# Architecture & Implementation of a Testbeds Repository

Eamonn Power, Zohra Boudjemil, Shane Fox
Telecommunications Software & Systems Group,
Waterford Institute of Technology,
Waterford, Ireland

## Abstract

Federations of testbeds have an intrinsic characteristic: they are composed of heterogeneous resources and services residing in different geographic locations. To support description of these resources, there is a need for a common repository to be made available to the overall Panlab testbed federation architecture. In creating and integrating this repository, a number of design decisions were made to produce a flexible solution. We describe the structure of the repository software architecture and discuss the information model and derived data model. Finally, implementation details are further described at a functional level based on observations during the work carried out.

## 1 Introduction

The Internet has evolved rapidly from a communications system to an almost ubiquitous part of every day modern life, supporting everything from the modern economy to society. It has evolved in a haphazard manor out of necessity and has managed to just about meet the requirements placed on it. However, in order for further advancements to be made, especially within the Future Internet, radical changes will need to occur (FIRE 2009).

To facilitate these changes, the Future Internet will require extensive testing facilities for new architectures, protocols and services to be developed. This testing will require highly hetrogenous, scalable experimental facilities.

FIRE is a European initiative (Future Internet Research & Experimentation) (FIRE 2009), that has been setup for just this purpose. Its vision is to create a multidisciplinary research environment for investigating and experimentally validating highly innovative and revolutionary ideas for new networking and service paradigms.

There are a number of projects under the FIRE initiative but two in particular are focused on network testing. These are Onelab (OneLab 2009) and Panlab (PanLab 2008). Onelab is building upon the PlanetLab project (PlanetLab 2007). It aims to extend PlanetLab, which was focused on provisioning homogenous computing resources for experimentation with the network and application layers. OneLab will also include new monitoring tools and federate PlanetLab Europe with other PlanetLabs worldwide. Panlab has been focused on establishing a framework for federating testbeds. The framework will enable users to create composite testbeds across multiple administrative domains that are specific to the testing needs of the application or service to be tested. Panlab target a usersbase of commercial and academic organisations that are developing communication systems, applications and services. It provides access to a communication environment for pre-commercial product testing, benchmarking and interoperability testing that may otherwise be too expensive for these organisations.

This paper focuses on work done in creating and implementing the architecture of the testbed federation repository. The repository is part of the Panlab federation architecture and provides a managed storage area for the set of tools provided by the federation architecture.

## 2 Background

Panlab is a federation of testbeds. The Panlab architecture defines a number of architectural components for federated testbeds (Gavras et al. 2009) (Tranoris et al. 2009) (See Figure 1). Panlab has a common management entity(Teagle) [1]. Each testbed has a Panlab Testbed Manager component (PTM). These PTMs have access to a collection of resources which are controlled by resource-specific Resource Adapters (RA). The RAs allow the testbed resource to be exposed as controllable services to Teagle. This common management entity contains a number of components concerned with the overall coordination of the federation. It provides a tool set for testbed resource management from a testbed provider point of view, provides a tool to create Virtual Customer Testbeds (VCT) from a customer/client perspective and provides a common storage facility (Panlab Repository) from a Panlab developer perspective.

