# Abstract

This deliverable comprises the specification of the service description system and is part of the specification of Teagle.

Break down the requirements from Task 3.1 into technical requirements for the specification of testbed services description system. The specification will be linked closely with the architecture work developed in task T2.1. Develop an advanced service description system based on the requirements and the specification. Implement the core functionalities of the description system.
Disclaimer

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

_In case of Public (PU):_

All PII consortium parties have agreed to full publication of this document.

_In case of Restricted to Programme (PP):_

All PII consortium parties have agreed to make this document available on request to other framework programme participants.

_In case of Restricted to Group (RE):_

All PII consortium parties have agreed to full publication of this document. However this document is written for being used by <organisation / other project / company etc.> as <a contribution to standardisation / material for consideration in product development etc.>.

_In case of Consortium confidential (CO):_

The information contained in this document is the proprietary confidential information of the PII consortium and may not be disclosed except in accordance with the consortium agreement.

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 Description Specification
[Editor: Name, company] Eamonn Power, TSSG
[Work-package leader: Name, company] Sebastian Wahle, Fraunhofer FOKUS
[Estimation of PM spent on the Deliverable] 14

Copyright notice

© 2008 Participants in project PII

Optionally list of organisations jointly holding the Copyright on this document
Executive summary

The Pan-European Laboratory is based on a federation of distributed testbeds. To manage this federation, a centralized support service is required. This is realised in the form of the TEAGLE search and composition engine. TEAGLE provides a web based interface for customers to browse federation offerings and instantiate Virtual Customer Testbeds (VCTs). It also provides a repository for storing testbed service descriptions for federation partners. This system should allow testbed partners to describe their resources in the context of a shared information model. This description should also support the orchestration system that will be used to instantiate VCTs. For this reason a centralized managed repository of descriptions is required. This repository should also support related information such as access policies, configuration information, user management and resource reservations.

To suitably design such a repository, requirements were examined from the Use Cases provided by WP1, the underlying architecture provided by WP2 and analysis of existing testbeds conducted in WP3. Related discussions were also considered from FIREweek and the Future Internet Assembly (FIA).

Given that the resource descriptions were anticipated to be quite heterogeneous, it was decided an overall information model would introduce consistency in the structure of these descriptions. This information could then be translated to a number of data sources where managed access can be provided to multiple TEAGLE components. Access to this information should be realised as standards based services which will allow integration and composition within TEAGLE.
## List of authors

<table>
<thead>
<tr>
<th>Company</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSSG</td>
<td>Eamonn Power</td>
</tr>
<tr>
<td>TSSG</td>
<td>Shane Fox</td>
</tr>
<tr>
<td>OCTO</td>
<td>Jussi Mäkinen, Marko Palola</td>
</tr>
<tr>
<td>ITL</td>
<td>Fausto Giovanni Andreotti</td>
</tr>
<tr>
<td>COSMOTE</td>
<td>George Korinthios, Niki Marouda, George Lyberopoulos</td>
</tr>
<tr>
<td>UoP</td>
<td>Christos Tranoris</td>
</tr>
<tr>
<td>UniPa</td>
<td>Vincenzo Mancuso, Silvana Greco Polito</td>
</tr>
<tr>
<td>Fraunhofer FOKUS</td>
<td>Thomas Magedanz, Florian Schreiner, Sebastian Wahle</td>
</tr>
<tr>
<td>Nokia</td>
<td>Sami J. Hämäläinen</td>
</tr>
<tr>
<td>FT</td>
<td>Mariano Belaunde</td>
</tr>
</tbody>
</table>
# Table of Contents

