Deliverable Title	**D3.1.1 – Teagle assessment and TEFIS portal specifications**
Deliverable Lead:	Fraunhofer
Related Work package:	WP 3
Author(s):	Ainhoa Gracia, Franca Perrina, Brian Pickering, Florian Schreiner, Sebastian Wahle & other TEFIS partners
Dissemination level:	Public
Due submission date:	30/09/2010
Actual submission:	31/10/2010
Project Number	258142
Instrument:	IP
Start date of Project:	01/06/2010
Duration:	30 months
Project coordinator:	THALES

Abstract	This document covers the assessment of the FP7 project PII results, namely Teagle, with respect to the TEFIS architecture and requirements defined as part of WP2. Further, the document specifies the TEFIS portal functionalities and requirements based on the use cases and experiments foreseen by the TEFIS consortium. It is expected that new building blocks requested by TEFIS users will be added later, based on the feedback received from the open call experiments.

Project funded by the European Commission under the 7th European Framework Programme for RTD - ICT theme of the Cooperation Programme.

License

Table of Content

Table of Figures

List of Acronyms

API	Application Programming Interface
CRUD	create, read, update, delete operations
DEN-ng	Directory Enabled Networks New Generation
HTTP	Hypertext Transfer Protocol
MVC	Model View Controller Pattern
OE	Orchestration Engine
PE	Policy Engine
PTM	Panlab Testbed Manager
RA	Resource Adaptor
REST	Representational State Transfer
RP	Request Processor
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
TGW	Teagle Gateway
URL	Uniform Resource Locator
VCT	Virtual Customer Testbed
XML	Extensible Markup Language
XML-RPC	XML Remote Procedure Call

Executive Summary

This document provides a summary and assessment of FP7 PII results, namely Teagle, with respect to the TEFIS architecture and requirements defined as part of WP2. On the other hand it specifies the TEFIS portal functionalities and requirements based on the use cases and overall TEFIS architecture as defined in D2.1.1.

The study of different and heterogeneous resource repositories, considering TEFIS requirements, led to the selection of Teagle as the main core for the management of the testbeds and their linked resources. The Federation Model on which Teagle lies, is considered as an optimum scenario for modeling the requirements that TEFIS resource management implies. A comprehensive study of the mapping between TEFIS and Teagle has demonstrated the suitability of that selection. Furthermore, synergies from other European projects that worked or are working on related topics can be deduced, such as FP6 SSA Panlab project, that developed the entire theoretical model for the Federation scenario, and FP7 PII project, where a deep development of Teagle has been achieved.

All the functions for resource management and booking provided by Teagle are maintained and used for TEFIS project. Nevertheless, in order to provide a more consistent integration with the rest of the portal, and also with the idea of hiding the most complex technical aspects to the experimenter, the majority of the functions that the VCT tool provides are not directly exposed to them, but carried out and managed by the Experiment Manager according to each particular experiment configuration.

As long as the design of the TEFIS Portal concerns, one of the main issues that have been considered is the difference of skill levels among the potential users of the portal and the various roles they could play. Considering that the portal will suppose a single access point for every TEFIS user, it must provide enough flexibility in order to satisfy the necessities of both unskilled and expert users. What is more, it shall also provide the different tools which will allow TEFIS administrators and testbed providers to manage their facilities (resources, TEFIS servers and repositories, data storage, …). It is expected that new building blocks requested by TEFIS users will be added later, based on the feedback received from the open call experiments. Therefore, the portal will be implemented using a modular design that shall allow the integration of new functionalities and extensions

Please note: parts of this deliverable cite text that has been published before by the FP7 PII consortium. This has been done for the reader's convenience only. The objective was to provide a condensed overview in a single document to ease the understanding of the complex Teagle framework. Permission for re-publishing this text has been obtained from the PII consortium.

1 Federation Model

This section introduces the Federation Model used by Teagle to federate heterogeneous resources across various administrative domains.

1.1 Model Entities

A federation is generally understood to be an organization within which smaller divisions have some internal autonomy (Oxford definition). Merriam-Webster defines federal as: (1) formed by a compact between political units that surrender their individual sovereignty to a central authority but retain limited residuary powers of government; (2) of or constituting a form of government in which power is distributed between a central authority and a number of constituent territorial units. [1]

This concept, clearly stemming from a political background, is transferred to a technical domain to enable the combination of infrastructural network resources and services of more than one administrative domain which enhances significantly the utility of the infrastructures. Federation enables access to additional resources increasing scale, or access to resources with unique properties to enrich experiments. [1]

Several model entities are involved which are shown in the figure below.

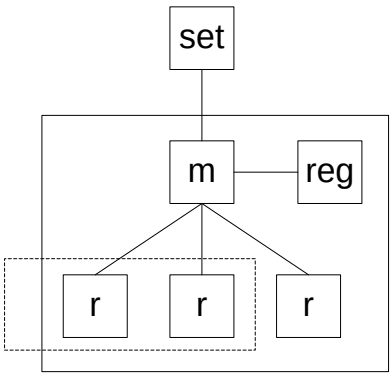

Figure 1: Federation Model entities [1]

The entities are explained in the following as defined by [1]:

- Resources (r): The model abstracts from concrete resource types. A resource can be anything that can be controlled by software. Examples are: physical and virtual machines, software packages, dedicated hardware such as sensors and routers, as well as abstract constructs such as for example domains, accounts, databases, and identities. Resources may contain other (child) resources.
- Domain manager (m): software that controls resources inside an administrative domain. It exposes resource management functionalities at the border of a domain and connects to a

resource registry. Supported operations on resources are typically the CRUD (create, read, update, delete) commands for controlling resources via a common interface. Proper security mechanisms and policies need to be supported in order to protect administrative domains from resource misuse.

- Registry (reg): holds data records for domain resources. Registries may or may not expose an interface to (external) setup utilities (set).
- Creation / setup tool (set): resides within or outside of a domain and communicates with domain managers and registries. Set utilities provide a user interface for the configuration, deployment, and monitoring of virtual resource groupings.
- Virtual grouping of resources (dotted rectangle): each administrative domain enables access to a number of resources. Jointly, all administrative domains provide a large pool of resources. Experiments usually require only a subset of the total resources that need to be provided in a certain configuration. This subset may or may not span the border of several domains and is here referred to as a virtual grouping.
- Administrative domain (solid rectangle): is typically represented by an organization such as a research institute and provides a collection of resources.

1.2 Federation Scenarios

Using the base model entities introduced in the previous section, several federation scenarios can be modeled as shown in Figure 2 and Figure 3.

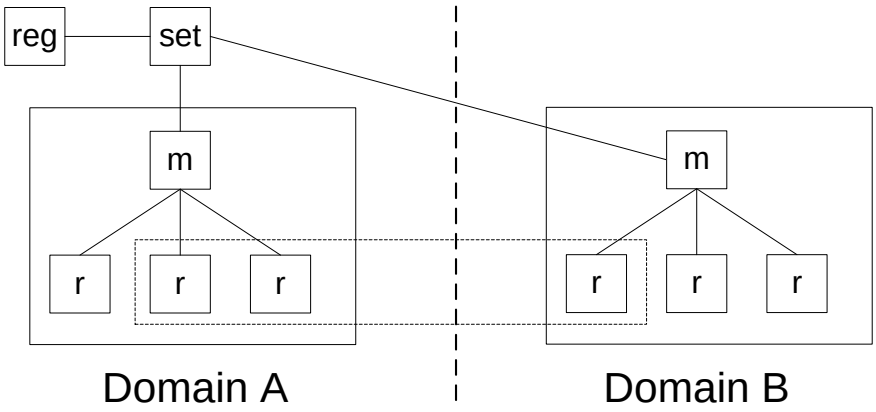

Figure 2: Centralized scenario [1]

Figure 2 shows a centralized scenario where resources provided by Domain B can fully be controlled by Domain A. This requires sufficient access rights for tools and users coming via Domain A to use resources in Domain B. It also requires a resource description for Domain B resources that can be understood and used within Domain A. Last but not least the control frameworks between Domain A and Domain B need

to be sufficiently aligned so that control and management operations known in Domain A can also be used within Domain B. Essentially, this scenario reflects common operational procedures across all the involved domains with little system-related heterogeneity. This can be achieved by either standardizing interfaces and operational procedures across all the domains involved or by a common system implementation (control framework + identity/policy handling + resource representations/descriptions).

This scenario reflects the current implementation of Teagle [2] and the underlying resource federation framework as used in Panlab [3].

The centralized approach works well for a dedicated system where the federation partners involved are well known and tightly interconnected. In the case of Panlab, a legal framework governs the rights and obligations by all stakeholders, the resources offered by the federation are described by a common model, and the control framework interfaces are standardized across all participating domains (many domains even run the same domain manager component code with proprietary resource adaptors, see the next chapter).

While this works well in a tightly coupled environment, centralized approaches are not acceptable for many resource providers and face several issues such as limited scalability and administrative overhead for centralized functions.

Figure 3 shows a federation scenario that is more open and should be investigated further with respect to federation in FIRE [4]. In this scenario both Domain A and B allow mutual accesses to control framework entities (domain managers), registries, and resources.

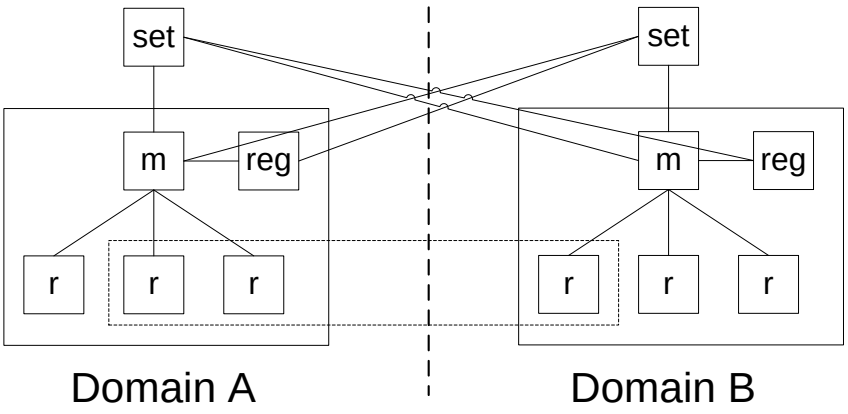

Figure 3: Full federation scenario [1]

Several issues need to be tackled in order to make this possible:

- Federated Identity Management (IDM): Users and resource representations need to be known and understood on both sides. Authentication and authorization need to be handled.

- Policy management and enforcement: The domains involved are likely to have their own policies regarding resource access and usage. These need to be respected and negotiated when forming virtual resource groupings.
- Resource description: in order to allow for meaningful virtual resource groupings that span several domains, the resources need to be known (mutual access to registries) and their usage and functioning must be understood (mutual understanding of resource descriptions).
- Control framework interfaces: in order to allow the tools and user interfaces (set) of both domains to provide meaningful functions to experimenters regarding the configuration and usage of virtual resource groupings, control framework functions (e.g. domain manager interface) must be understood on both sides.

2 The Teagle Framework

Much of this section is cited from [5]:

The idea and name of Teagle (being a "Testbed Google", a search engine for testbeds) was born during the course of FP6 SSA Panlab. Since then, the focus and feature set of Teagle has been extended considerably. Today, Teagle is a federation control unit that describes and controls arbitrary and heterogeneous resources across administrative domains. As such, Teagle has evolved from a standalone application into a rich framework of components that rely on each other to deliver the service of a customer facing federation frontend. Today Teagle consists of:

- A web portal
- A model-based repository
- A graphical testbed design and configuration environment
- A request processor
- A policy engine
- An orchestration engine
- A gateway (towards the underlying federation framework interfaces)

Below Teagle, the overall Panlab federation architecture relies on domain managers (Panlab Testbed Manager, PTM) with pluggable resources adaptors (RA) and interconnection gateways. The domain managers control resources inside the Panlab Partner domains while dedicated interconnection gateways (IGW) control the network connections between Panlab Partner domains.

Today, Teagle is attracting attention from various academic and industrial groups and organizations. However, users so far still consist mainly of Panlab Partner organizations. Access to external organizations and individuals has only been granted for demonstration purposes and only in exceptional cases. This is due to the fact that the Panlab Office has not yet been kicked-off officially at the time of writing. The office is needed to maintain the relationship with Panlab customers and to define the rights and duties on both ends. Without this formal establishment of legal and operational agreements, Panlab Partners are exposed to resource misuse.

The Teagle framework code will be released as Open Source apart from the policy engine. At the time of writing the code has not yet been released but it is expected that most components will be released under the Apache 2.0 license.

2.1 Panlab Framework

Much of this section is cited from [6]

The project Pan European Laboratory Infrastructure Implementation (PII) [3] addresses the need for large-scale testing facilities in the communications area by implementing an infrastructure for federating testbeds among innovation clusters. It builds upon the Panlab Specific Support Action which has received funding by the European Commission's Sixth Framework Programme.

Coordination of testing activities, for example infrastructure provisioning, ensuring the necessary interconnection of customers and testbeds, and the overall control and maintenance of the

environment, is ensured by the so-called Panlab Office and tools offered and maintained by this office. The main roles of the Panlab concept are (see also Figure 4):

- Panlab Partner – an entity that participates in Panlab activities by providing infrastructure elements and services necessary for the provisioning of testing services. Panlab Partners are connected to the Panlab Office to offer functionality to the Panlab Customers and provide the basis for the entire federation concept.
- Panlab Customer – an entity that uses services provided by the Panlab Office and the Panlab Partners, typically to carry out research and development activities and to implement and evaluate new technologies, products, or services, benefiting from the Panlab testbed federation offerings.
- Panlab Office – an entity that realizes a brokering service for the test facilities, coordinating and supporting the Panlab organization. It is responsible for the provisioning of the testing infrastructure and services by using tools and interfaces at the partner testbeds. Furthermore, the Panlab Office ensures and facilitates the communication between Panlab Partners and Panlab Customers.

