16] and existing XMPP servers such as OpenFire. This system is composed of distributed servers peering with each other, and hosting topics. Authenticated clients can connect to a server, subscribe to any topics hosted by any servers and publish messages to them. XMPP’s server-to-server protocol ensures that a message published to a topic is forwarded to all of its subscribers regardless of which server they are connected to. Figure 5 shows the typical organisation of OMF-F’s pub/sub topics for a federated experiment scenario as described in Section 3, along with the subscribed entities from Figure 3.

A topic is identified by name@domainID. The domainID is the domain name of the server hosting that topic. Each aggregate must have an associated XMPP

Listing 1: Simple OMF-F Topology with optional key/value parameters partially showed for the first resource only.
server, with domainID = aggregateID (section 4.3), and a DNS entry to resolve domainID for server-to-server sessions. The name part is a unique identifier for the topic within a domain. While this identifier could be a fixed-length hash, for maintenance purpose we define it according to the OMF-F tree structure as illustrated on Figure 5. For example, the ID of topic res3 created for the experiment exp1 part of sliceX hosted by the server foo is: sliceX/exp1/res3@foo.net. The experimenters involved in sliceX will never deal with such a topic ID, they will only manipulate resources and OMF-F will handle any mapping to pub/sub topics.

Using discovery and provisioning tools [2, 7], the user selects a pub/sub server to host her study’s slice and creates the slice’s topics, e.g. sliceX@foo.net and sliceX/resources@foo.net. As part of the provisioning process, an additional topic is created for each resource assigned to the slice, e.g. sliceX/resources/res1@foo.net, and a RC is associated to this resource and subscribed to that topic.

In the experiment orchestration phase handled by OMF-F, the user’s EC (EC_{Exp1}) creates further topics related to the experiment (exp1), such as sliceX/exp1@foo.net, and sliceX/exp1/res1@foo.net. The EC_{Exp1} also creates the topics for all new resources that are created during an experiment execution, for example a new application instance or a new container for other resources (group i). For each provisioned resource to be used in exp1, such as res3, the EC_{Exp1} publishes on the slice’s topic (sliceX/resources/res3@foo.net) a configuration message requesting it to join exp1. Upon receiving that message, the corresponding RC (RC_3) subscribes to the topics of exp1 which are relevant to it, for example sliceX/exp1/res3@foo.net and sliceX/exp1/groupi@foo.net if res3 is included in the group i container. The RC is then ready to accept any commands related to exp1’s execution, which are published on these topics by the EC_{Exp1}.

Some additional subscriptions enable restricted message broadcasts and are not displayed on Figure 5 for legibility reason. For example, all entities involved in an experiment must also subscribe to the experiment’s topic (sliceX/exp1@foo.net) to respond to experiment-wide broadcast, such as an emergency shutdown.

Our use of a pub/sub messaging system with a structured topic map (Figure 5)
for OMF-F is novel compared to the communication schemes in other frameworks [12, 13]. It allows any types of entities to communicate in a scalable and distributed manner since it does not rely on a centralised architecture. It also meets our openness requirement as it is based on an standard [16], and it adopts established pub/sub server implementations. Finally, its asynchronous nature allows the orchestration of potentially disconnected resources as illustrated in Section 7.

4.5. Authentication and Authorisation

OMF-F needs to guarantee the publisher’s identity for all messages. While XMPP supports the authentication of connecting clients, misconfigured or compromised servers may not enforce it properly. Thus OMF-F instead uses end-to-end authentication based on the well established practice of digital signature backed by a KPI infrastructure. Each OMF-F entity has a set of public and private keys, signs its generated messages with its private key, and verifies the signatures of received messages with the originator’s public key. Public keys of a slice’s entities are exchanged at resource provisioning between the users and the resources, or obtained via a trusted scheme such as a certificate authority or web of trust. The former method is currently implemented in deployed OMF-F testbeds. The other method is under evaluation as it may not scale to a high experiment churn involving large resource numbers. PlanetLab [10] uses a similar key-based authentication scheme, which allowed OMF-F RCs to be deployed on its slivers. This allows OMF-F to orchestrate experiments with resources from both PlanetLab and other OMF-F testbeds as illustrated in Section 7.