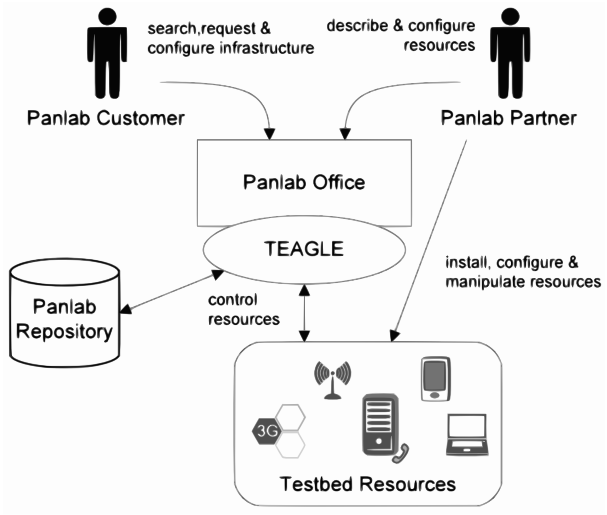


Figure 1: Panlab Architecture Overview

When designing the Panlab Repository, a number of existing architectures and methods were examined which enabled the decoupling of client applications from the data access mechanism. Service Data Objects (SDO) (Goodson & Preotiuc-Pietro 2005) and XML-RPC (Laurent et al. 2001) appeared as good choices. They both used HTTP for transfer purposes and there are a number of supporting implementation libraries and frameworks already available. However, SDO required a large amount of supporting technology and significant buy in from client application developers. XML-RPC, while easy to work with, lead to the re-creation of interfaces for data management purposes with the addition of each entity. This has been observed as one side-effect of RPC and SOAP style development. The final design decisions are detailed in Section 3.

Section 4 looks at the usage of the information model and data models. Section 5 looks at the implementation issues based on the architecture and model choices presented.

## 3 Software Architecture

In the Panlab federation architecture, Teagle provides a set of tools for interacting with the federation resources (Wahle et al. 2009). Applications within this tool set require common, managed storage to enable the sharing and reuse of data. This storage entity is designated the Panlab Repository.

For example, through the VCT Tool, the user can compose a testbed specific to their requirements via a simple drag-and-drop mechanism. They can connect and configure their chosen testbed resources. Teagle will then automatically orchestrate the testbed setup across the multiple administrative domains by reserving and configuring the required resources.

[1] See http://fire-teagle.org

The Repository provides access to coherent data gathered from diverse sources of the Panlab architecture components and provides it through a standards-based interface to an equally diverse set of applications and architectural components.

With these requirements in mind, a RESTful (REpresentation State Transfer) architecture approach was taken. With REST the emphasis is on the diversity of resources, each having their own identifier in the form of a URL so accessing and tracking the resources is inherently managed. The constraints developed in Chapter 5 of Fielding's thesis (Fielding 1999) provide a framework which go toward explaining the architecture of the repository and the benefits of using a RESTful architecture.

The software architecture of the repository, as seen in Figure 2, is composed of a number of applications running as contexts on an application server. Each application has its own data storage facility, set of business logic and exposes a HTTP-based RESTful interface with a number of REST resources. This shows the repository as a server entity in the Client - Server architectural style. Each application is also capable of being the client of another application allowing the reuse of the RESTful resources. This separation of concerns from client applications clears up the role the repository plays in the overall Panlab architecture. The repository is only concerned with the storage and retrieval of data for client applications. Other tasks such as orchestration script generation, resource status updating or VCT management are carried out by specifically designed applications using the repository for data storage. This allows these tools to develop independently of the repository but maintain a common data model.

The communication with the repository is stateless. Each request/response pair contains the information needed for the client and server to process the data. The session state and context data are managed by each client application thus allowing large scale and fault tolerant operation.

The ability of the architecture to support caching allows future needs for scale to be easily addressed without any additional special considerations. Server-side caching can be used to improve interaction with commonly accessed resources without the need to regenerate those resources each time they are requested.

This leads to the use of a layered system where each entity is only aware of its immediate layer. This allows the implementation of the repository architecture to vary as required (e.g. inclusion of caching or load balancing) without modification to client applications being required to accommodate these changes.

