

PII

Deliverable D3.5

Testbed Service Orchestration System Implementation

Editor:	Sebastian Wahle, Fraunhofer
Deliverable nature:	Prototype (P)
Dissemination level: (Confidentiality)	RE Restricted to a group specified by the consortium (including the Commission Services)
Contractual delivery date:	July 2009
Actual delivery date:	September 2009
Suggested readers:	PII consortium and community, EC, public
Version:	1.0
Total number of pages:	9
Keywords:	Teagle Implementation

Abstract

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

This document is part of the documentation on the Teagle_v1.0 prototype. It describes the Teagle VCT Tool, the Orchestration Engine, the Request Processor, and the Teagle Gateway on a very high level. Also, it provides the links to the online installations of the running prototypes as footnotes where appropriate.

Disclaimer

This document contains material, which is the copyright of certain PII consortium parties, and may not be reproduced or copied without permission.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the PII consortium as a whole, nor a certain party of the PII consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using this information.

Impressum

[Full project title] Pan-European Laboratory Infrastructure Implementation

[Short project title] PII

[Number and title of work-package] WP3, Service description, discovery and orchestration

[Document title] Testbed Service Orchestration System Implementation

[Editor] Sebastian Wahle, Fraunhofer

[Work-package leader] Sebastian Wahle, Fraunhofer

[Estimation of PM spent on the Deliverable] 30

Copyright notice

© 2009 Participants in project PII

1 Teagle

This document summarizes the work done for the testbed service orchestration system implementation which mostly relates to the development of the Teagle Orchestration Engine and the Teagle VCT Tool.

Teagle is the central search and composition engine of the Panlab testbed and experimental facility federation. It provides a web-based customer interface for browsing the federation’s offerings and the provisioning of Virtual Customer Testbeds (VCT).

A VCT is the sum of all resources and interconnections configured and rented by a specific customer. It is an isolated network where the customer has direct access to the resource and configurations assembled by using Teagle. Each customer operates inside its own VCT and has no access to other VCTs. Teagle combines several functions (Figure 1) that are currently implemented in a centralized manner (although distribution is generally possible but requires more complex trust hierarchy concepts):

- Registry & Repository (users, resources, configurations)
- Creation Environment (setup and configuration of VCTs, this is the VCT tool)
- Request Processor (for validating VCT configurations and trigger setup execution)
- Orchestration Engine (for generation of an executable workflow that orchestrates services form different domains to actually instantiate a VCT)
- Web Portal (for exposing search and configuration interfaces)