An OMF-F entity also needs to verify that the originator of a message has sufficient rights for any enclosed requests. For example, although a user acquired the right to use a spectrum analyser to collect some data, it may be limited to use only some frequency ranges. We propose to attach a set of assertions or their resolvable references to each message, which establish the originator’s rights. In the previous example, the user’s request will have a first assertion from her institution confirming her affiliation to it, then a second assertion from the owner of the resource giving rights on a set of ranges to affiliates of that institution for a reserved time period. All assertions are signed by their originators and verified using the same key mechanism as above. Some assertions such as the last one in this example may be generated during the resource reservation and provisioning phase. This scheme can be considered a restricted, but light-weight variant of ABAC[17]. It is restricted as it does not allow for additional rules to be attached to assertions and it is light-weight as most assertions will only be passed by reference and is based on widely adopted industry standards.

4.6. Experiment and Resource Controllers

As previously mentioned, the EC is the entity responsible for orchestrating the experiment. It interprets an experiment description from the user, developed using a domain-specific language [4], and sends related commands to RCs using the pub/sub communication scheme and our defined resource protocol. In OMF-F,
we significantly extended the basic EC and RC entities of the original OMF [4] along the following four directions.

First the RC was redesigned to implement the resource model from Section 4.2. This generalises its source code to support different types of resources, such as virtual machines, mobile phone applications, or wireless sensors in a consistent manner. It also provides a reference implementation of our resource model and protocol, which serves as a base for custom RC implementations by third parties.

OMF-F experiments are now fully event-driven, i.e. the experimenters define events associated with tasks in their experiment. These events are synchronisation barriers based on either time or values of properties or measurements from resources, when their conditions are met the tasks associated to them are executed. An example event could be “when all traffic generator have sent 10Mps of data” and the associated tasks could be “pause them, decrease rate by X, and resume them”. We propose in Section 5.2 a formal model to represent events and in Section 5.1 our design approach for the new event-handling engine at the core of the OMF-F EC.

The communication stack of all original OMF entities were modified to support the new OMF-F messaging system and structure. This allows the EC and RCs to be on a different network domains, removes the need for a permanent connection between them, and provides scalable group communication. It also enables new experiment capabilities, such as disconnected experiments (e.g. a mobile resource temporarily out of range of an EC), or long-running surveys (e.g. EC connecting episodically to RCs in a x-month data collection).

Finally, the EC and RC entities were extended to support the authentication scheme described above, which allowed RCs to be readily deployed on PlanetLab, and enabled orchestration of federated experiments (Section 7).

4.7. Interface with Resource Discovery and Provisioning, and other tools

The use of a distributed pub/sub scheme as the core communication system of OMF-F allows it to easily interface with existing resource discovery and provisioning solutions. Indeed, the scheduler, registry and aggregate manager functionalities often defined in contributions from the GENI or FIRE initiatives can all be adapted to exchange messages using the OMF-F pub/sub system. The challenge remains then in the sequence of interactions between these entities and the OMF-F ones in order to provide a seamless transition between the discovery and provisioning phase to the experiment orchestration phase. We are currently collaborating with the developers of the NITOS Scheduler\(^\text{10}\) and the PlanetLab Europe SFA to address these challenges. As an example of this work in progress, we are developing an SFA-compliant OMF-F module, which should be released soon\(^\text{11}\).

\(^{10}\)http://nitlab.inf.uth.gr/NITlab/

\(^{11}\)http://omf.mytestbed.net/projects/omf/repository/show?rev=sfa
Similarly, this core communication system allowed the interface of OMF-F with a web-based digital laboratory notebook, which provides integrated functionalities allowing the record of research notes and experiment designs, the versioning of experiment descriptions and associated software, their automatic orchestration, and the processing of the result with a powerful analytical and statistical tool [5].

4.8. Distributed Instrumentation

OMF-F uses the existing OML system [15] to instrument resources and collect measurements. For an OMF-F EC executing an experiment, a OML-instrumented resources or measurement entities is handled just like any other resources. OMF-F and OML are two separately designed and developed frameworks, and recent OML development are presented in a separate contribution [18].

5. Describing and Orchestrating Experiments

5.1. Distributed Orchestration using Events

As we have mentioned before, experiment orchestration can be fully described by a set of event/action (S) declarations, or more formally as:

\[ S = (E, A) \]

We further define an event as some function \( e_i \in E \) over a set of observations on resources and time-out events \( T \). When the event \( e_i \) fires, an action \( a_i \in A \) is executed. Execution of \( a_i \) in the context of the messaging model described above, equates to issuing a set of messages \( M_i \). We can therefore describe a declaration \( s_i \) as follows:

\[ s_i = e_i(R_{obs}^i, T_i) \Rightarrow a_i() \Rightarrow M_i = \{m_{i,j}(R_{i,j}) : R_{i,j} \subseteq R_{act}^i\} \]

Each declaration \( s_i \) is described by an event \( e_i \) consuming observations from the resource set \( R_{obs}^i \). When the event \( e_i \) fires, action \( a_i \) is executed, resulting in a set of messages \( M_i \). Each message \( m_{i,j} \in M_i \) is addressed to a resource set \( R_{i,j} \). The union of all \( R_{i,j} \) is the set of resources \( R_{act}^i \) which declaration \( s_i \) may act upon. We also observe, that execution of declaration \( s_i \) only requires communication among the union of \( R_{obs}^i \) and \( R_{act}^i \).