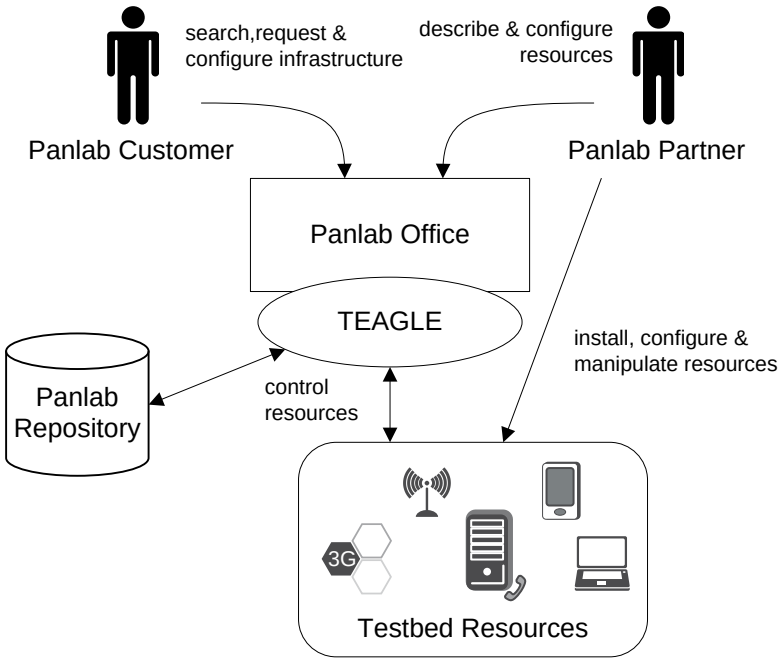

Figure 4: Panlab Federation stakeholders [6]

2.2 Teagle Architecture

As shown in Figure 5 Teagle provides the user-facing tools that allow working with distributed resources provided by different domains. A resource can be anything that can be controlled with software. In order to allow such generic resource control, Teagle relies on abstraction mechanisms (PTM + RA) and several Teagle-internal components such as an orchestration engine.

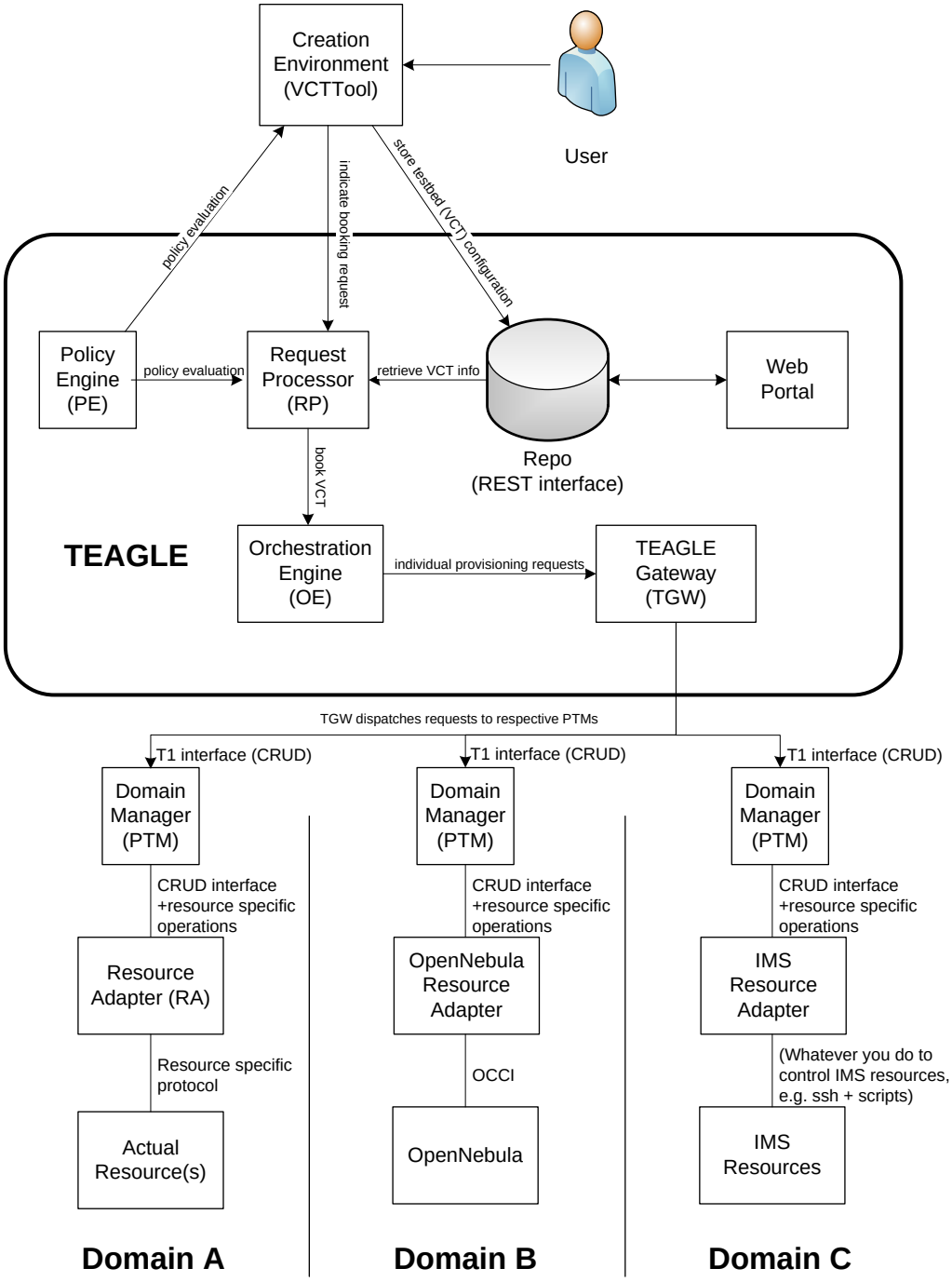

Figure 5: High level view of Teagle and the underlying control framework

An important distinction has to be made between resource types and resource instances. Each resource instance is an instance of a certain resource type. To some extent, the idea follows that of Object Oriented Programming. Resource types can be thought of as classes, which describe common attributes and behavior, while resource instances correspond to objects which can have individual values for the attributes the type specifies. This does not mean that every resource type can be instantiated at will. For some types only one instance or a fixed number of instances might preexist and no further instances can be deployed. This might for example apply to physical machines.

Resources are described by their resource type. Each resource type can have a number of attributes, which are named, typed and can further specify a default value. Possible resource attribute types are:

- Strings
- Integers
- Floating Point values
- Booleans
- references to other resource instances

Furthermore, arrays and dictionaries of the mentioned types are possible. Attributes can be both read-writable, read only, or even write only (e.g. passwords) for users. One attribute that all resource types have in common is the "id" attribute, which identifies a resource instance uniquely across a single domain (for multi-domain scenarios, the domain id is pre-fixed to the id, so the id becomes unique across domains). The id attribute is always read-only.

Relations between resources can be twofold:

- Containment - this denotes a resource that "lives" inside another resource. This might for example apply for a piece of software that is installed on a computer. Every resource instance can be contained in at most one parent instance. The containment relation for a resource might be omitted, leaving it to domain managers to choose an appropriate parent for a newly created instance.

- References - this denotes one application (resource instance) that "uses" another one to function. The referenced resource instance will be assigned to a configuration parameter of the referring instance. The latter can utilize this reference to query configuration parameters from the referenced instance.

As an example, think of two resource types: a virtual machine resource and some shared storage resource, the latter of which is available through NFS. A virtual machine (VM) resource uses (references) a shared storage resource as its data storage.

The virtual machine resource type might specify the following attributes (for a full specification please refer to section 2.11):

```
<resourceSpec>
   <name>vm</name>
   <attribute name="cpus" type="integer" default="2" />
   <attribute name="memory" type="integer" default="512" />
   <attribute name="storage" type="reference" />
</resourceSpec>
```

The shared storage type might be described like this:

```
<resourceSpec>
   <name>shared_storage</name>
   <attribute name="size" type="integer" default="10000" />
   <attribute name="uri" type="string" />
</resourceSpec>
```

When instantiated, respective instances might look like this:

```
<shared_storage>
   <id type="string">shared_storage-abc</id>
   <size type="integer">20000</size>
   <uri type="string">nfs://1.2.3.4/abc</uri>
</shared_storage>

<vm>
   <id type="string">vm-123</id>
   <cpus type="integer">4</cpus>
   <memory type="integer">2048</memory>
   <storage type="reference">shared_storage-abc</storage>
</vm>
```

The interesting part here is the storage parameter of the vm instance. It specifies, which shared_storage instance is to be used as data storage for the vm. The resource adapter responsible for setting up/configuring the virtual machine instance can use this reference to query information it needs for configuring the virtual machine from the shared storage resource (in this case probably the NFS URI).

Note, that this syntax currently does not reflect the access rules for attributes. For example, the "uri" attribute of the shared storage resource will not be chosen by users, it is thus a read-only attribute.

2.3 Example Workflow

In order to instantiate the setup described in the example in the previous section, the following steps would be performed:

- Before any interaction with resources can happen, information about them must exist within the Teagle repository. Additionally, their availability in individual domains must be indicated. This would happen by an administrator of the respective domain through the Teagle Web Portal.
- Assuming information about the resources exists within the repository, a user can now use a creation tool to create a Virtual Customer Testbed (VCT). At present, Teagle provides the so called VCT Tool as a general purpose means for this, however alternative implementations are not only possible but considered useful if not necessary for specialized purposes. A screenshot of the VCT similar to the one illustrated above is shown in Figure 6.

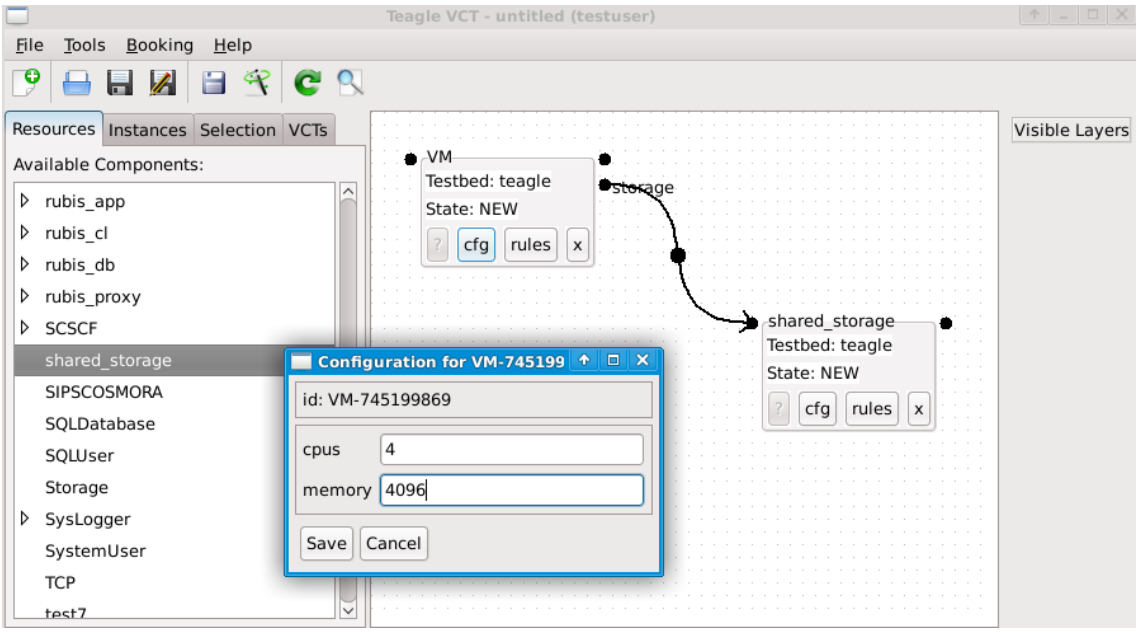

Figure 6: VCT Tool interface

- During creation of the testbed, the tool interacts with the Policy Engine, to indicate impossible or prohibited testbed layouts or configurations. This process is not a security mechanism, since the tool runs entirely under the control of its users. The evaluation of policies could thus be manipulated.
- When the testbed description is finished, the tool will store it in the Teagle repository. The testbed can now be booked.
- Upon booking, the VCT tool indicates this to the Request Processor. The Request Processor retrieves the VCT from the repository and sends it to the Orchestration Engine. Before this happens, policies are evaluated once more. This time the evaluation process happens inside the Teagle platform and is thus trusted.
- The Orchestration Engine (OE) creates individual provisioning (CRUD: create, read, update, delete) requests from the testbed description and compiles them into a script which is executed to initiate the actual provisioning of resource. Furthermore, the OE facilitates asynchronous operations.
- Each individual request is sent to the Teagle Gateway.
- The Teagle Gateway dispatches the provisioning requests to the domain managers of the respective domains.
- The domain managers (PTMs) are now responsible for conducting the provisioning of resources according to the incoming requests. From the perspective of the Teagle platform, this operation is opaque and could be performed in a number of ways. However, the implementation of a PTM available at present utilizes so called resource adapters as an abstraction layer for actual resources.

- The domain managers send the results of the operations performed back, which are aggregated at the orchestration engine, stored in the repository and finally presented to the user.

2.4 Teagle Portal

The Teagle Portal [2] is a website that is the primary contact point for customers and partners to use Teagle tools and interact with Panlab resources.

A testbed operator that wants to join the Teagle infrastructure by making his resources available to Teagle customers will register his organization and the resources it provides using the functions that the portal provides. The figure below shows the page that enables administrators to describe resources.

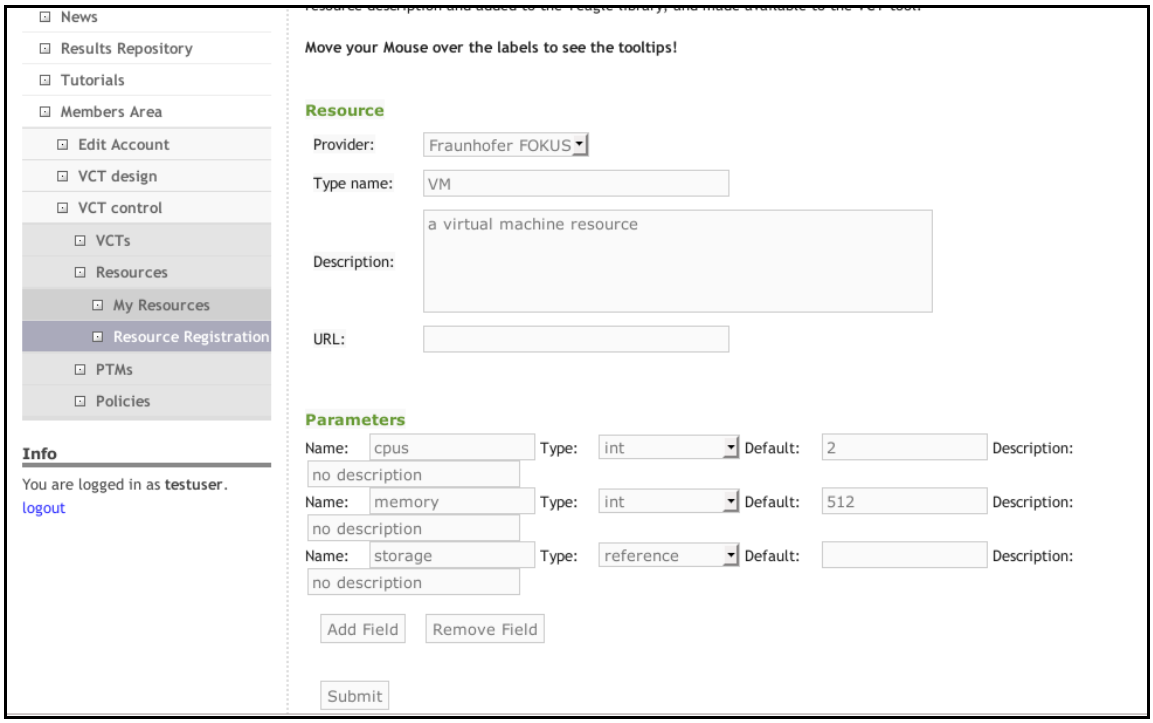

Figure 7: Teagle Portal screenshot

Via the Portal experimenters can also access the VCT Tool for which some tutorial videos are available at http://www.fire-teagle.org/tutorials.jsp

2.5 Teagle Repository

Parts of this section have been cited from [5].