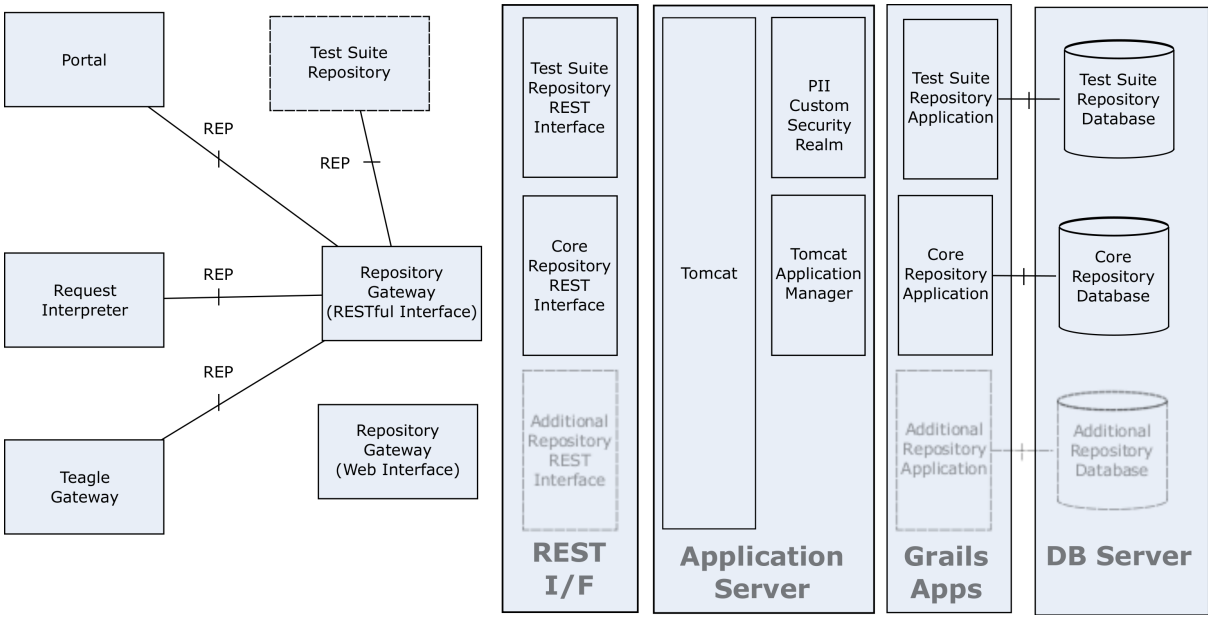


Figure 2: Panlab Repository Architecture Implementation

This underlying architecture can be used to support any type of data model specification and in Section 4, the data model specified for federated testbed resources is examined.

# 4 Panlab Repository Data Model

Federations of testbeds have an intrinsic characteristic: they are composed of heterogeneous resources and services residing in different geographic locations. Panlab aims to allow each of the testbed providers to connect their testbed resources to the Panlab federation by introducing interoperability and harmonised access to heterogeneous data sources. This means that a large number of unknown heterogeneous testbed resources are to be managed, so that each resource can be stored, searched, reserved and configured to function in a custom testbed setup. This large number of heterogeneous resources will invariably use numerous configuration methods and have different functionalities thus making it difficult for a single system to manage.

To automatically orchestrate a testbed setup, Teagle has to process detailed knowledge of the testbed resources and the dependencies between these resources, e.g. the pre/post conditions constraining how resources can be connected to each other. Hence, these descriptions have to be meaningful so that a management system has the required information to be able to reason in a manner necessary to orchestrate the specified testbed.

An information model could be used to provide the supporting infrastructure for these requirements. It firstly enables the precise definition of both the testbed resources and their relationships in a common form, thus alleviating the interoperability issues of the heterogeneous resources.

An information model can be defined as a "representation of the characteristics and behaviour of a component or system independent of vendor, platform, language, and repository" (Strassner et al. 2007). The information model is an abstract representation of entities which can be real objects such as devices in a network or logical such as the entities used in a billing system. The information model provides shareable, stable, and organised structure of information for a specific domain. This level of abstraction allows homogeneous specifications of different types of resources and their relationships in the context of federated testbeds.

The DEN-ng (Strassner 2009) (Strassner et al. 2007) was conceived to enable the construction of autonomic networks. It is a UML-complaint object- oriented information model that describes network elements as managed entities, their characteristics, their relationships and most importantly support the description of their status, and life-cycle for a management framework. DEN-ng adds a business view to the model by providing business concepts to the network and the system. Also, DEN-ng is primarily concerned with the management of policy from a customer right the way through to the configuration of network and system devices (Strassner et al. 2007) (Strassner 2008).

Panlab has chosen the DEN-ng information model as the reference to the PanLab Data Model because of three key properties. First, it is the only model that was built from the beginning to support patterns, specifically the role pattern. This provides inherent extensibility. Second, it uses a number of powerful abstractions (Strassner 2008) that dramatically simplify management. Third, it is the only model whose design philosophy was to support orchestration of services through their entire life-cycle. Most models are limited to modelling the current state of an entity; DEN-ng models the life-cycle of an entity using state machines.