In fact, we can further restrict that requirement by observing that an action \( a_i \) can be executed multiple times in response to a firing event \( e_i(t) \) as long as any resource \( r \in R_{act}^i \) will only receive one copy\(^{12}\) of each produced message \( m_{i,j}(t) \). In fact, we can expand on the above formalism to describe a system where the action

\^{12}This restriction is really only necessary for non-idempotent messages
blocks are co-located with each resource and all relevant messages are delivered locally only.

\[ S_i = \{ s_j \in S \mid i \in R_{j}^{\text{act}} \} \]

\[ O_i = \{ r_j \in R \mid s_k \in S_i \land r_j \in R_{j}^{\text{obs}} \} \]

In the above equation, \( S_i \) defines the set of declaration \( s \) which can potentially act on resource \( i \). In turn, \( O_i \) is the set of resources contributing observations to the events associated with the declaration in \( S_i \). We can therefore conclude that the correct orchestration of an experiment is assured if every resource \( r_i \) can at least receive messages from all the resources in \( O_i \), assuming that it can locally execute the relevant event and action operators. The ability to determine the “communication” set for a resource \( r \) allows us to reason about the impact of resources which may temporarily become disconnected from the messaging plane. This is especially important for experiments in the wireless, mobile, and sensor domain.

A special case occurs when a resource only depends on itself (\( O_i = \{ r_i \} \)) and therefore can successfully participate in an experiment even if disconnected from everyone else. In practice, our fine-grain modelling of resources normally means that multiple resources will be “hosted” by a single physical device. While note always true, we can usually assume that all resources on a device can communicate independent of network connectivity and therefore the “practical” special case is for a situation where the observation set of a resource \( r_{i,h} \) hosted by resource \( r_h \) is also hosted in its entirety on \( r_h \) (\( O_{i,h} = \{ r_j \in O_i \mid \text{hosted}(r_j) = r_h \} \)). The experiment in Figure 6, which we will describe in more detail shortly, is a simple but typical example. In this experiment, a resource is dynamically modified based on measurements taken on the same physical node. Other typical examples include declaration which control resources on a mobile device based on the mobile’s location.

5.2. Model for Experiment Description

We introduce a semi-stochastic analytical model to represent the orchestration of events in OMF-F experiments. This model is used to provide a bounded framework for forthcoming experimentations as well as to provide novel mathematical expressions of large scale distributed experiments.

In order to provide this analytical model, we propose the use of Petri Networks [19]. This family of tools allows to test for example deadlocks, liveness and boundedness of a specific system. In particular, we opted for the use of Generalised Stochastic Petri Networks (GSPN). This model allows us to describe and analyse the expected behaviour of any given OEDL scripts. In the remainder of this section, we apply, without loss of generality, this model to the case of an event-rich experiments.

In order to demonstrate the usability of this model we propose to apply it to a simple experiment consisting of three physical resources which in turn create three actors, namely a traffic source, a sink and an observer. In this scenario, once
the three physical nodes are up and operational, the sender and the receiver are exchanging data and the observer is reporting all the packets to the OML server. In
addition, we defined an event $E$ based on the measurement on the observer actor. This event is trigger if the measured parameter reaches a certain threshold. As a result, once this event is triggered the observer modifies its sampling rate to not interfere with the studied system.

Based on the aforementioned scenario, we established a Petri Net model as shown in Figure 6. In this figure, we represent the specific event “ALL_UP_AND_INSTALLED” by the transition $t_6$ which requires three tokens to be fired. The transitions $t_2$ to $t_4$ represent the temporal transition for a resource to be ready, while the transition $t_5$ represent a potential experiment failure by either a cancel from the user or a problem during resource setup.

After the event “ALL_UP_AND_INSTALLED” is triggered, we enter into the temporal script of OMF as illustrated in Listing 2. In particular, the sender and receiver actors are driven by temporal events while the observer actor is both driven by a temporal event and a measurement-based event.