The repository is a crucial component in the overall Teagle architecture. It stores information about available resource types, the configuration of existing resource instances, data about individual users and their role, and finally VCTs. Accordingly, the other core Teagle components carry out their tasks using the repository as data storage.

Interface

The repository application exposes an HTTP-based RESTful interface. 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.

The CRUD scheme is mapped to HTTP methods as follows

- Create - POST
- Read - GET
- Update - PUT
- Delete - DELETE

Software Architecture

The software architecture of the repository, as seen in Figure 8, 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 an 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.

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.

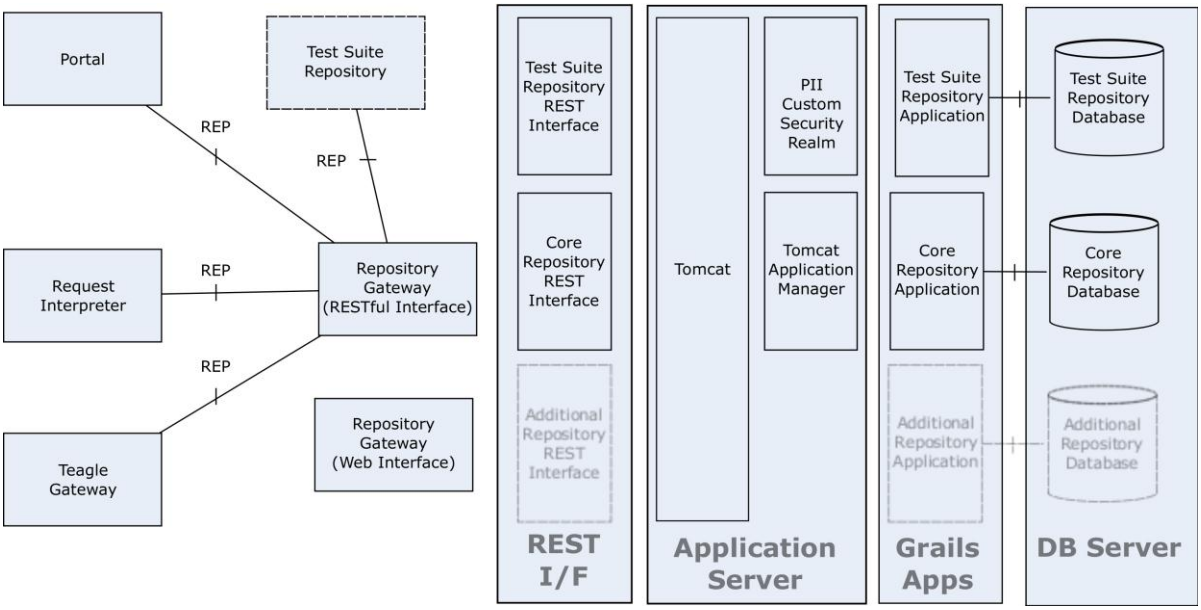

Figure 8: Repository Architecture [5]

This underlying architecture can be used to support any type of data model specification.

Repository Data Model

Panlab has chosen the DEN-ng information model [7] as the reference for the Panlab Data Model because of three key properties. First, it is the only model that was built from the outset to support patterns, specifically the role pattern. This provides inherent extensibility. Second, it uses a number of powerful abstractions 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 (DEN-ng). To maintain the separation of concerns, the Repository Data Model is divided by functional domains:

- The Resource domain is the core domain of the Repository Data Model, which covers the specification of the resource characteristics and their associations; these entities are defined in the Core Data Model.
- The TestSuite domain covers the specification of the TestSuite definition and Test Results. These entities are defined in the TestSuite Data Model.
- The Policies domain covers the definition of policies. These entities are defined in the Policies Data Model.

Resource Description

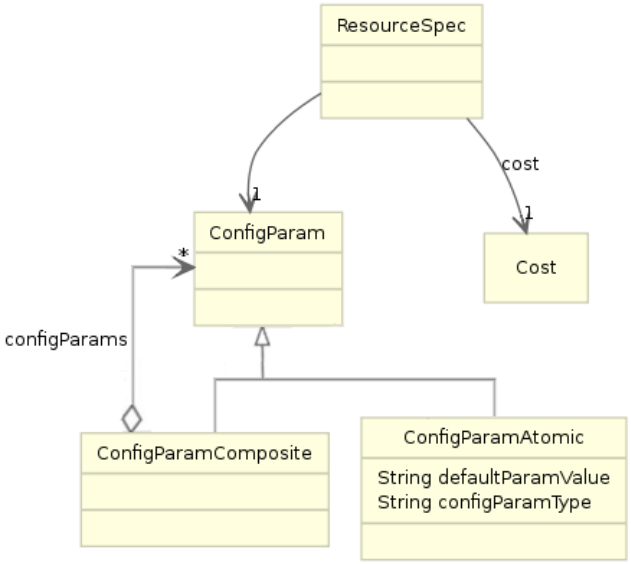

Figure 9: Resource Description Entities [5]

The *ResourceSpec* domain class defines a resource type. It does this via the *configurationParameters* association. Essentially, this association defines the parameters needed to configure the resource. It can either be a single configuration parameter (*ConfigParamAtomic*) or a composition of configuration parameters (*ConfigParamComposite*). This is based on the composite pattern used throughout the DEN-ng information model.

A Resource can also have an associated cost. The *Cost* class allows a monetary value to be placed on using the resource.

Resource Management

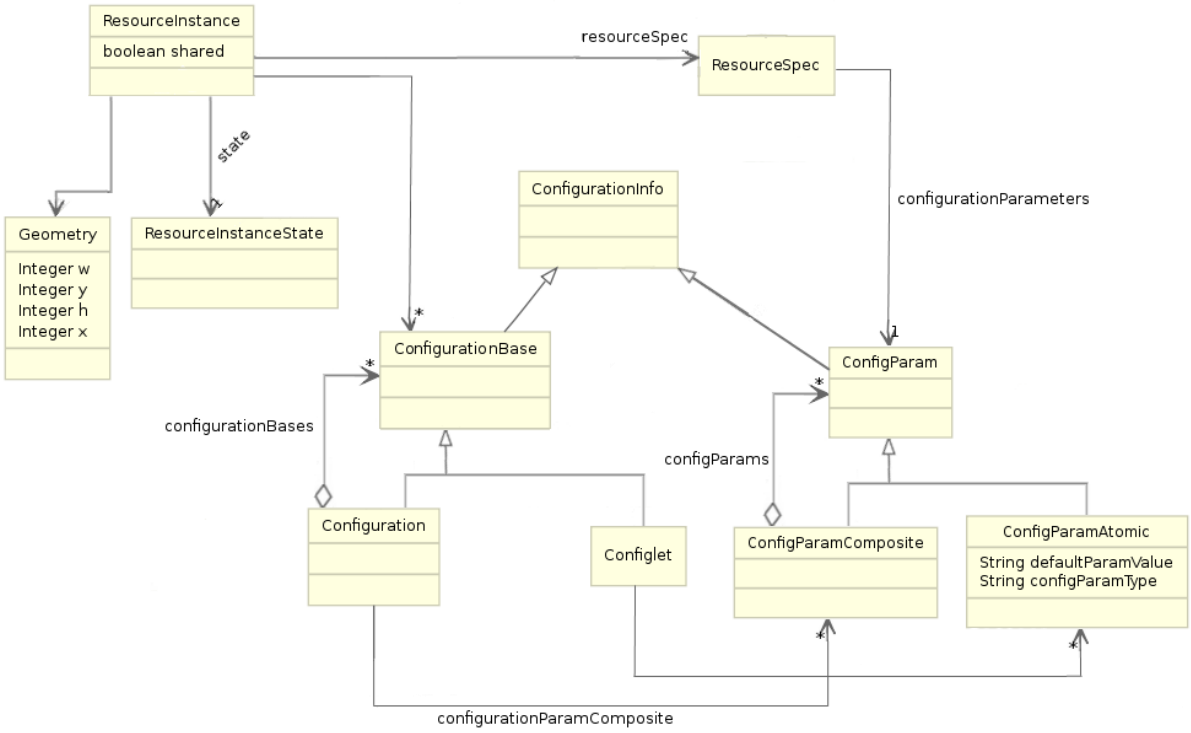

Figure 10: Resource Management Entities [5]

The *ResourceInstance* domain class represents the instantiation of a *ResourceSpec*. It contains the actual configuration data values. These values are associated through the link with *ConfigurationBase*. The *ConfigurationBase* domain class again implements the composite pattern, thus a *Configuration* can be composed of one or more *Configurations* and/or *Configlets*, whereas a *Configlet* is a single piece of configuration and is usually not stand-alone. As both *Configuration* and *Configlet* contain the actual configuration data values they must be associated with the configuration parameters whose values they contain. *Configuration* is associated with *ConfigParamComposite* and *Configlet* with *ConfigParamAtomic* respectively.

Geometry is required by the VCT Tool to store the coordinates of the resource representations for the VCT Tool workbench. However, since the issue of storing geometries is specific to the implementation of a certain creation environment (the VCT Tool) it is planned to move this information out of the central Teagle repository in the near future.

Virtual Customer Testbed

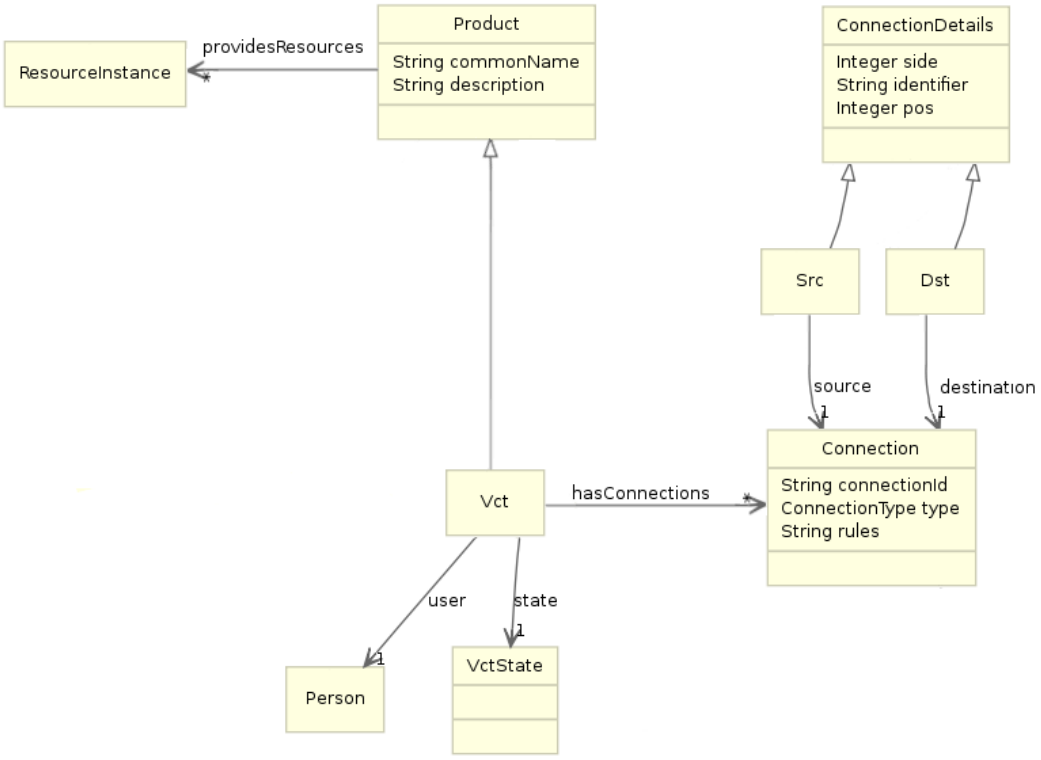

Figure 11: VCT Entities [5]

A VCT is a representation of the testing setup the customer requires. It is composed of the testing resource instances and the interconnections between these.

The *Connections* class is key in the composition of a *VCT*, it contains the details of the connections between the testing resources. The *hasConnections* association allows for the composition of one-to-many *Connections* within a *VCT*. The *Connection* class defines the type of *Connection* (References or Contains), any rules for the connection and the source (*Src*) and destination (*Dst*) resources (*ResourceInstances*) for the Connection.

Test Suite Data Model

Figure 12 represents an overview of the main entities of the TestSuite Data Model. A TestSuite is a type of *TestResultResource*. It is a collection composed by one or more *TestCases* that define test scenarios to check specific behaviours of a system/application/protocol. Each TestSuite references different VCTs (*ResourceReference*) which define the system configuration to be able to run the TestCases.

TestSuite defines a list of *Features*: Features to be tested and Features not to be tested.

A *TestCase* defines a set of conditions (*TestExecutionInfo*) under which a tester is able to determine if a system/application/service is working correctly according to a defined specification. A TestCase specifies the input values along with the anticipated outputs for the assessment of the *TestResult*.