As part of the modelling task, a common Data Model has been defined that captures all necessary shared data in the context of the orchestrated testbeds. The definition of the Panlab Data model is based on the DEN-ng information model (Strassner 2009).

The Panlab modelling domain focuses on the resources of a testbed. Based on the DEN-ng specifications, a resource is defined in the Data model as a manageable entity supporting and delivering services. It can be a *PhysicalResource* or *LogicalResource*. Each testbed provider can define a list of resource instances as specific subtypes based on the Resource model.

Management of the resources involves configuration and monitoring to capture performance and usage. In the context of PanLab, the configuration of resources are an important requirement to support the effective deployment of the testbed as requested and defined by a customer using the VCT. The resource configuration is captured by the configuration entities instance of the *ConfigurationInfo* class. The subtypes of this class allows the definition of configuration data as a collection of parameters or a more complex composition of configuration data.

Following the description of the VCT Tool, the customer is able to design experimental environments composed of one or more resources offered by the Panlab federation. This experimental environment description is encapsulated by an instance of the *VCT* class type, which is defined as a Product type.

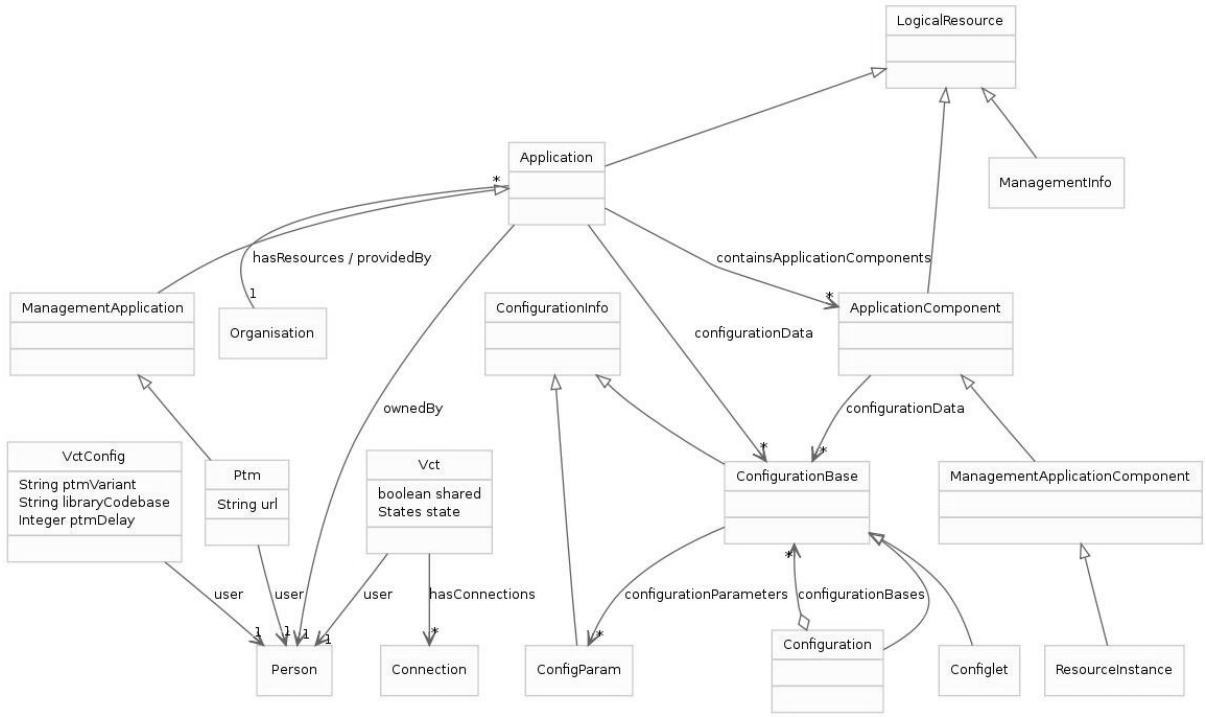


Figure 3: Panlab Repository Data Model Overview

The implementation of the PanLab data model is supported by the Grails Object Relational Mapping (GORM)language as detailed in Section 5.