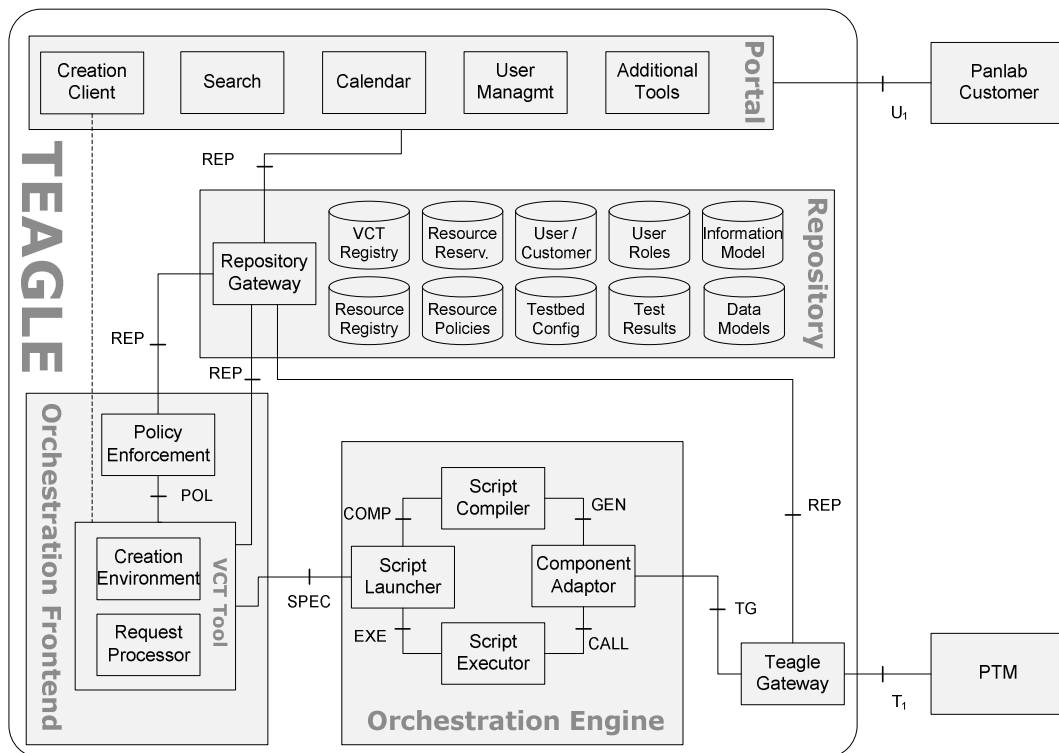


Figure 1: Teagle architecture overview

2 The VCT Tool

The VCT Tool enables the design of a desired VCT by offering a creation environment¹ as shown in Figure 2. It combines a selection panel on the left hand side that allows browsing available federation resources and getting information on their functions and availability with a workbench. Selected items (e.g. HSS, MySQL, etc.) can be placed and interconnected in the workbench. The arrows interconnecting the items have specific semantics. In the setup shown in Figure 2, the solid lines represent a protocol interconnection while the dotted lines reflect containment. This means for example that the MySQL server is hosted (contained) by the XenNode that itself is hosted by the physical node. On the other hand, the HSS is requesting data from the MySQL Server and a specific database via a query interface which requires specific configuration (e.g. MySQL Server address and port) on the HSS side.

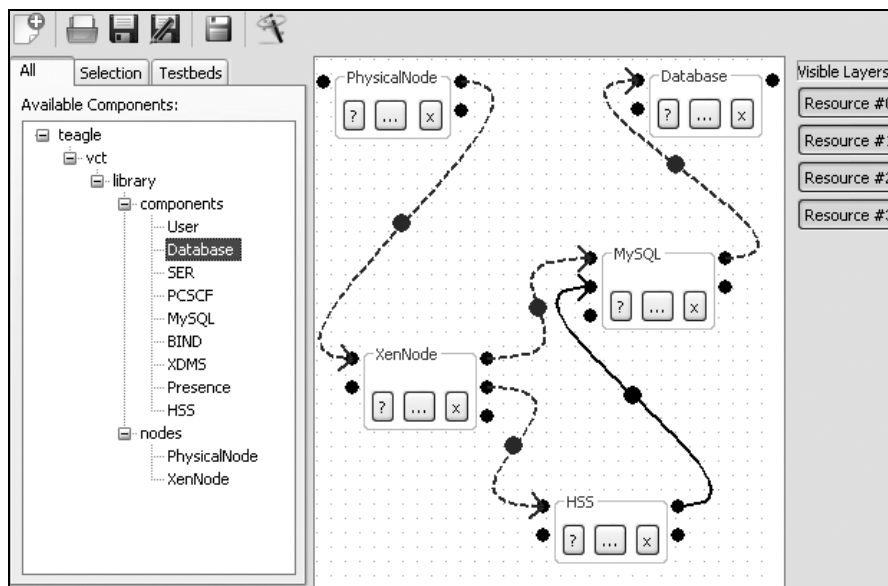


Figure 2: The VCT Tool as one of the main Teagle assets

The VCT layout and its associated resource configuration settings can be saved, upon which a XML document is produced. This output is passed to the Request Processor and then to the Orchestration Engine for further processing and execution. The Orchestration Engine transforms the VCT specification, which is a list of involved resources, their configuration, and some dependencies, into an executable script that performs the actual VCT instantiation, namely the provisioning of resources in different domains. Upon execution of the script, Web Service requests are sent to all involved Domain Managers triggering the deployment and configuration of resources according to the VCT specification.