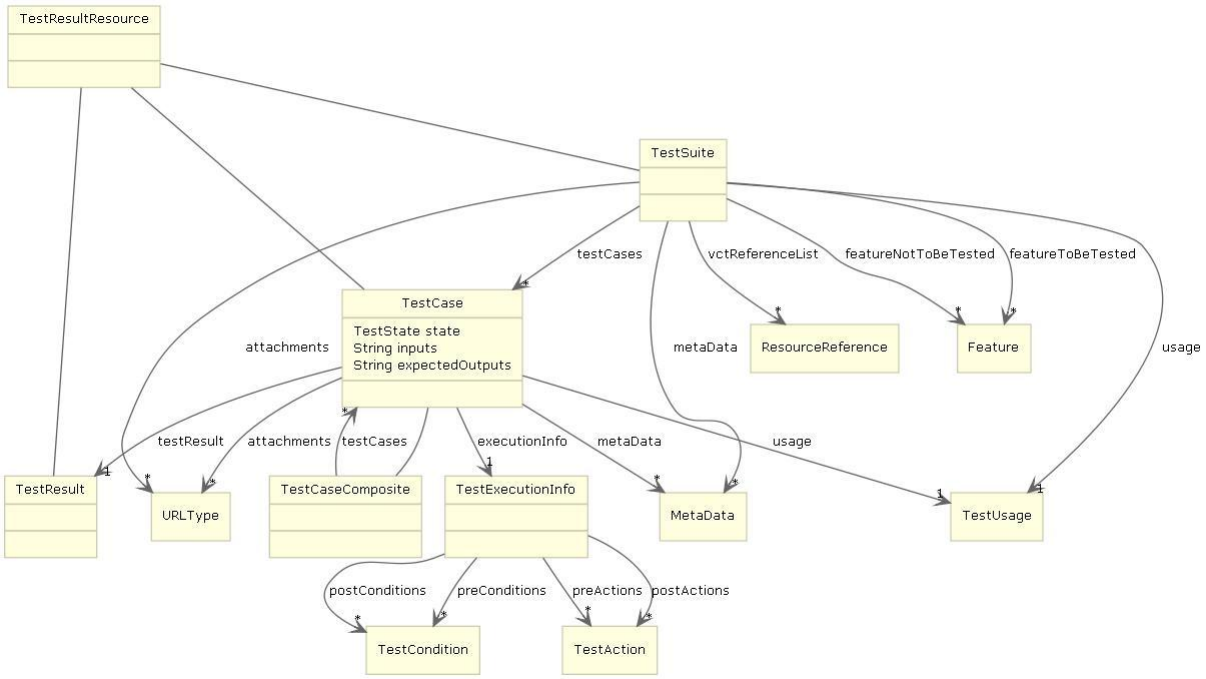

Figure 12: Test Suite Data Model [5]

Policies Data Model

In support of the implementation of the Access Control mechanism in Teagle, the Policies Data Model is provided to store the policies rules.

The Policies Data Model is based on the DEN-ng Policies specification. The following figure shows the main entities representing the type of policies supported. Currently, the model defines a set of *AccessControlPolicy*: *UserAccessPolicy*, *OrganisationAccessPolicy*, *ResourceAccessPolicy* and *RoleAccessPolicy*. Each policy references a specific type of entity (User, Organisation, Resource and Role) using *ResourceReference*. This association defines the policy subject.

The *AccessControlPolicy* is defined by a set of *ECAPolicyRule*. An *ECAPolicyRule* is composed of a *PolicyEvent*, *PolicyCondition* and a *PolicyAction*.

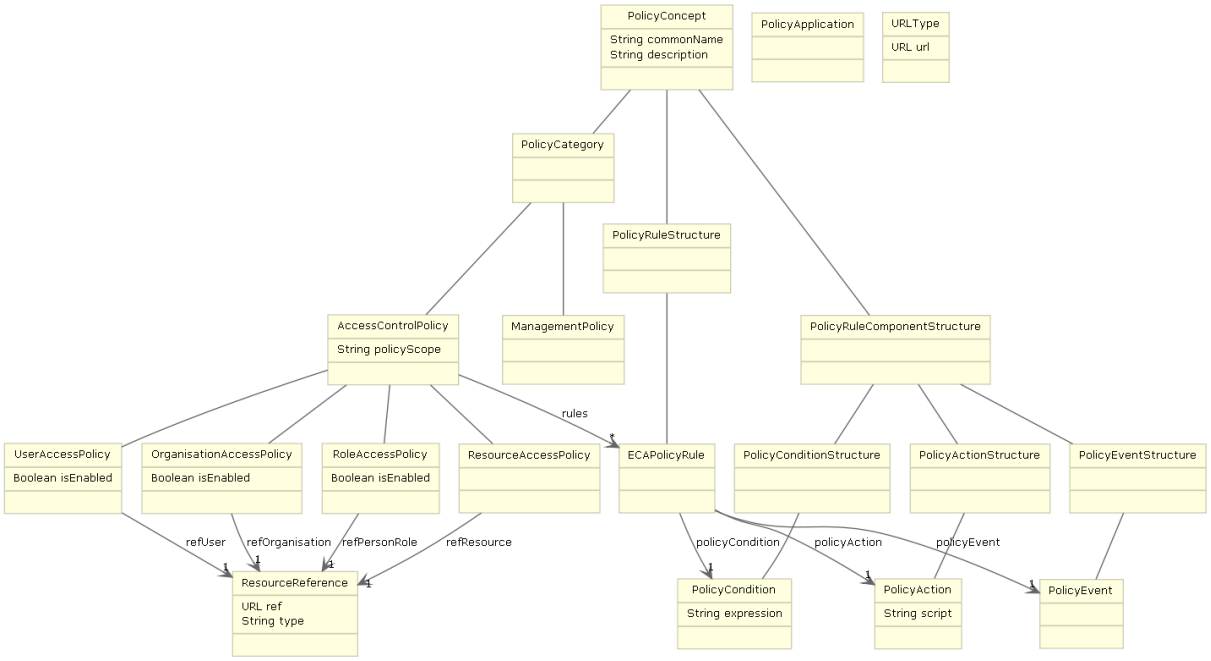

Figure 13: Policies Repository Data Model [5]

2.6 Teagle Policy Engine

Parts of this description have been cited from [5].

The Policy Engine ensures personalisation of the system according to a specific setup without any alterations on the core system. This functionality can be achieved by defining policies for specific situations which when evaluated may trigger specific actions to be executed by the core system. The core system needs to be able to issue a request that is understood by the Policy Engine and also to understand the response in order to act appropriately depending on each situation that occurred.

The concept of the policy engine is based on the proposed specification of the Open Mobile Alliance for the Policy Evaluation, Enforcement and Management (PEEM) enabler which was defined to control resource usage. This specification defines the basic architecture and interfaces between this component and interacting entities. Two modes of operation are defined: proxy mode and callable mode. In proxy mode, the Policy Engine transparently intercepts and proxies the requests according to the results of the policy evaluation. In this project we implemented the callable mode in which case the Policy Engine is specifically called for policy evaluation through the PEM-1 interface. The specification also proposes two policy formats from which one can choose: BPEL and Common Policy Format. For the management of the policies the specification proposes an interface based on XCAP referred to as PEM-2.

The policies are defined by the user through the Teagle portal where they define preferences regarding their organisations and resources. Only a partner or admin user can perform operations such as view/create/delete on the specific policies defined for the organisation or resources belonging to them. In addition to the policies editor, the Teagle Portal displays an overview of the policy evaluation and enforcement process taking place on the Policy Engine side.

Policies are selected based on specific criteria received as input by the policy engine. This input is provided by different trustable components in the network. The input request contains information based on which policies are selected, evaluated and then enforced. The request is based on a flexible template compatible with the OMA PEM-1 specification and provides information regarding the specific parameters and values. The Policy Engine exposes a web service interface for incoming PEM-1 messages thus the underlying protocol is SOAP.

The Policy Engine starts enforcement of the collected set of actions as soon as the evaluation of the rules has ended. An exclusive action is represented by the “deny” action which will drop the enforcement of all other actions. Another possible action is represented by “allow” which implies no specific behaviour be enforced. For example, a user may be authorized to book a specific resource while another user is not allowed to proceed with the same resource booking because they do not meet specific criteria (e.g. not part of a specific organisation).

The enforcement of the actions is performed either on the side of the issuer of the evaluation request or of the Policy Engine depending on the action type.

On the VCT Tool side the enforcement of the resulting actions may consist of:

- The restrictive display/booking of the resources on the VCT Tool according to predefined policies (as a consequence of the evaluation of the policies regarding the operation *bookResource* performed in the VCT Tool).
- A restrictive combination of resources in a connection in the case where such a connection does not make sense (e.g. a physical node resource contains another physical node resource). This results as a consequence of the evaluation of the policies regarding the operation connectResources.

The policies are stored in the policy repository and are defined according to the policy model used by the Teagle repository. According to the policy model, the following types of policies are possible:

- OrganisationAccessPolicies which contain policies defined for the organizations of the identities and the scopes “Target” or ”Originator”. They also includes policies defined for “all” identities and the scope of “all” identities. This type of policy is selected based on the organizations to which the identities (originatorID and targetID) belong.
- ResourceAccessPolicies which contain policies defined for the identities of type resource and the identity scopes “Target “or “Originator”. This type of policy is selected when one of the identities (originatorID or targetID) is of type resource.
- UserAccessPolicies which contain policies defined for the identities of type user and the identity scopes “Target “or “Originator”. This type of policy is selected when one of the identities (originatorID or targetID) is of type user.
- RoleAccessPolicies which contain policies defined for different user roles and the identity scopes ”Target” or “Originator”. This type of policy is selected based on the role of the user. Currently, the integration of this type of policy into the Policy Engine is work in progress.

Figure 14 shows the policy admin interface on the Teagle portal.

Figure 14: Policies overview from the admin perspective [5]

2.7 VCT Tool and Request Processor

The Teagle VCT Tool is a Web Start application where customers will normally spend most of their time when booking a VCT. It can be accessed by registered users from the Teagle Portal.

Due to the large volume of data stored in the Teagle Repository, the security and consistency issues involved with accessing this repository data, as well as the goal of allowing complete portability of VCT booking sessions, the VCT Tool communicates at startup and runtime with the repository. It does this when fetching user settings, available resources, saved testbeds from previous sessions, as well as when initiating the booking of VCTs and for monitoring the progress of a booking, or for querying the availability of a resource or of a set of resources.

Overview of the VCT Tool interface

The VCT Tool user interface is composed of a main grid/workbench, where designing and configuring resources in a VCT takes place. In addition, the tabs on the left panel allow browsing for resources, checking their availability, importing existing resources, etc. The figure below shows the interface of the tool with a simple VCT loaded.

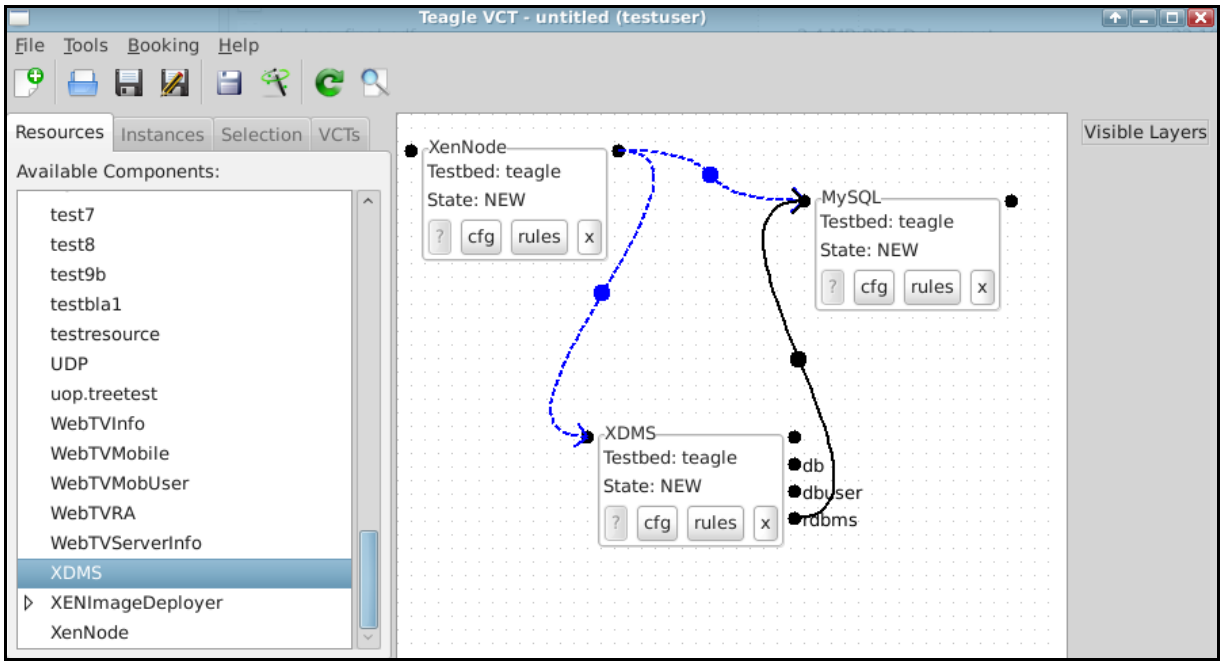

Figure 15: VCT Tool interface [5]

Creating a VCT

The main purpose of the VCT Tool is to enable customers to design experiment environments composed of one or more resources offered by individual domains. On startup, the tool presents an empty VCT, and booking the desired resources for the experiment requires adding resource instances to the main grid. This can either happen by selecting one of the resource types under the *Resources* tab – which will create a new instance of this type – or by selecting an instance that already exists from the *Instances* tab.

Some Resource types may be available in several domains. In this case the VCT Tool presents the option of explicitly choosing the domain in which a resource is to be instantiated as shown in the figure below, where the *rubis_cl* resource type is available in both the *uop* and *tssg-ptm* domains. Selecting either of these entries will cause the resource to be instantiated in the respective domain, while merely selecting the resource type will leave it to the request processor to choose between the available domains.

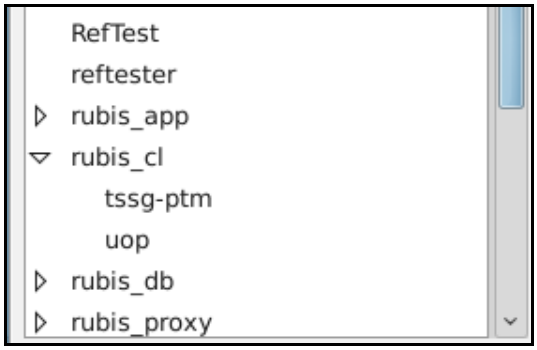

Figure 16: Choosing the target domain for a resource [5]

Resource instances can then be configured by clicking the *cfg* button. The configuration window for a *XenNode* resource instance is shown in the figure below.

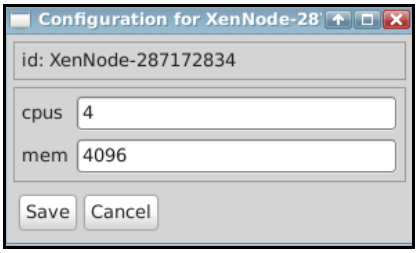

Figure 17: Configuration window [5]

Typically, more than one resource will be required for a meaningful VCT. However, VCTs restricted to only one Panlab Partner can consist of only one resource, given that the PTM managing the Partner domain implicitly handles the infrastructure required for provisioning that resource. In the VCT Tool, relations between resources can be specified using visual connections.

Booking

After a user has finished designing and configuring a VCT, he can request its actual instantiation by clicking the *book* button. Upon doing so, he will be presented with a summary page for review.