1 Introduction .............................................................................................................................. 13
  1.1 Objective of Task 3.2............................................................................................................ 13
  1.2 Objective of this document ............................................................................................... 13
2 Requirements................................................................................................................................. 14
  2.1 Sources ................................................................................................................................ 14
    2.1.1 From Use Cases ............................................................................................................. 14
      2.1.1.1 Class 1: Outsourced Testing and Testbeds .............................................................. 14
      2.1.1.2 Class 2: Test of Services and Applications ............................................................ 15
      2.1.1.3 Class 3: Interoperability, Conformance and Certification ...................................... 16
    2.1.2 From Underlying Architecture (D2.1), Existing Testbed Repository and System Analysis (D3.1) ................................................................................................................. 17
    2.1.3 From FIRE Week ........................................................................................................... 19
    2.1.4 From Future Internet Assembly ...................................................................................... 20
3 Alignment with Architecture ....................................................................................................... 21
  3.1 Summary of Underlying Architecture (in D2.1) .................................................................... 21
4 State of the Art .............................................................................................................................. 23
  4.1 Introduction ........................................................................................................................... 23
    4.1.1 Repository ..................................................................................................................... 23
    4.1.2 Information Categories .................................................................................................. 23
      4.1.2.1 Resource Reservation .............................................................................................. 23
      4.1.2.2 Orchestration Persistence ......................................................................................... 23
      4.1.2.3 PII Logging .............................................................................................................. 23
      4.1.2.4 Customer and User Information .............................................................................. 25
    4.1.3 Databases ...................................................................................................................... 25
      4.1.3.1 Relational Databases ............................................................................................... 25
      4.1.3.2 XML Native Databases .......................................................................................... 26
  4.2 Information and Data Models ................................................................................................. 26
    4.2.1 Shared Information Data Model (SID) ............................................................................ 26
      4.2.1.1 IPSphere ................................................................................................................ 28
      4.2.2 DEN-ng ...................................................................................................................... 29
  4.3 Languages & Tools .................................................................................................................. 30
    4.3.1 Languages .................................................................................................................... 30
      4.3.1.1 UML/Class Diagrams ............................................................................................. 30
      4.3.1.2 WSDL ................................................................................................................... 30
      4.3.1.3 RDF ...................................................................................................................... 31
      4.3.1.4 SoAML .................................................................................................................. 33
      4.3.1.5 Semantic Annotations ............................................................................................ 34
        4.3.1.5.1 Semantic Annotations for UML ....................................................................... 34
        4.3.1.5.2 SAWSDL .......................................................................................................... 34
        4.3.1.5.3 OWL ............................................................................................................... 35
    4.3.2 Tools ............................................................................................................................... 36
      4.3.2.1 Introduction ............................................................................................................. 36
5 Specification .............................................................................................................................. 40
  5.1 Introduction .......................................................................................................................... 40
  5.2 Requirement Index ............................................................................................................... 40
  5.3 Operational Aspects .......................................................................................................... 43
    5.3.1 Activity 1: Customer creates a testbed and an SLA is established.............................. 43
    5.3.2 Activity 2: Provisioning according to a SLA ................................................................. 45
    5.3.3 Activity 3: Partner Interaction ...................................................................................... 47
  5.4 Entities ................................................................................................................................ 48
    5.4.1 Teagle Portal .................................................................................................................. 48
      5.4.1.1 Search ..................................................................................................................... 49
      5.4.1.2 Calendar ................................................................................................................. 49
      5.4.1.3 User Management ................................................................................................. 49
      5.4.1.4 Visualisation Tool ................................................................................................. 49
      5.4.1.5 Creation Client ........................................................................................................ 49
      5.4.1.6 Additional Tools ..................................................................................................... 50
    5.4.2 Repositories .................................................................................................................. 50
      5.4.2.1 Repository Gateway .............................................................................................. 50
      5.4.2.2 VCT Registry ......................................................................................................... 50
      5.4.2.3 Resource Registry ................................................................................................ 50
      5.4.2.4 Resource Reservation ............................................................................................ 50
      5.4.2.5 Resource Policies .................................................................................................. 50
      5.4.2.6 User/Customer ........................................................................................................ 50
      5.4.2.7 Testbed Configuration ............................................................................................ 51
      5.4.2.8 User Roles .............................................................................................................. 51
      5.4.2.9 Test Results ............................................................................................................. 51
      5.4.2.10 Information Model .............................................................................................. 51
      5.4.2.11 Data Models .......................................................................................................... 51
    5.4.3 Orchestration Frontend ................................................................................................... 51
      5.4.3.1 Policy Enforcement ............................................................................................... 51
      5.4.3.2 Creation Environment .......................................................................................... 52
      5.4.3.3 Request Processor ................................................................................................. 52
    5.4.4 Orchestration Engine ...................................................................................................... 52
    5.4.5 Teagle Gateway ............................................................................................................. 52
  5.5 Reference Points ................................................................................................................. 52
    5.5.1 External Reference Points ............................................................................................. 52
      5.5.1.1 REP1 ...................................................................................................................... 52
      5.5.1.2 REP2 ...................................................................................................................... 53
  6 Conclusion ............................................................................................................................... 54
List of figures and/or list of tables