For the development of the VCT Tool which is a Java implementation, we relied on the following technologies and software projects:

- Java SWT - The Standard Widget Toolkit for the VCT Tool window
- SWTCalendar for availability filtering functions on the selection panel
- Java XStream for serialization of objects to XML and back
- eXist Database for storing resource descriptions
- Apache log4j for logging at different levels like DEBUG, ERROR, INFO, etc.

The current VCT Tool prototype makes a compromise between the simplicity of web-based interfaces (no installation, no maintenance, portability of user profiles) and the power of desktop applications. In its current implementation, the VCT Tool runs as a Java Web Start application and can be launched

¹ <http://www.fire-teagle.org/members/vct.jsp> (credentials needed for access)

from the Teagle website² as needed. In order to avoid storing user profiles and VCTs on customers' local machines, the Web Start application uses the Teagle Repository (see D3.4) for storage.

The mechanism used for synchronizing access to the repository and Teagle components connected to it is the Model-View-Controller (MVC) paradigm. In MVC terms, the repository is the model, the VCT Tool represents the view and the Request Processor is the controller. Updates in the repository trigger notifications that components can register to. As an example, the Request Processor is notified about new VCTs inserted into the repository by the VCT Tool. It takes the steps necessary for provisioning these testbeds via the Orchestration Engine and pushing the result of the provisioning operation back into the repository.

² <http://www.fire-teagle.org/>

3 The Request Processor

As the Request Processor is an important component for the overall functioning of Teagle, it has been dedicated its own section, although it is part of the VCT Tool, according to Figure 1.

Items added into the repository (as triggered by the VCT Tool, such as VCTs, bookings, etc.) and modifications to existing data are processed by the Request Processor. Instead of requiring frontend tools such as the Creation Client to have knowledge about all Teagle components and their interfaces, this responsibility is centralized at the Request Processor, which implements conceptual operations in Teagle as one or several lower level backend requests.

Using this approach keeps the communication semantics between the VCT Tool and the repository to a minimum. More importantly, it hides orchestration complexities and the topology of services from the customer, exposing only an abstracted view and making automated operation a future opportunity.

The ability to register for notifications from the repository is not reserved exclusively to the Request Processor. Watching for updates on a running provisioning process can be coded as set of repository queries at regular intervals, and even plain email notification can be enough during prototyping. But the natural way, using the notification mechanism, is to have the VCT Tool register for changes on the state of a booking. Instead of re-implementing polling for all other asynchronous operations, this mechanism is reused for new resource types being added, as well as new resource instances being created.

The figure below illustrates the primary functionality of the Request Processor – booking a testbed. The series of messages involve the VCT Tool as the customer user interface, the Request Processor itself, the Repository and the Orchestration Engine. The message flow starts after the requested resources have been described and their availability has been verified.