After confirmation, a booking request (along with an XML-based description of the VCT) will be sent to the Orchestration Engine which in turn is responsible for distributing individual provisioning requests for each resource instance in the VCT to the PTMs in the right order. However, since the VCT Tool runs entirely on the user's computer, and its behavior is thus under the user's full control, it is essentially not trustworthy, and may not be given access to the Orchestration Engine directly. The tool will therefore resort to storing the updated VCT in the repository and indicating the booking intent to the Teagle Request Processor, a backend component that is responsible for validating a VCT's configuration and transforming it to an XML format suitable as input for the Orchestration Engine.

After the booking has finished which might take several minutes depending on the resources involved, the user is presented with a booking result window, which indicates the success or failure of the operation and summarizes the booking. Furthermore, it presents the user with the opportunity to retrieve a more detailed log of the operation from the Orchestration Engine.

2.8 Teagle Orchestration Engine

The Orchestration Engine component is in charge of instantiating a VCT on top of one or more testbeds that are part of the federation. This is achieved by translating the original VCT layout model in the form of an executable orchestration script that includes the necessary sequence of service invocations to achieve the actual provisioning of the requested resources on each testbed. The service calls issued by the Orchestration Engine are redirected to the Teagle Gateway for intermediation purposes.

The Orchestration Engine prototype is built on top of the Spatel Engine framework which is based on SOA principles and finite state machines for service orchestration. All components inside Teagle (such as the Teagle Repository and the Teagle Gateway) and beyond the Teagle boundary (such as the list of

registered PTMs) are seen as web services exposing a list of operations. An orchestration generated to instantiate a VCT is itself seen as a service exposing the operation in charge of realizing the provisioning (the orchestration script).

Interface

The Orchestration Engine prototype offers a web-based interface to compile a registered VCT layout definition. The outcome is a new available service - whose logic is represented by the orchestration script - stored in the space of the Teagle client. The orchestration script can then be launched programmatically or using a specific web-based interface. The VCT Tool uses the programmatic interface (HTTP REST API) to launch the orchestration.

Orchestration Process

Figure 18 shows the typical steps of orchestration: the automatic generation of the orchestration script and then its execution. In case of immediate booking the execution is done straight after script compilation, otherwise the two steps are separated in time.

- Generation/Compilation: The OE carries out an analysis of the VCT input description (in the example depicted by Figure 18 we have a VLAN that includes a computing resource offered by an 'eict' testbed). It then generates a script sequencing a list of create or update service calls to instantiate or change the requested resources. Each service call is preceded by a configuration loading step (*loadConfiguration*) in which any design-time resource IDs found in the configuration of the resource are replaced by actual resource IDs. At the end of the service call, the table of the resources that have been created is updated so that ID substitution can be repeated in subsequent calls (*checkResponse*).
- Execution: Create and update service calls produce HTTP POST requests. For instance, the log in the figure shows the effect of a CREATE call for the VLAN: the endpoint is hosted by the Teagle Gateway, the *path url* represents the parent of the resource to be created and the XML data passed contains all the configuration information needed for the booking of the resource.

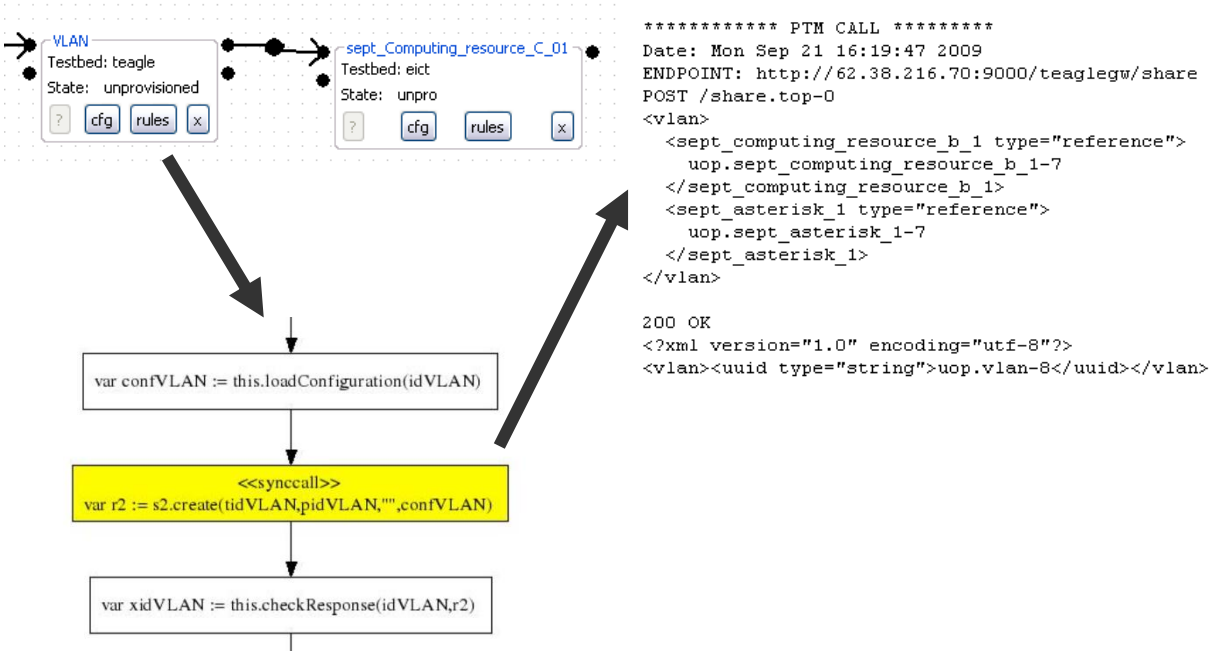

Figure 18: Example of generated orchestration script [5]

When the orchestration completes, the caller - the VCT Tool in our case - receives the list of all newly created resources in XML format. It uses this information to update the state of published resources as seen by the VCT designers.

Orchestration Strategies

Two orchestration strategies were implemented within the Orchestration Engine: the first one is fully sequential and is based on containment and references links found in the VCT definition. The compilation step computes a dependency graph of the required actions using the following assumptions: a contained resource depends on the container resource; and a referencing resource depends on the referenced resource. From this, the algorithm derives a valid ordering (which of course is not necessarily unique) and produces a script that satisfies the computed ordering.

The second strategy aims to optimize the booking of the resources. Resource booking can often be parallelized and then synchronized. This is particularly useful when dealing with intra-domain VCTs. In this case the compilation step generates an event loop: at each execution of the loop launching conditions are computed for each resource node (create or update). The launching conditions are initially computed from the dependency graph. The conditions depend on state variables that are updated each time the booking of a resource completes. The Figure below shows the structure of the state-machine script generated:

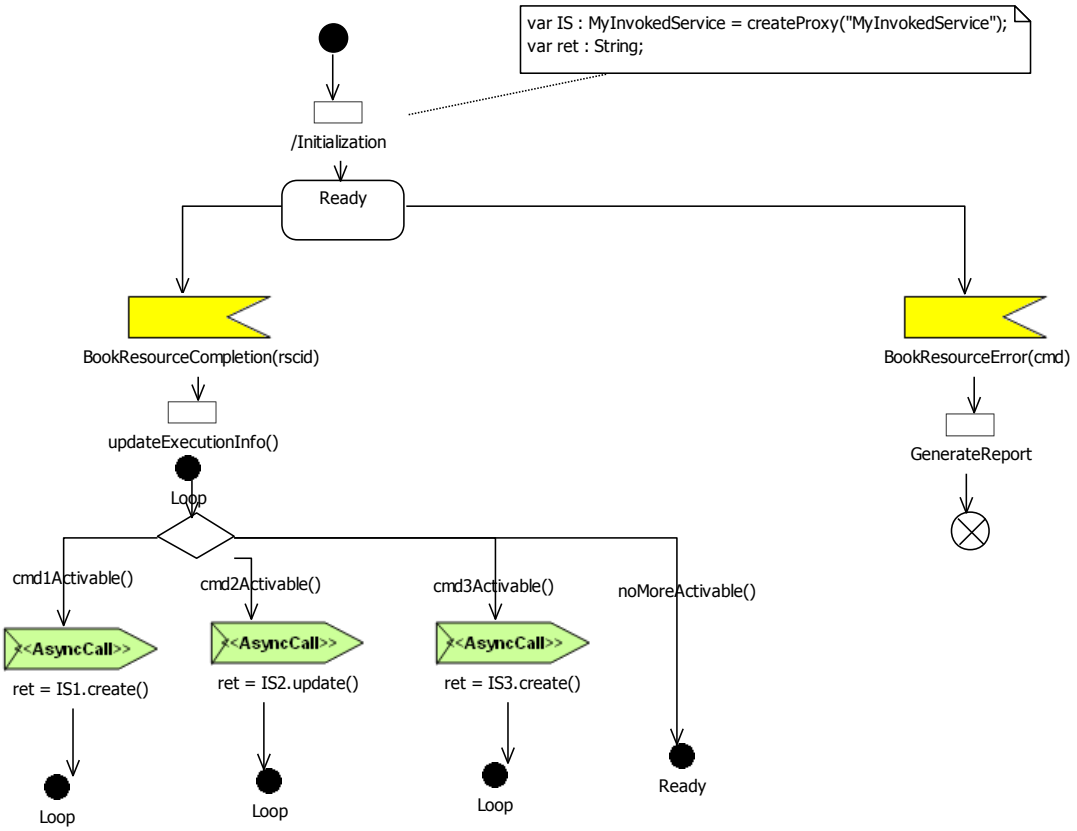

Figure 19: Asynchronous orchestration [5]

This mechanism assumes the reception by the Orchestration Engine of asynchronous notifications to signal the completion of booking commands.

Rollbacks

The actual prototype currently does not implement "compensation" actions when one of the booking commands fails. This is left for future development. Basically *delete or update* commands need to be executed in reverse order on all resources already created or updated.

2.9 Teagle Gateway

The Teagle Gateway instantiates and maintains service endpoints for all the registered PTMs. These service endpoints are utilised by the Orchestration Engine during VCT script execution. The service endpoints are accessed as RESTful web services in the context of CRUD operations (create, read, update, delete). The convention followed is to use the POST method for creating and updating resources, whereas the GET method implies a *query* request.

The Teagle Gateway routes the actions requested through SOAP-based web services to the appropriate PTMs according to the protocol defined for the T1 interface. The application logic of the gateway, as far as the processing and forwarding of the requests is concerned, is as follows:

- The PTM involved in an operation is determined by use of the first part of the path indicated in the invoked URL
- The second part of the path identifies the resource involved in that PTM
- The PTM identifier is used for selecting the correct web services endpoint for forwarding the request
- It provides an event reporting mechanism used by the PTMs as a SOAP-based web service. Incoming notifications are processed so that the appropriate entity (OE or Repository) is clarified as the recipient of the contained information.

Furthermore, an event reporting mechanism allows for the utilisation of asynchronous requests towards the PTMs in case of lengthy operations. In this case the outcome of a request is collected and forwarded to the OE when it becomes available to the PTM. An annotation indicating asynchronous requests is inserted by the OE.

2.10 Domain Managers (PTM)

As stated previously, Teagle interacts with the domain managers via the Teagle Gateway. The front-end interface to which the Teagle Gateway (TGW) dispatches requests is called T1.

Domain managers are responsible for processing provisioning requests on behalf of their respective domains. How they achieve this is basically left up to them as long as they adhere to the T1 interface specification.

T1 caters for connection and communication requests initiated by either side. Currently, the T1 interface is implemented as a SOAP webservice, which is secured through SSL/TLS. The Teagle GW invokes one of the following services that the PTM offers:

- Create – instantiate a resource type. The required parameters are the id of the parent resource instance (if any), the desired type and the instance's configuration.
- Update – reconfigures an existing instance. The required parameters are the id of the instance and the new configuration.
- Delete – deletes an existing instance. This is triggered when a resource instance is no longer in use by any VCT. Whether or not the instance is actually deleted is left to the corresponding RA.
- Query – Retrieve the configuration of an instance

The outcome of each of the operations is included in the response that is returned to the gateway.

In each of these requests the resource to be addressed is defined along with the data to be sent with the request. There is the possibility of inserting a callback indication defining the callback service endpoint. This results in a direct response from the PTM stating that the request has been received along with a request identifier to be used for clarification. Once the operation completes the PTM sends the outcome to the callback interface that was defined in the request indicating also the request identifier.

Resource Adapters

The PTM implementation currently available utilizes so-called Resouce Adapters (RA) as an interface to control resources. An RA can be thought of as a device driver, that abstracts resource-specific semantics to a generic CRUD interface. This interface resembles the T1 interface. It is currently implemented through an XML-RPC mechanism. However, for the actual RA implementation the technical implementation of its interface is completely transparent. An RA implementation never contacts any XML format or RPC mechanism but rather receives information about other entities as objects native to their programming environment. Currently, a framework for the implementation of RAs is available in Java™ and Python®.