Once the Petri Net associated to any given experiment is established, we can derive the embedded Markov Chain of the corresponding stochastic process. For sake of clarity, we present the Markov Chain associated to the aforementioned Petri Net in Appendix A.

The stochastic Markov Chain representing the different marking schemes on the Petri Net allows several quantitative analyses. In particular it can give temporal boundaries to any given experiment using sojourn time analysis. The Markov Chain analysis can also identify deadlock and liveness through the establishment of the steady state distribution and the identification of the transient and absorbing states.

As a preliminary conclusion, we claim that, based on the event-based architecture of OMF, it is possible to build analytical model of any kind of experiments using Generalised Stochastic Petri Networks. We are currently working on the generalisation of the Petri Net models in order to offer a comprehensible algebra for the automation of this model generation.

6. Evaluation

We performed two sets of experiments to quantify the performance of the pub/sub communication scheme used by OMF-F and the responsiveness of its event-based experiment orchestration.

6.1. Evaluation of the OMF-F communication scheme

The objective of this first experiment set is to quantify the delay between the time when the EC publishes a command towards a group of RCs and the time when all of these RCs publishes back that they have executed the command. In that regard, we used OML [15] to instrument the EC with two measurement points (MP), one at message publishing and the other at message reception. Each MP records a timestamp and the message content, so we can match received replies corresponding to sent commands.
In this experiment set, the EC asks a group of RCs to start an application with negligible startup delay. As soon as this is done, the RC published a message back to the EC. We ran this experiment on both a local testbed (TB1) with ORBIT-like resources and the PlanetLab testbed (TB2). We varied the number of involved resources $N$ from 1 to 30 on TB1 and from 1 to 190 on TB2, and ran 10 trials for each $N$ value\(^{13}\).

Figure 7 shows the mean response delay $D$ over all resources for both testbeds and different number of resources $N$. $D$ seems to grow linearly with the number of involved resources, with $D < 1.3s$ for $N = 190$ on TB2. A linear regression using this data gives an estimated slope of $a = 5.17 \times 10^{-3}$ (thus $D$ should remain low for large $N$ values), and predicts $D = 2.85$ for $N = 500$ and $D = 5.43$ for $N = 1000$.

Figure 8 shows the ascending mean response delay $d$ for each resource when $N = 30$ on TB1 and $N = 190$ on TB2. It shows that within a given trial, $d$ seems to

\(^{13}\)Originally $N = 200$ for TB2, however when inspecting the data we noticed faulty PlanetLab resources on some of the trials and decided to remove them from all trials.
6.2. Evaluation of the OMF-F event engine

The goal of this second set of experiments is to quantify the responsiveness of OMF-F event-handling engine. Thus we measure the delay between the moment when the EC detects that an event happens (i.e. a synchronisation barrier is triggered) and the moment when its associated tasks are performed.

In this set of experiments, 100 random resources each run a single application. We define an event, which monitors the status of these applications, and which triggers if any application dies (i.e. quits with an non-zero error code). When such an event happens, the associated tasks are to select some replacement resources from a list of stand-by ones and then start a new application instance on these replacements. We measure the duration between the first detected failed application until 100 application instances are running again over the entire experiment. This duration is composed of the time it takes for the failures to be detected (which includes the 1s pooling time of the event-engine), the EC and RCs processing times, the communication time between them, and the startup time of an application instance\(^1\). This is a classic failure recovery scenario likely to occur in large scale experiments with unreliable resources. We ran 10 trials of that experiment on 200 PlanetLab resources. For each trial, we kill the running application on 50 randomly selected resources after 150s of runtime.

Figure 9 shows the recovery time for a single trial of the above experiment and for all of the 10 trials. It shows that the EC event-handling scheme is responsive

---

\(^1\) The application here is a simple infinite while loop, thus startup time is negligible.
as it needs on average about 11.18s to recover from the failure of 50 application instances after the first detected failure. This event engine is currently implemented using a 1s active pooling, i.e. every 1s the EC pools information relevant to the event and decides if it triggered or not. This is done using “Green Threads” in Ruby 1.8, thus its performance depends on the scheduling efficiency of the Ruby virtual machine.

7. Case Studies

This section presents some examples of real research experiment, which were performed on federated testbeds using the OMF-F framework. This experiments were also demonstrated live during the GENI Engineering Conferences (GEC).