# 5 Implementation

As defined in Section 3, the RESTful architecture specifies a uniform interface to resources. This uniform interface allows access to data for other applications without the need for special code as the only permitted operations are the HTTP standard methods of GET, POST, PUT and DELETE (additionally HEAD and OPTIONS). This was especially beneficial as the testbed repository would be integrated with many other applications within the Panlab system, such as the Teagle Portal, the VCT tool and the PTMs. This was useful as all requirements were not explicitly clear and flexibility was needed.

The Grails Framework (Grails 2010) was chosen as the platform to implement the repository applications. Grails is an open-source web application framework that leverages the Groovy language and complements Java web development. The framework has numerous facilities that are exploited for the development of the repository application. Firstly, the framework is an MVC framework thus separating the application's domain model from the user interface (view) and the controller(business) logic. This allowed the focus to be placed on creation of the domain model which is the implementation of the Panlab data model. This forms the core of the application.

The framework has support for implementing REST resources in the form of URL mappings to controller methods. The default mapping in Grails is */controller/action/id* but this can be configured for a RESTful interface by allowing the HTTP methods (GET, POST, PUT and DELETE) to be mapped to their REST equivalents. This is configured in *UrlMappings.groovy*.

An example of a GET request for a person resource from the repository application is listed below:

```
GET /repository/rest/person/1

<?xml version="1.0" encoding="UTF-8"?>
<person id="1">
    <personRoles>
```

```
                <personRole id="1"/>
        </personRoles>
        <userName>jbloggs</userName>
        <fullName>Joe Bloggs</fullName>
        <emails>
                <email id="1"/>
        </emails>
        <password>5gs0f33516b8276d758a9a7274737417</password>
        <organisations>
                <organisation id="1"/>
        </organisations>
</person>
```

The framework has many support mechanisms for developing web applications. In the implementation of the repository application, it proved invaluable for its ability to map the object model onto its relational representation. The framework builds upon Hibernate (Hibernate 2010), a widely used object relational mapping (ORM) library to create an easy to use ORM called Grails Object Relational Mapper (GORM). The advantage in using a language like GORM is that it uses the convention of the domain classes themselves to perform the mappings, whereas many ORMs require the mappings to be described in an external form. This advantage allows the developer to focus on the data model rather than its representation at a database level.

The scaffolding and templating mechanism of the framework are used to generate the views and controllers for the application. The scaffolding generated the CRUD [2] operations for the controllers, however many of these had to be manually altered as Grails could not handle many of the complex relations that were inherent in the domain model. The views were largely a by-product of the generation process, as they were not strictly required for the repository application but they proved a very useful resource for basic testing and also provided a visual aid for understanding the model.

XML is the suitable data language for representing the data objects and a means of translating this language so it could be stored and retrieved via the relational database was required. The framework provides a solution for this through its XML Marshalling facility. This allows XML to be read and processed as input and written to output as responses.

There are a few techniques for generating XML responses in the framework. The first is the render method. This method can be passed a block of code which, using a builder, can generate XML. An alternative method is the automatic XML marshalling of domain classes to XML via converters. The *grails.converters* package has a render method that is both flexible and has an easy to use syntax. In the implementation, it is used in all the REST controller methods to produce the XML responses.

To support POST and PUT via XML, the framework provides a data binding mechanism that can 'bind' incoming request parameters onto the properties of an object. Grails utilises the Spring (Spring 2010) framework to provide its own support for data binding. This implementation simplifies the binding process to passing a params object into the domain class' constructor. It automatically recognises that an attempt to bind the request parameters and implements the binding. In developing the repository application this mechanism is used extensively.

During development of the repository application, unit and functional testing was implemented. The framework provides support for testing through Groovy tests which extends JUnit's TestCase with GroovyTestCase (GroovyTestCase 2008). This primarily provides assert methods for Groovy. The RestClient plugin (RestClient 2010) is also used in the testing of the repository application as it allows the developer to create RESTful requests and handle response data.