Figure 1: Overall PII Architecture........................................................................................................21
Figure 2: Inferring interoperability by using data models that are subsets of the information model...28
Figure 3: First and Fundamental Classification in DEN-ng................................................................29
Figure 4: Customer creates a testbed and an SLA is established..........................................................44
Figure 5: Provisioning according to a SLA..........................................................................................46
Figure 6: Partner Interaction..................................................................................................................47
Figure 7: Teagle Internal Architecture..................................................................................................48
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACF</td>
<td>Autonomic Communication Forum</td>
</tr>
<tr>
<td>ABE</td>
<td>Aggregate Business Entity</td>
</tr>
<tr>
<td>BMM</td>
<td>OMG Business Motivation Model</td>
</tr>
<tr>
<td>BPDM</td>
<td>OMG Business Process Definition Metamodel</td>
</tr>
<tr>
<td>CCM</td>
<td>CORBA Component Model</td>
</tr>
<tr>
<td>CIM</td>
<td>Common Information Model</td>
</tr>
<tr>
<td>DEN-ng</td>
<td>Directory Enabled Networks-new generation</td>
</tr>
<tr>
<td>DMTF</td>
<td>Distributed Management Task Force</td>
</tr>
<tr>
<td>EAST-ADL</td>
<td>EAST Architecture Description Language</td>
</tr>
<tr>
<td>EMF</td>
<td>Eclipse Modelling Foundation</td>
</tr>
<tr>
<td>eTOM</td>
<td>enhanced Telecom Operations Map</td>
</tr>
<tr>
<td>FIA</td>
<td>Future Internet Assembly</td>
</tr>
<tr>
<td>GEF</td>
<td>Graphical Editing Framework</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IGW</td>
<td>Interconnection Gateway</td>
</tr>
<tr>
<td>JET</td>
<td>Java Emitter Templates</td>
</tr>
<tr>
<td>JUL</td>
<td>Java Sun’s java.util.logging package</td>
</tr>
<tr>
<td>LwCCM</td>
<td>Lightweight CCM</td>
</tr>
<tr>
<td>MARTE</td>
<td>Modelling and Analysis of Real Time and Embedded systems</td>
</tr>
<tr>
<td>MOF</td>
<td>Meta-Object Facility</td>
</tr>
<tr>
<td>NGOSS</td>
<td>New Generation Operations Software and Systems</td>
</tr>
<tr>
<td>NXD</td>
<td>Native XML Databases</td>
</tr>
<tr>
<td>OCL</td>
<td>Object Constraint Language</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>PTM</td>
<td>Panlab Testbed Manager</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDFS</td>
<td>Resource Description Framework Schema</td>
</tr>
<tr>
<td>SAWSDL</td>
<td>Semantic Annotations for WSDL and XML Schema</td>
</tr>
<tr>
<td>SID</td>
<td>Shared Information and Data Model</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SoaML</td>
<td>Service oriented architecture Modelling Language</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL Protocol and RDF Query Language</td>
</tr>
<tr>
<td>SysML</td>
<td>Systems Modelling Language</td>
</tr>
<tr>
<td>TMForum</td>
<td>TeleManagement Forum</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>UML-DI</td>
<td>UML Diagram Interchange</td>
</tr>
<tr>
<td>VCT</td>
<td>Virtual Customer Testbed</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modelling Ontology</td>
</tr>
<tr>
<td>XCAP</td>
<td>Extensible Markup Language Configuration Access Protocol</td>
</tr>
<tr>
<td>XMI</td>
<td>XML Meta Interchange</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XSLT</td>
<td>Extensible Stylesheet Language for Transforms</td>
</tr>
</tbody>
</table>
## Definitions