2.11 T1 Interface

General specifications

This section describes some basic data types used during the specification of T1 interface methods in the Extended Backus–Naur Form (EBNF) as specified by ISO/IEC 14977. For a specification of the XML format used to describe resource configuration please see the next section.

```
None = ;
Digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
Integer = [ "-" ] Digit { Digit } ;
Float = [ Integer [ "." Digit { Digit } ] ] ;
Character = ? Any valid Unicode character ? ;
String = { Character } ;
Boolean = "true" | "false" ;
LowerCaseLetter = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" |
"k" | "l" | "m" | "n" | "o" | "p" | "q"  | "r" | "s" | "t" | "u" | "v" | "w" |
"x" | "y" | "z" ;
UpperCaseLetter = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" |
"K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" |
"X" | "Y" | "Z" ;
Letter = UpperCaseletter | LowerCaseLetter ;
Whitespace = ? Any ASCII whitespace character ? ;
PrintableNoDash = Letter | Whitespace | " | "!" | "§" | "$" | "%" | "&" | "/"
| "(" | ")" | "=" | "?" | "\" | "´" | "`" | "~" | "*" | "+" | "#" | "’" | "_"
| "," | "." | ":" | ";" | "<" | ">" | "|" | "^" | "°" | "{" | "}" | "[" | "]"
;
Printable = PrintableNoDash | "-" ;
LocalName = PrintableNoDash { PrintableNoDash } ;
Identifier = Printable { Printable } "-" LocalName ;
TypeName = Letter { Letter | Digit |  "_" } ;
ParameterName = Letter { Letter | Digit |  "_" } ;
ParameterValue = String | Boolean | Float | Integer | Identifier | None
ParameterType = ("string" | "integer" | "float" | "boolean" | "reference") [ -
("array" | "map")] ;
VCTName = Letter { Letter | Digit |  "_" } ;
InstanceName = Printable { Printable } ;
Identifier = Printable { Printable } ;
```

```
Configuration = ? as defined by XML schema at http://www.fire-teagle.org/T1 ?
;
```

Configuration data format

The configuration data accepted and emitted by the T1 interface methods is specified by the following XML schema which can also be found at the target namespace URL: http://www.fire-teagle.org/T1

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns=http://www.w3.org/2001/XMLSchema
       targetNamespace="http://www.fire-teagle.org/T1"
       xmlns:tns="http://www.fire-teagle.org/T1"
       elementFormDefault="qualified">

    <element name="configuration">
      <complexType>
            <choice minOccurs="0" maxOccurs="unbounded">
                  <element name="integer" type="tns:integer"
                        nillable="true"/>
                  <element name="string" type="tns:string"
                        nillable="true"/>
                  <element name="float" type="tns:float"
                        nillable="true"/>
                  <element name="boolean" type="tns:boolean"
                        nillable="true"/>
                  <element name="reference" type="tns:reference"
                        nillable="true"/>
                  <element name="array" type="tns:array"
                        nillable="true"/>
                  <element name="dict" type="tns:dict"
                        nillable="true"/>
            </choice>
      </complexType>
    </element>

    <complexType name="integer">
            <simpleContent>
                  <extension base="integer">
                        <attribute name="name"/>
                  </extension>
            </simpleContent>
    </complexType>

    <complexType name="string">
            <simpleContent>
                  <extension base="string">
                        <attribute name="name"/>
                  </extension>
            </simpleContent>
    </complexType>

    <simpleType name="referenceValueType">
      <restriction base="string">
            <minLength value="1" />
      </restriction>
    </simpleType>

    <complexType name="reference">
```

```
        <simpleContent>
            <extension base="tns:referenceValueType">
                <attribute name="name"/>
            </extension>
        </simpleContent>
    </complexType>

    <complexType name="float">
        <simpleContent>
            <extension base="float">
                <attribute name="name"/>
            </extension>
        </simpleContent>
    </complexType>

    <complexType name="boolean">
        <simpleContent>
            <extension base="boolean">
                <attribute name="name"/>
            </extension>
        </simpleContent>
    </complexType>

    <complexType name="arrayValueType">
        <choice minOccurs="0" maxOccurs="unbounded">
            <element name="integer" type="integer"
                nillable="true"/>
            <element name="string" type="string"
                nillable="true"/>
            <element name="float" type="float" nillable="true"/>
            <element name="boolean" type="boolean"
                nillable="true"/>
            <element name="reference"
                type="tns:referenceValueType"
                nillable="true"/>
        </choice>
    </complexType>

    <complexType name="array">
        <complexContent>
            <extension base="tns:arrayValueType">
                <attribute name="name"/>
            </extension>
        </complexContent>
    </complexType>

    <complexType name="dictValueType">
        <choice minOccurs="0" maxOccurs="unbounded">
            <element name="integer" type="tns:integer"
                nillable="true"/>
            <element name="string" type="tns:string"
                nillable="true"/>
            <element name="float" type="tns:float"
                nillable="true"/>
            <element name="boolean" type="tns:boolean"
                nillable="true"/>
            <element name="reference" type="tns:reference"
                nillable="true"/>
        </choice>
    </complexType>
```

```
    <complexType name="dict">
        <complexContent>
            <extension base="tns:dictValueType">
                        <attribute name="name"/>
                    </extension>
      </complexContent>
    </complexType>

</schema>
```

T1 Interface Methods

add_resource

```
add_resource(parent_id: Identifier, typename: TypeName, [name: LocalName,]
config: Configuration, vct: VCTName) : Identifier
```

The add_resource operation requests the instantiation of a given resource type with a given configuration as a child of the existing resource instance denoted by parent_id optionally specifying a local name. The vct parameter indicates which vct this instance will be part of. Upon success, an identifier of an existing resource instance is returned. This can be either an instance that was created in response to the request or an instance that had existed beforehand and might hence be used by others. Likewise, the domain manager can choose to return an identifier of an instance that is not a child of the instance given in parent_id. This is related to the general discussion on where resource control should principally reside (central vs. domain).

get_resource

```
get_resource(identifier: Identifier): Configuration
```

The get_resource operation retrieves configuration information for the existing resource instance denoted by identifier.

delete_resource

```
delete_resource(identifier: Identifier): None
```

The delete_resource operation requests the deletion of the existing resource instance given by the identifier. It is up to the domain manager to decide if the instance is actually deleted, so alternatively this can be viewed as an indication that a certain instance is not needed by Teagle anymore.

list_resources

```
list_resource(parent_id: Identifier, typename: TypeName): { Identifier }
```

The list_resources operation retrieves a list of all resource instances that are regarded as children of the instance denoted by parent_id and that are of the type given in typename. If parent_id is omitted, all

instances at the root of the resource hierarchy must be listed. If typename is omitted, instances if all types must be listed.

update_resources

```
update_resource (id: Identifier, config: Configuration):   Configuration
```

The update_resource operation requests the reconfiguration of an existing resource instance denoted by id with the configuration specified in config. This configuration does not have to include all parameters of the resource instance. It is sufficient to include only the parameters that are to be changed. Upon success, the full configuration of the resource instance is returned.

3 Mapping of TEFIS Requirements to Teagle

The source of requirements used for this section is the TEFIS Description of Work and deliverable D2.1.1 which defines the overall TEFIS architecture and use cases.

3.1 Overview

Requirement / Architectural Component as introduced byD2.1.1	**Teagle Functionality**	**Evaluation / Maturity**
Directory	Teagle Repository	✓ It remains to be decided what TEFIS related data will be stored in a Teagle based repository beyond data regarding resource management. Close cooperation with WP6 is expected.
Resource Configuration Support	VCT Tool	✓
Resource Provisioning	VCT Tool / Request Processor / Orchestration Engine / Panlab Testbed Manager (PTM)	✓
Resource Reservation	VCT Tool / Request Processor	😐 Limited
Experiment Design	Does not exist as a Teagle core concept, but the Federation Computing Interface (FCI) [8] provides some support	✘
Resource Management (TEFIS connectors)	PTM + RA	✓
Workflow / Scheduling / Experiment Execution	Does not exist as a Teagle core concept	✘

Cross domain connectivity (not explicitly defined by D2.1.1, but to allow experiments across sites, this will be necessary)	PTM + RA + IGW	😐 Early stages of development
Identity Management	Teagle Portal / Teagle Repository	😐 Centralized, no dedicated federation support (like Shibboleth)
Resource Abstraction (TEFIS connectors)	PTM + RA	✓
Experiment Monitoring	Monitoring as a Resource	😐 Resource dependent, monitoring tools can be booked as a resource offered by certain resource providers. For TEFIS a coherent experiment monitoring solution would be desirable.

3.2 Feedback from the TEFIS Testbed Providers

The general feedback received from the TEFIS testbed providers was that the resource model on which the Teagle Repository implementation is based is generic enough to describe TEFIS testbed resources.

3.2.1 ETICS

A SoftwareModule resource specification in the ETICS testbed could be:

```
<resourceSpec>
      <name>softwareModule</name>
      <attribute name="moduleName" type="string"/>
      <attribute name="configName" type="string"/>
      <attribute name="version" type="string"/>
      <attribute name="svnURL" type="url"/>
      <attribute name="checkoutCommands" type="string"/>
      <attribute name="buildCommands" type="string"/>
          ...
      <attribute name="parentModule" type="reference" />
      <attribute name="dependencies" type="List[string]"/>
      <attribute name="properites" type="List[string]"/>
</resourceSpec>
```

The meaning of operations implemented by a SoftwareModule Resource Adaptor could be:

- Create: create a module in ETICS using the parameters specified in the resource's configuration
- Update: update module's parameters
- Delete: delete the module from ETICS
- Query : retrieve the configuration for a given module

Similar specification and operations could be used for SoftwareTest resources.

3.2.2 Botnia

Concerns have been expressed for the Botnia testbed where only a limited set of resources (e.g. handbooks and checklists) could be controlled via Teagle, while in other cases this would not be possible due to legal reasons (e.g. specific software due to limited licences).

3.2.3 PlanetLab

Regarding a PlanetLab integration the TEFIS partners felt that the Teagle federation model and control framework would be adequate for using Planetlab as a resource provider: the Planetlab federation model (based on SFA: Slice Federation Architecture) also has notions of resources, registries, and domain managers. What is more, SFA is controllable via a single interface: the Slice Federation Interface (SFI).

Furthermore it was felt that for TEFIS, the centralized federation approach (as depicted by Figure 2) would be feasible to implement. In this case:

- The Teagle resources would be Planetlab virtual machines running on Planetlab servers.
- The Teagle Planetlab Domain Manager would control the Planetlab resources via a Planetlab Resource Adapter, which interacts with the Slice Federation Interface.

A typical workflow that could be run using the VCT tool could be:

- The experimenter creates a new TEFIS experiment including Planetlab support.
- The experimenter configures its Planetlab credentials in the VCT tool (each Planetlab user can generate an RSA key pair for authentication with the Planetlab infrastructure).
- The experimenter creates a new Planetlab slice or adds an existing one to their experiment via the VCT tool. Note that for legal reasons, the slice creation is not fully automated and must be validated by the Principal Investigator (the local person in charge of validating the requested Planetlab experiments). Let's assume the authorization has been granted, the experimenter may now control their slice.
- The experimenter uses the VCT tool to list the virtual machines in his slice. He can also add or delete virtual machines.

- The experimenter has now remote access to any virtual machines in his slice and he deploys a set of services on a subset of machines.
- Thanks to the orchestration capabilities of the VCT tool, deployed services are automatically configured and chained with each other as well as with services in any other domains.

Although the approach outlined above can be realized on a short term basis (and in fact has already been realized as outline in [9]), the full federation model is much more appealing as it grants more control to the individual domains while still enabling mutual resource access. Furthermore it enables recursion where a domain can essentially be a federation in itself. However, the full federation scenario is more challenging from a technical (resource description, policy negotiation, etc.) as well as from a legal point of view (resource usage contracts across a federation of federations).

3.2.4 PACAGrid

In the case of the PACAGrid testbed the resource type could be mainly a computational resource. The idea would be to model a computational resource used for computationally intensive experiments.

In this sense a computational resource would be represented by a ProActive agent acting as the access point to the machine when a job requiring resources is scheduled for execution. There could be resource types for native resources as well as for virtualized ones. A possible spec of a native computational resource could be:

```
<resourceSpec>
      <name>ComputationalResource</name>
      <attribute name="hostArchitecture" type="string"/>
      <attribute name="memory" type="int"/>
      <attribute name="numberOfCores" type="int"/>
      <attribute name="operatingSystem" type="string"/>
      <attribute name="javaJVMOptions" type="string"/>
      <attribute name="javaClasspath" type="List[string]"/>
      <attribute name="installedSoftware" type="List[string]"/>
</resourceSpec>
```

In this case one could imagine choosing among different configurations already existing in the infrastructure. A mechanism to automatically install and configure software packages could be adopted or reused in order to dynamically setup a computational resource.

As described in the eTravel use case, an application requiring computational resources could use ETICS resources to perform software building and PACAGrid computational resources in order to execute performance tests. A VCT designed using the VCT Tool would span those two TEFIS testbeds.

4 TEFIS Portal Specification

4.1 Definition of the TEFIS Portal

The TEFIS portal will be the single access point to specify the experiments to undergo managed execution remotely on TEFIS testbeds. This portal will provide the service intelligence that will allow the customization of the tools and testing methodologies to the task in hand, giving access to the resources for carrying out the experiment.

The set of components constituting the TEFIS portal will be made available directly through a common user interface which will allow testbeds to be searched for, the retrieval of reference data sets, the configuration of tests to perform, gathering of diagnostic parameters or the specification of resource prerequisites. It could also provide its users access to a wide and complete range of testing tools. Those user tools will be a strong support for the definition of the experiments and their requirements, their submission, and follow the scheduling, execution, and completion of an experiment.

Taking into consideration the users' skills at ICT languages and terminologies, the wide variety of such skill-levels – from nearly unskilled users to experts – requires that the portal has to be a customisable interface, enabling each user to create their own environment for defining, executing and interacting with the experiments on the testbeds.

The proposed solution will rely on a fully customizable web-based environment. The core services, exposed by the portal, will allow the gap to be reduced between the testbed users' expectations in terms of experiment support and the functionalities and complexity of underlying testbeds.

The TEFIS portal will be the user interface to define, execute, retrieve the results, and analyse the experiments on the different testbeds. Thus, the user will be able to describe the requirements for their test (configuration, data set, elements of the different platforms, resources needed and configuration of the nodes), the data required (previously loaded as part of TEFIS environment, or coming directly from the user spaces or third-party sources).

As shown in Figure 20 the TEFIS portal resides as presentation layer on top of the TEFIS middleware. It communicates with TEFIS backend components and core services via the TEFIS API.

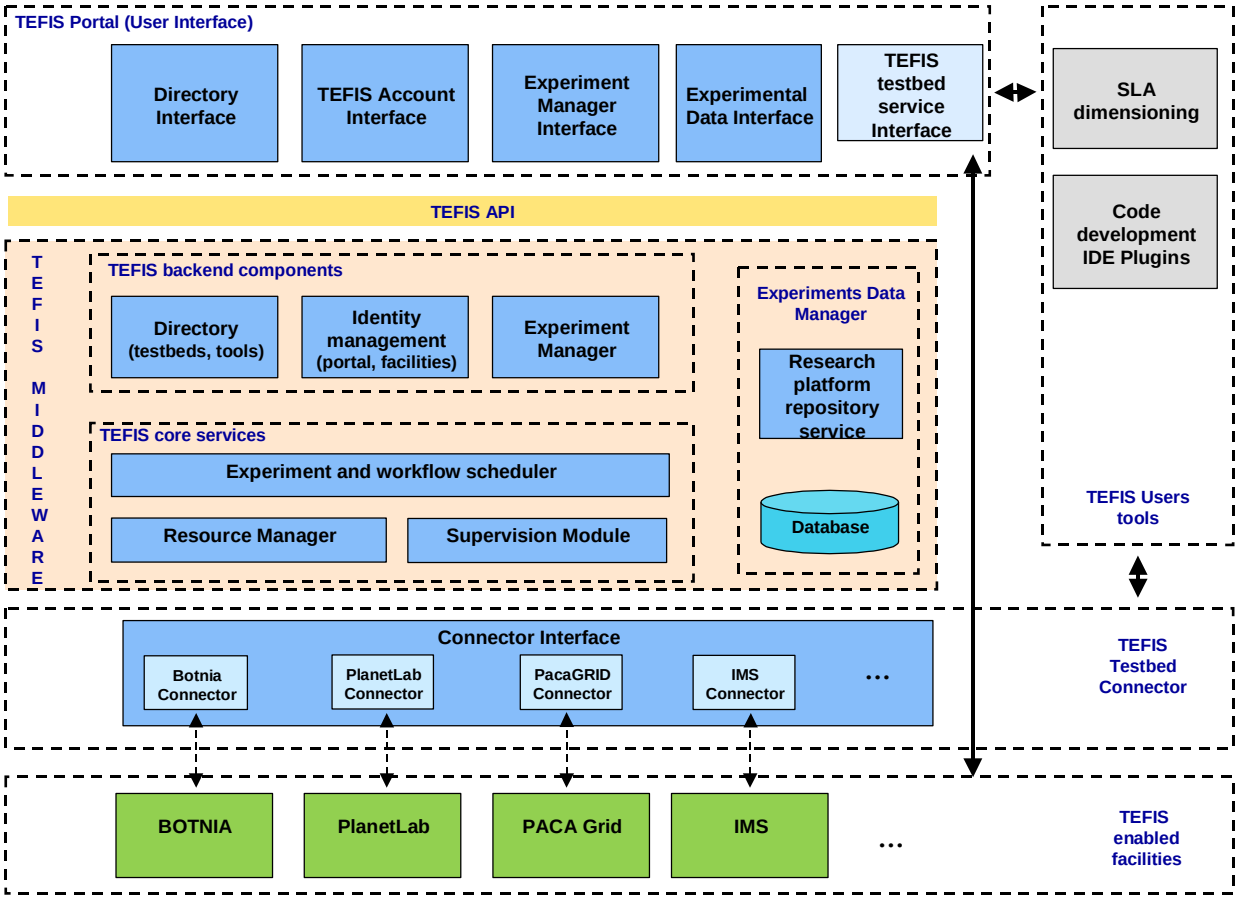

Figure 20: TEFIS overall architecture [10]

4.2 Portal User Roles & Groups

Different user groups will interact with the Portal. In the following the user roles are defined. Specific access privileges will be defined at a later stage and assigned to the user roles.

- TEFIS Consortium Partner: Is a full partner of the TEFIS consortium.
- TEFIS Testbed Owner: Is a resource provider to the TEFIS federation.
- TEFIS User: A user of TEFIS services.
- TEFIS Admin: Administrator role.
- TEFIS Editor: Web page content editor

In addition the Portal will allow for the collation of individual user accounts into groups (e.g. organizations).

Service level agreements as well as legal procedures regarding TEFIS usage terms and conditions are out of scope of this deliverable.

4.3 Portal Use Cases

4.3.1 Portal User Requesting TEFIS Credentials

Description:

Someone not known on the Portal accesses the Portal website and requests an account. Upon registration, access credentials are provided to the requester. The form of such credentials is to be decided elsewhere. Possible options are a username/password combination or a digital certificate.

Preconditions:

Public registration page including terms and conditions must be available on the Portal.

Workflow:

1. The user accesses a registration page on the Portal.
2. The user fills in a registration form. This defines the user profile.
3. The user acknowledges the terms and conditions.
4. The user sends the request.
5. The identity management system generates the credentials and the user is assigned a user group.
6. The credentials are passed to the user.

Postconditions:

The user has valid credentials to be presented at the Portal and TEFIS testbed level.

4.3.2 Portal User Logging Into the Portal

Description:

A user logs into the Portal. Based on its user role, the Portal functions are adapted.

Preconditions:

The user must have valid TEFIS credentials. The user must be known to the TEFIS identity management system.

Workflow:

1. The user accesses a login page on the Portal.
2. The user presents their credentials.
3. The credentials are validated.

4. The user is either allowed to access the Portal or not.
5. The Portal functions and content are adapted according to the privileges associated with the user account.

Postconditions:

The user is displayed Portal content based on their user role and associated privileges.

4.3.3 Portal User Communicating With the TEFIS Platform via the Portal

Description:

This is a generic use case. The user shall be enabled to make use of a variety of functions offered by the TEFIS API via the Portal. At the time of writing it is not possible to determine the exact set of features and functionalities that the user will be able to access because the final TEFIS API has not yet been defined. What types of features are exposed also depends on the user role. For example, a TEFIS testbed provider might be enabled to register new resource types, while a TEFIS User can access resource configuration tools. The Portal feature set will be highly dependent on the TEFIS overall architecture and available testbeds.

Preconditions:

The user is logged into the Portal. The web pages and/or tools to be used for specifying a request towards the TEFIS core services are available. The user has sufficient privileges to access said page/tool.

Workflow:

1. The user accesses a page or tool offering specific portal functionality such as registering a new resource, listing resources, viewing experiments, etc.
2. Such functionality might be integrated into the portal (e.g. a specific webpage) or might be offered by an application (e.g. Java Web Start Application).
3. The page or tool supports the user in defining a request that is understood by the TEFIS core services level (e.g. the TEFIS API, the Teagle API).
4. The request is passed to the TEFIS core services with attached user identity (e.g. the digital user certificate is attached to an HTTP message).
5. The request is processed by the TEFIS core services and an appropriate response is sent.
6. The response is processed by the Portal and displayed to the user. For example, if the request was to list the available testbed resources, the Portal would interpret the response and display the available resources to the User, rendering the information appropriately e.g. using HTML.

Postconditions:

This depends entirely on the type and content of the request made.

4.3.4 Portal User Requesting De-Registration

Description:

A user logs into the portal and requests a de-registration. This means that all data associated with this user should be deleted unless otherwise specified by the user (e.g. keep experiment results).

Preconditions:

The user must be logged into the Portal. The de-registration web page is available.

Workflow:

1. The user accesses a de-registration page.
2. The user is informed about the fact that de-registration will result in a complete deletion of associated data and has to acknowledge this.
3. The user is shown a choice of what data to keep and where to keep it, allow for export of experiment data, etc.).
4. Upon acknowledgement all user related data is deleted or moved to a specified location.
5. The user is logged off.

Postconditions:

All user related data has been deleted or moved to a specified location. The user is logged off.

4.3.5 Portal User Logging Off

Description:

A user logs off from the Portal.

Preconditions:

The user must be logged into the Portal. The log-off functionality must be available.

Workflow:

1. The user uses the log-off hyperlink (or button, whatever represents the log-off functionality).
2. The user is logged off and is shown the success of this operation.

Postconditions:

The user is logged off and all states associated with his session are cleared on the Portal side.

4.3.6 Portal User is Logged Off Automatically

Description:

A user is logged off automatically from the Portal due to a period of inactivity.

Preconditions:

The user must be logged into the Portal and has not performed any operation for, say, 20 minutes.

Workflow:

1. The system realizes that the user has not been active within the past 20 minutes.
2. The system initiates the log-off procedure.
3. The user is logged off from the system and all states are cleared apart from some entry regarding the automatic log-off event.

Postconditions:

The user is logged off and most states associated with his session are cleared on the Portal side.

4.3.7 Portal User Viewing and Editing his Profile

Description:

A user is editing his profile to update personal information.

Preconditions:

The user must be logged into the Portal. The profile page is available.

Workflow:

1. The user accesses his profile page.
2. The user can view personal data associated with his account.
3. The user can view the terms and conditions accepted during registration.
4. The user can edit certain fields of his profile (e.g. address details).

The user saves this information. Postconditions:

The updated user profile information is stored on the system.

4.3.8 Portal User Viewing the Public Part of the Portal

Description:

A user has accessed and is viewing public information on the Portal. This includes for example:

- Homepage: general information, teaser
- About: FAQs, public contact, etc.
- Imprint: usual imprint, incl. public contact
- News
- Help: FAQs, knowledge base (e.g. wiki, manuals)

Preconditions:

The public pages must be available.

Workflow:

1. The user accesses a public page.

Postconditions:

None

4.3.9 Portal User Viewing Previously Stored Experiment Data

Description:

A user is accessing and viewing previously stored experiment data that has been submitted either by himself of somebody else.

Preconditions:

The experiment results page and the requested data must be available. The user must have sufficient access rights to view the data.

Workflow:

1. The user accesses the experiment result page.
2. He can search for specific information, view the data and browse it. Which data can be accessed and processed specifically is specified in WP6.

Postconditions:

None

4.3.10 Portal User Storing Experiment Data

Description:

A user is publishing experiment data that has been collected during a TEFIS experiment.

Preconditions:

The experiment results must be available either offline or stored (where and how is out of scope of this deliverable) in some other TEFIS platform component.

Workflow:

1. The user accesses the experiment publishing page.
2. He uploads data.
3. He tags and enriches the data to be stored with meta-information
4. He publishes the experiment.

Alternative Workflow:

1. The user accesses the experiment publishing page.
2. Data has already been collected and made available on this page through other TEFIS components
3. He tags and enriches the data to be stored with meta-information
4. He publishes the experiment.

Postconditions:

The data is available publicly. The meta-data facilitates searching/browsing stored experiment results.

4.4 High Level Portal Architecture Specification

4.4.1 General Assumptions

1. Each user action has a response.
2. Each page has role-based page parts (to break down knowledge depth and access rights).
3. The Portal system requires a host with WSGI-enabled Web Server.
4. A domain name needs to be registered that is resolved to the Portal endpoint.

4.4.2 Layered Architecture

The portal will follow the widely accepted and used Model-View-Controller (MVC) Scheme. MVC is an architectural pattern used in software engineering that allows for the separation of concerns. Generally, it isolates the presentation of information from the application logic. This allows independent development and maintenance while ensuring easy integration.

The Portal shall implement this concept as shown in the figure below.

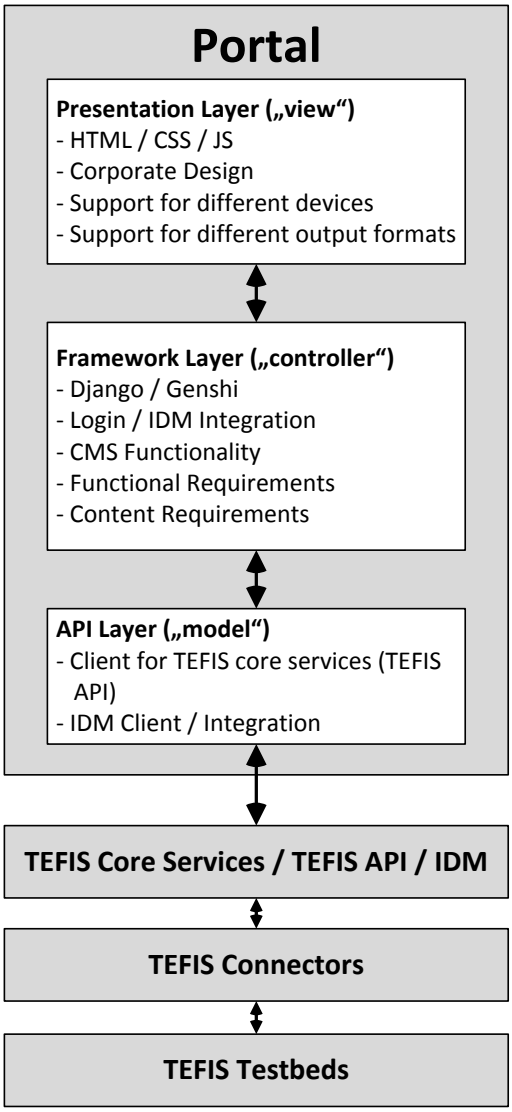

Figure 21: TEFIS Portal design

4.4.3 Presentation Layer Requirements

First Release

- HTML
- Support for desktop browser supporting current web standards
- Basic support for mobile devices
- Basic support for b-grade browsers
- Basic print optimizations

Later Releases

- HTML, RSS
- A-grade mobile browser optimizations (webkit iOS, webkit android, webkit web os)
- B-grade desktop browser (incl. internet explorer 7+) optimizations

4.4.4 Framework Layer Requirements

The framework layer functionality is defined by the use cases and will be subject to changes as TEFIS evolves. Currently identified items/pages are:

User

- registration: view, edit
- terms & conditions: view
- login: view
- profile: view, edit (incl. remove)

Public

- homepage: view, edit; may consist of other pages (teaser part)
- about: view, edit; incl. public FAQs (read only), incl. public contact
- imprint: view, edit, incl. public contact
- news: view, edit

Resources (limited access)

- list: view (sortable)
- service: view, edit (incl. remove), add (vodcast help link)

Experiments

- list: view (sortable)
- experiment: view, edit (incl. remove), add (vodcast help link)

Help

- FAQs (dynamic): view, edit (incl. remove), add (needs moderator).
- contact: view, edit; public and internal parts
- knowledge base: might be external wiki; incl. manuals, vodcasts, etc.

4.4.5 API Layer Requirements

This very much depends on the TEFIS core services and their specification. At the time of writing, the TEFIS API had not been specified. It remains to be seen what exact functionalities it will offer. The same is true for the identity management system (IDM). However, due to the layered model of the portal, the higher layers can be developed, tested and maintained while the API relevant parts are still changing or being extended.

4.5 TEFIS Portal Model and Information Flow

The TEFIS Portal will be based on three main components:

- Directory interface
- Account management interface
- Experiment management interface
- Experimental data interface
- TEFIS testbed service interface

4.5.1 Directory Service & Interface

The Directory will be the repository of tools, facilities, and any resources provided by the heterogeneous testbeds. Resource listed the directory can be used for experiment specification and execution.

In the TEFIS model, the Directory will provide the fundamental assets of the infrastructure to the other components. This component will handle the assignment of resources for each experiment, based on their respective needs and requirements, and will allow check-in-check-out management of software, hardware, laboratories, documentation or any other resource the Directory accounts for.

The portal interface will allow the communication with the directory as a TEFIS backend component. This includes adding new resources, updating existing ones, etc. The business logic associated to this module will be implemented in a TEFIS middleware service while the TEFIS Portal will be responsible for offering the assets to the user via the presentation layer.