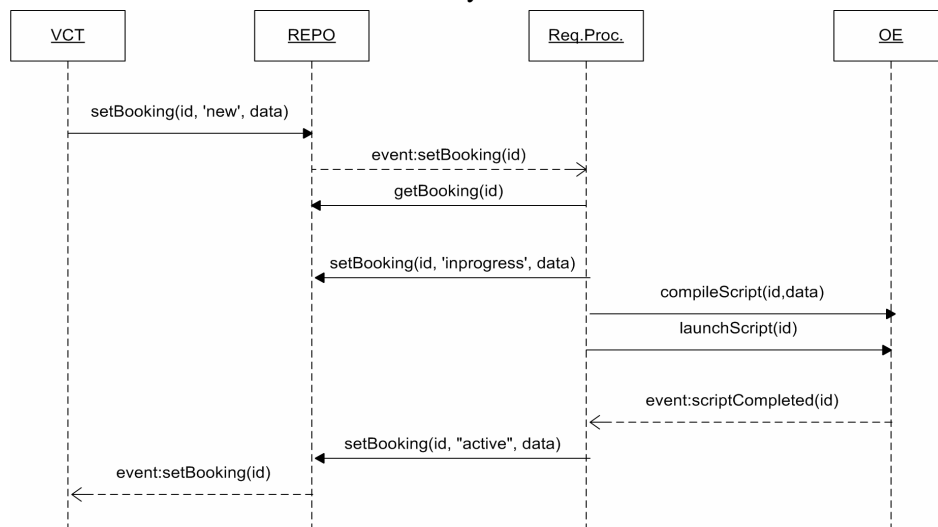


Figure 3 – Booking a new testbed

In the figure above, direct operations (on the Repository and Orchestration Engine) are represented by solid arrows. Notifications from the Repository for the VCT and the Request Processor are represented by dashed arrows. To reduce the traffic overhead, the data attached to notifications (usually a unique id) is just enough to allow components to obtain from the repository any additional information required for reacting to the operations they are interested in.

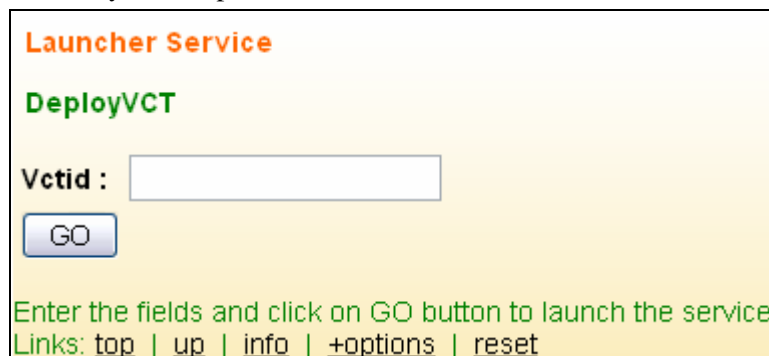
In its current implementation, the Request Processor is a servlet running in a J2EE container on the same physical machine as the Teagle Portal. Event notifications are sent by the Repository in the form of HTTP requests that contain the operation name and any additional parameters, as described above. With this approach, “Request Processor” can be not just the name of a standard component in the Teagle architecture, but can also denote a role for similar components that provide additional functionality. Examples of such functionalities include evaluation of customer requests based on policies, or triggering moderation for newly added resources.

4 The 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 (produced by the VCT Tool) 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 purpose.

The Orchestration Engine prototype³ is build on top of the Spatel Engine framework (see D3.3) which is based on SOA principles and finite state machines for service orchestration. All components inside Teagle (like Teagle Repository and the Teagle Gateway) and outside Teagle boundary (like 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).

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. A simulation mode can be used to trace the HTTP requests to be sent by the script.



Launcher Service

DeployVCT

Vctid :

GO

Enter the fields and click on GO button to launch the service

Links: [top](#) | [up](#) | [info](#) | [+options](#) | [reset](#)

Figure 4 - Web-based interface for compiling a VCT Layout

Figure 5 below shows an example of a very simple generated script for a VCT layout involving three components: MySQL, BIND and HSS. In this example we see a sequence of calls to the "create" operation in order to instantiate each of the involved resources. The implementation of these three calls results in three HTTP POST requests to comply with the REST-based design of the target testbeds implementation. The configuration data associated to each resource - initially included in the VCT export model, and stored in the script by the variables prefixed by 'conf' - is passed in the request as part of the XML data accompanying the HTTP POST request.

³ <http://mdasoa.elibel.tm.fr/teagle-site/> (credentials needed for access)