The following is a list of definition and terms used in the main body of this document.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>The term user is referring to a person that is actively using or testing services provided by a customer’s resource inside the testbed federation. Differentiation between End-User and Test-User may only be useful when directly referring to the special task of active service testing or the usual task of common service usage. End-User is also a person acting as a part of User Driven Innovation (UDI) process, thus collecting information and giving feedback to Customers for instance in Living Labs environments’ field testing projects.</td>
</tr>
</tbody>
</table>
| User Driven Innovation | Inclusion of the user as part of the testing process in order to take into account early feedback. In this project two levels of User Driven Innovation (UDI) will be applied.  
  - The first level refers to the customer (the organisations and companies that use the federation to test their products and services).  
  - The second level refers to the potential end-users of the services.  
To generally differentiate between these two groups in this document, we will refer to the first as federation-customers, service-testers or service-providers and the second we will refer to as end-user. |
| Provider          | This term refers to a party owning testing infrastructure and other testing resources and has entered an agreement allowing a customer to use its testing infrastructure and resources to develop and test services.                                                                                                                                                                                                                                                                                                                                                     |
| Testbed           | A testbed is an environment which allows experimentation and verification for research and development products. A testbed provides rigorous, transparent and replicable testing and herein it is always used in the context of new information and (tele-)communications technologies for networks and services.                                                                                                                                                                                                                                                          |
| Testbed federation | A testbed federation or federated testbeds is the interconnection of two or more independent testbeds for the creation of a richer environment for testing and experimentation, and for the increased multilateral benefit of the users of the individual independent testbeds.                                                                                                                                                                                                                                                                       |
| Customer          | A customer of Panlab is someone that used TEAGLE to set up testbed
resources for the purpose of testing or developing services. The customer is able to directly connect to the rented resources (his VCT) using a VPN client (U3 interface).

The customer can offer his services under the terms of Panlab user domain access (U2 interface) to external Test-/End-Users.

<table>
<thead>
<tr>
<th>Testing session</th>
<th>Herein the term testing session refers to a well-defined temporal and spatial relation of testing infrastructures and testing resources by the customer or user(s).</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCT</td>
<td>A Virtual customer Testbed is the sum of all resources, including interconnections, rented by the customer. It basically is an isolated network in that the customer is able to “dial in” and directly access the items assembled in TEAGLE. Each customer operates inside its own VCT and has no access to other VCTs. The purpose of direct VCT access is to configure and develop but not to widely test new services; such testing is intended to be done from outside using external Test- and/or End-Users.</td>
</tr>
<tr>
<td>Information Model</td>
<td>An information model is a platform independent model of entities and high-level relationships linking them.</td>
</tr>
<tr>
<td>Data Model</td>
<td>A data model is a platform specific model of entities and lower level relationships linking them. From RFC 3444: “They are intended for implementers and include protocol-specific constructs”.</td>
</tr>
</tbody>
</table>
1 Introduction

This document identifies, discusses and addresses issues surrounding the definition and implementation of the Testbed Service Description Specification. This work is carried out as part of WP3, which is tasked with describing and implementing the Teagle. Teagle is the component in the PII testbed federation that will make the testing resources on offer available to PII Customers.

1.1 Objective of Task 3.2

From the PII Description of Work [OBJ_1] the objectives of WP3 are as follows:

- To carry out a system analysis and to specify the testbed requirements from a user point of view
- To develop a testbed service description system as well as an orchestration system
- To implement the necessary support tools

The second point relates specifically to this task and is further described in [OBJ_1] as follows: “Break down the requirements from Task 3.1 into technical requirements for the specification of a testbed services description system. The specification will be linked closely with the architecture work developed in task T2.1. Develop an advanced service description system based on the requirements and the specification. Implement the core functionalities of the description system.”