This presentation layer will interact with the experimenter providing their single point of access to all the resources in an intuitive and an easy-to-use form. It will list the most appropriate facilities, tools, documentation, etc. for the user's profile and allow them to choose according to their needs at different times of the experiment lifecycle. The presentation layer will provide format to the information provided by the core services relating to the recommended resources and list them to the user.

4.5.2 Account Management Service & Interface

The account management system will be in charge of creating accounts for TEFIS users and providers. The appropriate management of the user profiles will be a critical part from the point of view of the maintenance of the security of the system, as ineffective user and privilege management often leads to

many systems being compromised, so it is essential to protect the whole platform with simple and effective user account management techniques.

The information regarding TEFIS users will always be available to the TEFIS Portal administrator taking into consideration that the situation and needs of the users could differ from time to time or they specifically request membership of a different and more appropriate user group, and it will be necessary to modify their privileges, access policies, profile, etc.

The process for managing TEFIS users and groups will be clearly and concisely defined in the Account Manager Component implemented as a middleware service so that it will be capable of performing the required operations in a controlled and secure environment and in an automatic way, saving time and personal involvement as well as assuring the appropriate levels of adequacy to the TEFIS project objectives.

The TEFIS Portal will implement the user interface for managing user profiles as a frontend for the TEFIS identity management system residing on the TEFIS middleware service layer.

The presentation layer to be included in the Portal will interact with the experimenters providing them a single point of access to their profile, so they could view, edit, and remove themselves as users quickly and in few steps. It will provide a registration form, a profile record that they could review and edit at any time, and login functionality.

The presentation layer will provide access then to all the specific information regarding a user.

4.5.3 Experiment Manager Service and Interface

The Experiment Manager will provide the TEFIS users with the ability to define, configure and execute the experiments, and also receive the result reports.

The experiment manager will manage the whole experiment process, combining the integration of

1. experiment specification, configuration and management
2. experiment environment creation and configuration
3. organisation and scheduling of the testing process
4. submission of the execution of test programmes
5. analysis of results and report generation

and the provision of traceability throughout the process between specification, test cases, requirements and executions.

Figure 22: Experiment lifecycle

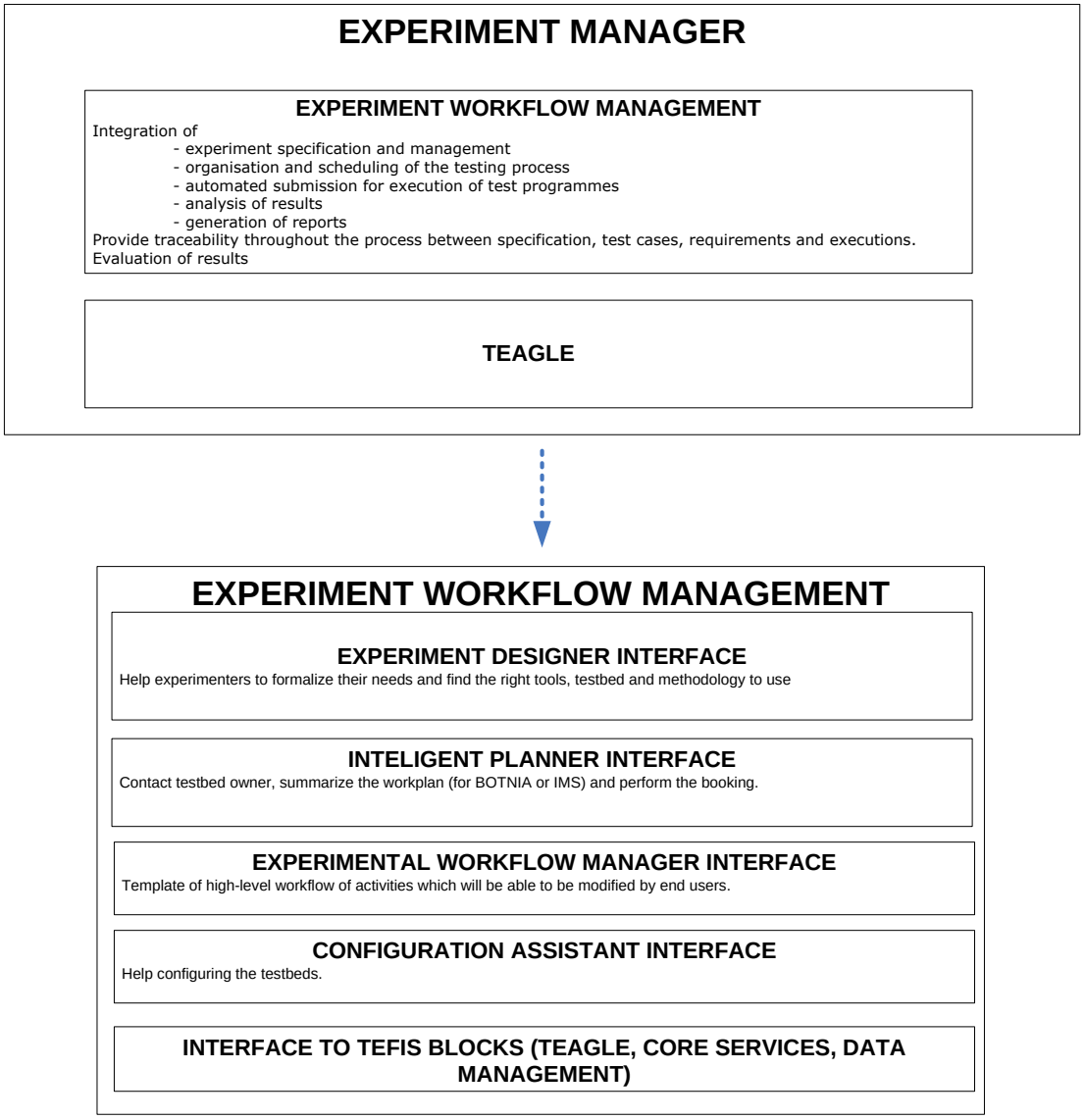

Figure 23: Experiment manager

The Experiment management user interface will allow the definition of experiments to be executed on a combination of TEFIS testbeds. It will make use of the TEFIS directory services to list available resources, allow configuration of resources and plan for experiment execution.

The interfaces will draw upon TEFIS core services and experiment management tools to allow for an easy interaction with the TEFIS platform.

Specifically, TEFIS users shall be supported in defining their experiments, executing them, and collecting the necessary data in order to produce valuable experimentation results. This will aid the overall vision of the FIRE initiative to allow for experimentally driven research where results from complex experiments can be obtained and compared while relying on a heterogeneous federated platform.

Summarising, it will provide the user the basic information it manages in order to deal with the experiment specification, scheduling and execution during the whole life-cycle.

In this context, the experiment manager will offer four interfaces to the user in order to ensure the completion of the experiment.

The user is understood as the final user of the Portal who is going to design and specify the experiment. Other user profiles as administrator or testbed provider will not be considered for the purposes of this document.

1 **Experiment designer** that will help experimenters to formalize their needs and find the right tools, testbed and methodology to use.

Experiment designer inputs:

- User needs
- Experiment specification and design

Experiment designer outputs:

- Test strategy
- Types of tests to perform: functional testing, integration testing, etc.

The personalization of the experiments makes it feasible to adapt the tools and testing methodologies to the specific profile of a user and the experiment they want to perform in order to establish a productive working environment.

The TEFIS portal will provide a powerful and customisable interface where the users will create environments adjusted to their needs.

The users could specify the types of tests they want to perform, the capacity, etc. in order to obtain an accurate Test strategy.

The user will receive as result a test strategy with at least one proposal of tools, testbeds, methodologies, facilities, etc. to use, but depending on the characteristics of the experiment it is possible that more than one proposal will be found that complies with the experiment requirements and needs.

They will also receive a specific test plan.

The basic characteristics of the interface will be the following:

- An experiment designer main page that will load a set of basic options related to the experiment specification.

  The options will open subsequent pages where the user could detail the concrete specifications of the experiment to perform such as technology, bandwith requirements, facilities, etc.

- Test strategy page with one or more strategies proposed by the TEFIS platform that satisfy the needs of the user.

2 **Experiment planner** that will establish contact with testbed owner and summarize the workplan (for BOTNIA or IMS) and perform the booking.

Experiment planner inputs:

- Test strategy
- User's specific criteria for the TEFIS strategy

Experiment planner outputs:

- Test plan

The experiment planner will be the point of access to the distributed testbed infrastructures, so once the user has specified and personalized the experiment, and a test strategy has been provided, the control will revert to this interface.

The user could reorganize all the knowledge about the experiment provided previously and link their specific needs to the options provided by the Test strategy.

In this way, the user could establish their own criteria with more concrete needs (dates, usage of resources, execution order, infrastructures, etc.) to the test strategy in order to obtain a tight Test plan.

The user could have access to this module when required in order to review the workplan defined in the Test plan and ask for the booking.

The basic characteristics of the interface will be the following:

- An experiment planner main page that will load the test strategy obtained during the previous phase.

  The user could review all the available options and proposals and establish the execution criteria, period of time, percentage of usage of the resources, etc. in order to configure a detailed Test plan.

  They could also add any comments they consider are essential to understand the workplan.

  The user could have access to this page in any moment and add or update the criteria or change the test plan.

- Test plan page with the final test plan configured by the user. It displays all the information related to the workplan of the experiment.

3 **Experimental workflow manager** that will propose a template of high-level workflow of activities which will be able to be modified by end users.

Experiment workflow manager inputs:

- Test plan

Experiment workflow manager outputs:

- Partial results of the execution of the activities
- Activities workflow

Once the users have finished the design and specification of the characteristics of the experiment, established a final Test plan and sent the booking request, the TEFIS platform, after completing the workflow process where a request is sent to the core services and the testbeds are managed, will obtain a complete list of activities and schedules.

The information related to the activities and their schedule will be shown to the user.

The user will then have access to a workflow of activities based on the experiment definition provided and the resources managed via testbeds.

This proposal could be updated by the user at any time removing activities, modifying specifications of some of them (change the execution order or time assigned for activities) or adding new activities. But in the case of modifying or adding the user must perform a booking update and a review of the test plan and test strategy so the core services can manage the new situation.

The basic characteristics of the interface will be the following:

- An experimental workflow manager main page that will display the set of activities proposed for the experiment workflow as a conceptual diagram.

  The user could link each of the activities and then have access to a more detailed description of the activity such as the schedule, resources involved, etc.

- Partial results page, although the entire experiment workflow would not have finished, the users could consult the information of the activities carried out up to that moment.

  In this way, the users could have access to the partial results obtained at any time and take decisions in respect of the progress of the experiment.

- Historical information as the same workflow of activities could have been carried out several times.

In this way, the user could have access to an experiment repository containing the historical information related to execution made for those experiments, the changes made to the workflow of activities, comments additionally introduced to detail the performance of the workflow at specific times, etc.

Furthermore, linked to the description the user would be offered direct access to the results obtained during that execution cycle. In this way, the user could assess the results obtained, the resources and facilities used, the time needed, etc.

Finally, the different workflow executions will be shown to the user ordered by date and the user can provide different names to every execution to identify them more easily.

In summary the complete use case executed in terms of the experience gained so the user can apply their know-how to other experiments that may share features in common.

4 **Configuration assistant** that will help configuring the testbeds. In this step the VCT Tool provided by Teagle will be a good basis for TEFIS work.

It will guide the user in definition, organisation and monitoring tasks of their experiment, providing an easy-to-use interface with a set of basic experiment features that the user will be able to choose and detail.

The more specific the definition of the experiment is, the more accurate will be the resources, facilities, tools and data provided to the user for the experiment by the TEFIS platform, so step-by-step and guided by the experiment configuration assistant, the user will detail the experiment in different phases.

The interface will help the user throughout the complete experiment specification and designing process, and it will present the following basic characteristics:

- A help page with the basic documentation about the TEFIS platform, which will include manuals, vodcasts or the link to the TEFIS project wiki with all public documentation available to the users and generated during the development of TEFIS.

- A dynamic FAQ section, constantly updated with the questions submitted by the users to the TEFIS administrator.

  It will be accessible at any time through a link included in the help menu and present in a web page a set of basic and more advanced questions with their respective answers related to the usage of the TEFIS platform

  - How to register a user

- How to specify a new experiment
- How to list the available resources and tools
- etc.

- More specific guidance with detailed and specific explanations

  It is essential to provide information about how to complete the questionnaires, fill the forms, review specifications and make changes, set the configuration and plan the schedule in terms of time, availability of resources and facilities, execute and re-launch a set of tests, support the methodology applied, or interpret the results, statistics, or diagrams provided.

  This specific guidance will be included in the controls to fill in, and the users can have access to them clicking the question mark linked to them.

- An Assistant could be called when the users are configuring the test plan or refining the activities to perform.

- A basic glossary of common terms used in the TEFIS context.

  It will provide the experimenters with a useful of knowledge base in case they are unsure about the meaning of several terms used in the Portal.

4.5.4 Experimental Data Interface

The experimental data interface will allow communication with the research platform repository service as defined by D2.1.1 and further WP6 deliverables. The interface will enable the TEFIS users to retrieve data related to their experiments. This includes test profiles, monitoring data, etc.

The easy access and aggregation of all experiment related data will be what will make this interface an added value of the TEFIS platform, instead of the experimenter logging into various resources in order to gather meaningful experiment results.

4.5.5 TEFIS Testbed Service Interface

Several TEFIS testbed providers expressed their need to expose testbed services directly on the Portal layer. This includes services that cannot be offered via the TEFIS platform for various reasons such as legal and operational restrictions regarding specific testbed resources.

Such interfaces will be implemented and integrated into the portal by the testbed providers. In this regard it is important for the portal to rely on a modular and extensible design in order to allow for easy integration of custom pages implemented by various TEFIS testbed providers. The MVC pattern as outlined in section 4.4.2 has been chosen with this requirement in mind.

References

[1] Sebastian Wahle, Thomas Magedanz, and Anastasius Gavras. Towards the Future Internet - Emerging Trends from European Research, chapter Conceptual Design and Use Cases for a FIRE Resource Federation Framework, pages 51-62. IOS Press, April 2010. ISBN: 978-1-60750-538-9 (print), 978-1-60750-539-6 (online). http://www.booksonline.iospress.nl/Content/View.aspx?piid=16471

[2] Teagle portal website: http://www.fire-teagle.org

[3] FP6 Panlab and FP7 PII projects website: http://www.panlab.net

[4] European Commission, Future Internet Research and Experimentation - FIRE Initiative website: http://cordis.europa.eu/fp7/ict/fire

[5] FP7 project PII Deliverable D3.7: Teagle Implementation Report, draft version, August 2010

[6] Sebastian Wahle, Thomas Magedanz, Anastasius Gavras, Halid Hrasnica, and Spyros Denazis. Technical Infrastructure for a Pan-European Federation of Testbeds. In Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops, 2009. TridentCom 2009. 5th International Conference on, pages 1-8, Washington DC, USA, April 2009. IEEE. ISBN: 978-1-4244-2846-5. http://dx.doi.org/10.1109/TRIDENTCOM.2009.4976205

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

[8] Federation Computing Interface (FCI) website: http://trac.panlab.net/trac/wiki/FCI

[9] Konrad Campowsky, Thomas Magedanz, and Sebastian Wahle. Resource Management in Large Scale Experimental Facilities - Technical Approach to Federate Panlab and PlanetLab. In 12th IEEE Network Operations and Management Symposium (NOMS 2010), pages 930 - 933, Osaka, Japan, April 2010. IEEE. ISSN: 1542-1201, Print ISBN: 978-1-4244-5366-5.

[10] TEFIS D2.1.1 – Initial Global architecture and overall design