7.1. Distributed Resolution of the Obstacle Problem across Testbeds

This experiment was part of a research work investigating novel peer-to-peer scheme to solve problems (e.g. the classic Obstacle Problem) in a distributed manner [20]. It was also presented as a federation demonstrator during the 7th GEC. In this scheme, the P2P application built location-based clusters in order to configure adequate communication characteristics (reliability or not for example) for a better resolution of large scale numerical problems using asynchronous iterative steps. In this experiment, the researchers used two sets of solvers, one deployed on the Orbit aggregate and the other on the PlanetLab aggregate. Figure 10 presents the setup of this federated experiment. This experiment demonstrated the following features of OMF-F, as described in Section 4: the PKI-based authentication scheme, the use of the message scheme across domains, and the extension of the orchestration tools to control heterogeneous (PlanetLab, Orbit) resources.

7.2. ParkNet Demonstration over Three Federated Domains

This experiment was developed within the ParkNet project [21], which aimed at collecting and processing parking space statistics via sensor-enabled vehicles [21]. It involved three aggregates, namely the Poly NYU aggregate providing WiMAX connectivity the vehicles driving in Brooklyn, the Orbit aggregate hosting the ParkNet cloud, and a GENI aggregate providing backbone connectivity.
OMF-F was used to design and execute the experiments across these federated aggregates. Figure 11 presents the setup of this experiment, which was demonstrated at the 9th GEC. This experiment illustrates many features of OMF-F, which are the support for highly mobile wireless resources, for potential disconnections, and for even more heterogeneous resources (e.g., cars, sensors, video devices). Finally, it also demonstrated OMF-F’s capability to drive a highly distributed experiment with geographically distant resources.

8. Conclusion

We proposed OMF-F, a framework which allows the definition and orchestration of networking experiments over shared resources provided by different federated administrative domains. Its main three features are 1) the support for rich, event-based experiment descriptions in a domain-specific language; 2) the specific definitions of how it models resources and how to communicate with them over an asynchronous publish-subscribe system to enable automatic orchestration at large scale; and 3) the capability to interface with existing resource discovery and provisioning tools [2], and experiment cycle and data management tools [5].

Within the global initiatives on future Internet testbeds, many contributions have focused on the critical tasks of discovering and provisioning resources within a federation of testbeds. However, few groups have focused on the equally important tasks of developing experiments and executing them in that same federated environment. Such a system would foster reproducible and scientifically sound experimental research and facilitate peer-review. OMF-F focuses on that challenge and provides a complete system to define and orchestrate large scale experiments over federated testbeds.

In that regard, we made three main contributions in this paper. First we provided a detailed description of the models used in OMF-F, its architecture, and its involved entities. We complemented this with overviews of OMF-F’s open resource protocol and experiment definition language, with links to their complete online descriptions. Second, we presented an evaluation of two key components of OMF-F, namely its underlying messaging and event-handling systems. This evaluation showed that they are scalable to a high number of resources (∼1k) and
responsive (∼11s response time to 50% failure). Then, we presented a couple of real examples of OMF-F being deployed at different federated domains and used to define and execute concrete interesting research experiments. These case-studies demonstrate the operation and ease-of-use of OMF-F in real-life.

Our future plans include completing the authorisation scheme based on assertion sets, improving the performance of OMF-F’s messaging and event-handling systems, polishing the current implementation and documentation to attract more users (not only experimenters, but also developers contributing supports of new resources), and extending the existing deployment base. We are collaborating with other teams within both the FIRE and GENI initiatives to provide seamless OMF-F interface with various SFA implementations, and with measurement analysis, management and curation systems. Finally, we plan to regularly review OMF-F’s core model, design, and mechanisms based on the insights gained through our involvements with FIRE and GENI projects and the feedbacks from enthusiastic OMF-F users.
References

URL http://issuu.com/world.bank.publications/docs/9780821387092


URL http://portal.acm.org/citation.cfm?id=1713254.1713267


Appendix A. Petri Net embedded Markov Chain

Figure A.12: Embedded Markov of the GSPN.

Figure A.12 presents the embedded Markov Chain for the Petri Net from Figure 6. We simplified this Petri Net by removing the transition $t_5$ (i.e. the potential failure case) as it would have added many states in the Markov chain which would have resulted in an obliterated presentation of the problem. Nevertheless, without loss of generality we can analyse the experiment represented in Figure 6 using the Markov Chain in Figure A.12.

The embedded Markov Chain gives us the transition matrix $P$. In this matrix $f_i(.)$ represents the function on the line $i$ such as $f_i(.) = 1 - \sum_{j=1}^{n} p_{ij}$.

Based on the transition matrix, we would be able to extract the sojourn time for every marking as well as the steady state distribution $\pi$. Nevertheless the steady state distribution in this particular case is not very interesting as it contains a single absorbing state.