1.2 Objective of this document

The objective of this document is to complete the D3.2 deliverable which is defined in the PII Description of Work [OBJ_1] as, “This deliverable comprises the specification of the service description system and is part of the specification of Teagle”. This document takes both the task and deliverable objectives into consideration.

The requirements for the service description system are gathered from a variety of sources in Section 2 of the document and later mapped to specific components of the Teagle in Section 5, by which point the architecture of the Teagle has been outlined. Once the requirements are outlined, the technical issues of how to describe, store, discover, broker and provision the testbed resources are examined. An information model will provide the structure to model the testbed resources as managed entities and possibly also provide operation support systems. A number of state of the art information models are examined in order to find a best-fit for the service description system requirements. The state of the art in both languages and tools capable of implementing the testbed service descriptions are explored. A number of the other modules that comprise the Teagle such as the repositories, resource reservations, logging, etc. are described and the state of the art technologies that meet the requirements of these modules are outlined. In the final sections of the document, the components of Teagle are mapped to the most suited technologies and a conclusion of the deliverable’s work is provided.

The testbed service description should also specify in detail the interconnections between the elements of the technological playground and the players, i.e. human beings in their various roles utilizing the tools of Testbeds.
2 Requirements

2.1 Sources

This section gathers all requirements relevant for the testbed description specification. Different requirements sources have been considered. The use case analysis carried out as part of D1.1 [REQ_1] served as a first source. Furthermore, the general architecture (D2.1) [REQ_2] and system analysis (D3.1) [REQ_3] documents provide some requirements that are outlined in their respective subsections below. Finally, the FIRE Week that took place at Les Cordeliers, UPMC Paris Universitas in September 2008 [REQ_4] is considered as a source of requirements in subsection 2.1.3.

2.1.1 From Use Cases

This section outlines the basic requirements for the provisioning of resources and services within the PII architecture, as derived from the analysis of the Use Cases presented in D1.1 [REQ_1]. These requirements are mapped into specific technology characteristics in order to advance the discussion towards deciding on a suitable service description model.

These requirements are labelled in accordance with the following naming scheme:

##D3.2- Source##M.ReqNumber, where

- **Source**: name of the Source/Origin (Use cases, Existing Testbeds or D3.1, Architecture, etc.) from where stems the requirement.
- **ReqNumber**: A number identifying the requirement within each category.

NOTE: Two consecutive numbers differ in 50. This allows including new requirements between two already existing ones without changing the numbering.

2.1.1.1 Class 1: Outsourced Testing and Testbeds

Requirement ##D3.2-CL1 Use Cases##M.0100

Statement: Teagle Portal shall incorporate a VCT request designer which will enable Customers to request/define a desired testbed facility or topology.

Justification: Panlab Customers should be able to formulate requests through the Teagle Portal for specific testing facilities or testbed topologies (testbed components, interfaces, testbed capabilities etc.). The Teagle Portal should provide a user friendly interface for customers to discover available resources (search the testbed repository for available resources) and specify their VCT requests and the corresponding tests to be performed when the tests have been provisioned. This is made possible by means of an input interface (uniform input model) that will ensure a common understanding of the input received.

Requirement ##D3.2-CL1 Use Cases##M.0150

Statement: Teagle Portal shall present Customers with feedback on the availability of requested resources, testing facilities or testbed topologies.

Justification: Customers should receive information regarding the availability of specific testing facilities or testbed topologies and on whether the testbed functionalities are limited towards the federation or also available for local hosting at the Customer’s premises through the Teagle Portal. Teagle should be able to retrieve such information by means of communicating with a Reservation System keeping track of resources.