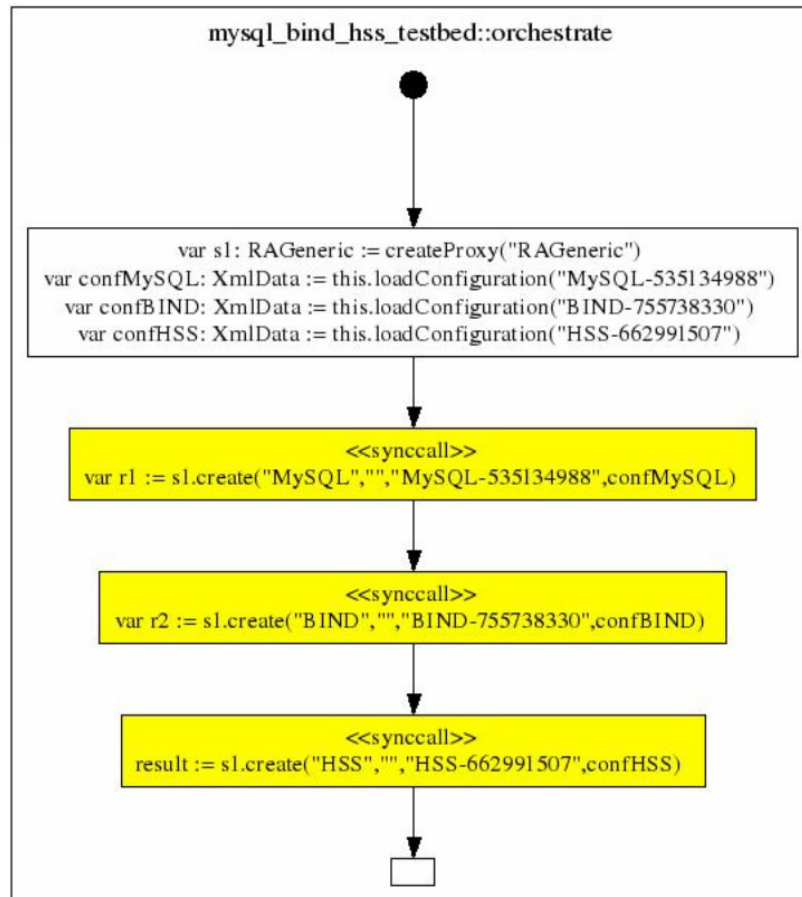


Figure 5: Example of generated orchestration script

The actual prototype of the Orchestration Engine translates the VCT layout into a sequence of synchronous service calls, meaning that long-running steps (like the initialization of a database) need to complete before a new step can be started. The translation algorithm will be refined in future releases of the prototype to optimize the provisioning process by enabling some form of parallelism.

5 The 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 regards the use of POST method for creating and updating resources, whereas GET method implies a *query* request.

The Teagle Gateway routes the requested actions through SOAP-based web services to the appropriate PTMs according to the protocol defined for T1 interface as described in D2.3. The application logic of the gateway, as far as the processing and forwarding of the requests is concerning, is made according to the following principles:

- 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 involved resource in that PTM
- The PTM identifier is used for selecting the proper web services endpoint for forwarding the request
- The resource identifier is used according to T1 definition in arguments of the subsequent call to the identified PTM
- The data (if any, depending on the actual web method used) are inspected for clarifying the requested CRUD operation. This is useful in the case of a POST request. In case an update attribute has been included in the root element of the data, an *update* request will be sent to the relevant PTM. Otherwise a *create* request will be invoked.
- The posted data are further inspected for locating any references to other resources that have been included according to the script execution in the specific orchestration request. In such case, the gateway is resolving the reference, which is presented as a PII resource ID, by replacing the data in the element containing the reference with the data provided by the referenced resource after querying it with a specific query type (“reference”). The data obtained by issuing this kind of query are resource specific and can be translated by the Resource Adaptor that is the recipient of the orchestration request. A very simple example can be the resolution of the IP address of a specific application server that has been referenced in the configuration requested directed to a potential client of that server. This feature allows for associating resources located in different PTMs during VCT design.

The next version of the Teagle Gateway will be providing an event reporting mechanism to be utilised by the PTMs and it will be offered as a SOAP-based web services endpoint. It will also include an event processing mechanism that will be invoked prior to forwarding the event in the appropriate entities (OE or Repository). This event reporting mechanism will allow for utilisation of asynchronous requests towards the PTMs in case of lengthy operations. In this case the outcome of a request will be collected and forwarded to the OE when it becomes available by the PTM. Also, the notification mechanism will cater for other types of functionality such as: PTM registration and status reporting, resource registration, availability, and status.

The Teagle Gateway is implemented as a standalone Java application integrating:

- A Jersey Engine for supporting RESTful web services
- A Grizzly servlet-web server for supporting the Jersey Engine
- An Axis-1.4 Engine with embedded Jetty web server for supporting SOAP web services