After successful testing and running of the repository application, the application was deployed onto an Apache Tomcat (Tomcat 2010) application server with a MySQL (MySQL 2010) database. A custom security realm had to be implemented to provide a first level of security for the repository, which is supported by the data model of the repository.

[2]Create Read Update Delete

# 6 Conclusion and Future Work

In this paper, the position of the Panlab Repository in the overall Panlab architecture was described and related material was cited. Following this, the internal structure of the repository software architecture was described. The content that is managed by this software was elaborated on by reference to the information model and derived data model. Finally, implementation details were further described at a functional level based on observations during the work carried out.

Further aspects of the RESTful architecture style will be explored as the implementation of the repository expands. For example, there is currently no requirement for "code-on-demand" (Fielding 1999) from the repository. However, the VCT tool is delivered as a rich internet application from the Teagle Portal. Other application components may be stored and delivered from the repository as the application portfolio accessing the repository expands. The capability of supplying additional features through this mechanism at a later stage becomes useful in this case.

The final stage implementation work on expanding and improving the searching facilities provided by the repository are moving toward the integration of off-the-shelf searching frameworks. This would enable decoupling of the repository applications from the proposed search mechanism.

Finally, the data model itself is being expanded through additional repository applications. This allows the inclusion of new entities which can be represented as RESTful resources.

# 7 Acknowledgment

Parts of this paper are based on work and ideas generated in the course of the project PII (PanLab 2008), which receives funding from the European Commissions Seventh Framework Programme (FP7).

# References

Fielding, R. T. (1999), Software architectural styles for network-based applications, Technical report.

FIRE (2009), 'Future internet research & experimetation (fire) initiative'. http://cordis.europa.eu/fp7/ict/fire/.

Gavras, A., Hrasnica, H., Wahle, S., Lozano, D., Mischler, D. & Denazis, S. (2009), *Towards the Future Internet - A European Research Perspective*, IOS Press, chapter Control of Resources in Pan-European Testbed Federation.
**URL:** *http://www.booksonline.iospress.nl/Content/View.aspx?piid=16465*

Goodson, K. & Preotiuc-Pietro, R. (2005), Jsr 235: Service data objects, Technical report, IBM & BEA proposed request to the Java Community Process. http://jcp.org/en/jsr/detail?id=235.

Grails (2010), 'Grails home page'. http://www.grails.org/.

GroovyTestCase (2008), 'Groovytestcase documentation'. http://groovy.codehaus.org/Unit+Testing.

Hibernate (2010), 'Hibernate home page'. http://www.hibernate.org/.

Laurent, S. S., Dumbill, E. & Johnston, J. (2001), *Programming Web Services with XML-RPC (O'Reilly Internet Series)*, O'Reilly Media.

MySQL (2010), 'Mysql home page'. http://www.mysql.com/.

OneLab (2009), 'Onelab project website'. http://www.onelab.eu/.

PanLab (2008), 'Panlab project website'. http://www.panlab.net/.

PlanetLab (2007), 'Planetlab website'. http://www.planet-lab.org.

RestClient (2010), 'Restclient documentation'. http://groovy.codehaus.org/modules/http-builder/apidocs/groovyx/net/http/RESTClient.html.

Spring (2010), 'Spring source home'. http://www.springsource.org/.

Strassner, J. (2008), 'The role of standardization in future autonomic communication systems', *Engineering of Autonomic and Autonomous Systems, 2008. EASE 2008. Fifth IEEE Workshop on* pp. 165–173.

Strassner, J. (2009), 'Information model - DEN-ng'. http://www.autonomic-management.org/denng/index.php.

Strassner, J., Lehtihet, E. & Agoulmine, N. (2007), 'Focale - a novel autonomic computing architecture: extended version', *ITSSA journal* .

Tomcat (2010), 'Apache tomcat home page'. http://tomcat.apache.org/.

Tranoris, C., Denazis, S. & Gavras, A. (2009), A workflow on the dynamic composition and distribution of orchestration for testbed provisioning, *in* 'NEM Summit 2009 proceedings'.

Wahle, S., Magedanz, T., Gavras, A., Hrasnica, H. & Denazis, S. (2009), Technical Infrastructure for a Pan-European Federation of Testbeds, *in* 'TridentCom 2009 proceedings'.