Requirement ##D3.2-CL1 Use Cases##M.0200

Statement: Software Images of testbed components should be provided in the form of a resource stored in a Central Repository.
Justification: The prerequisite for a customer to host a testbed locally is the discovery and download of suitable software images which could be easily deployed in standard virtual machines at the customer’s premises. Teagle shall store software images of software components belonging to testbeds of Panlab Partners in a central Panlab repository along with a description regarding capabilities and possibly a tested configuration. This will allow for software images, uniquely identified within the repository (name, version number) to be treated as Panlab resources which shall be discovered via Teagle’s Portal and ‘Search’ tool and downloaded from the repository via the UI interface (Teagle Portal). The Software Repository shall also have the ability to accommodate new software revisions uploaded by Panlab partners.

Requirement ##D3.2-Cl.1 Use Cases##M.0250
Statement: Panlab federation shall provide support for the set-up of an SME local testbed in the form of a service offered to Panlab customers.

Justification: If support is required regarding the deployment of a software image and the setup of the local testbed, the federation shall provide responsible contact persons from the respective PII partner testbeds. A customer shall be able to request and ‘book’ assistance through the Teagle Portal. Panlab partner contact persons could also be individually stored in a Reservation System in order to allow for tracking of their availability and for a calendar based reservation.

Requirement ##D3.2-Cl.1 Use Cases##M.0300
Statement: For an outsourced testbed environment, test results should be gathered exclusively by the customer without intervention of the Panlab Office.

Justification: To enhance privacy and security, test results should be collected at the customer premises by the customer alone. Test data shall not be communicated towards Teagle’s Result database.

2.1.1.2 Class 2: Test of Services and Applications

Requirement ##D3.2-Cl.2 Use Cases##M.0350
Statement: The Panlab Office and the Panlab federated testbed shall be able to provide users/user groups for testing purposes in the form of a resource.

Justification: Panlab Customers may require user groups to setup test environments, execute tests (e.g. for the evaluation of a new service) or even collect the test data and analyze the test results. Panlab should be able to provide a variety of specialized user groups (different profile, size, etc.) available to perform tests, in the form of a resource. Real user groups could also be provided for the purpose of collecting experience regarding the usability of a new application/service. Panlab user groups could be individually stored in a Reservation System in order to allow for tracking of their availability and for a calendar based reservation.

Requirement ##D3.2-Cl.2 Use Cases##M.0400
Statement: Teagle’s VCT request designer (Teagle Portal) shall also enable Customers to request testing environments over which specific services developed by third parties (not belonging to the federation) could be tested (sometimes concurrently in different environments).

Justification: A Panlab Customer may wish to use the PII platform to test a 3rd party service over different technologies and by different user groups. The Customer should be able to formulate a request through Teagle portal for a specific technical setup. The request may also include specific tests to be performed by specific testing groups and a desired type of test results.

Requirement ##D3.2-Cl.2 Use Cases##M.0450
Statement: Teagle’s VCT request designer (Teagle Portal) shall enable Customers (SMEs/Users) to request testing environments over which specific components (network components, terminals) or applications developed by third parties (not belonging to the federation) could be tested (Support for User Driven Innovation).

Justification: A Panlab Customer may wish to use the PII platform to test a 3rd party component or application (sometimes even by different user groups). The Customer should be able to formulate a request through Teagle portal for a specific technical setup. The request may also include specific tests to be performed by specific testing groups and a desired type of test results.

**Requirement ##D3.2-Cl.2 Use Cases##M.0500**

Statement: Teagle Portal shall provide the means for Customers to view current/past test results and logs.

Justification: Customers should be able to retrieve/view the results of the tests they requested in their VCT request as well as test traces and logs through Teagle Portal. Teagle should be able to retrieve such information by means of communicating with the Panlab Results Repository (Test Result Database) and the Panlab Log Repository (in case logs are stored in a different repository). Moreover, Teagle Portal shall allow for access to past supplied test results in a similar manner.

2.1.1.3 Class 3: Interoperability, Conformance and Certification

**Requirement ##D3.2-Cl.3 Use Cases##M.0550**

Statement: Teagle Portal (VCT request designer) should enable customers to define and configure the environment in which interoperability, conformance and certification testing shall be performed.

Justification: SMEs, Service Operators and Research Institutions may wish to utilize the PII infrastructure to perform interoperability, conformance and/or certification testing of a network component/device, a service, a protocol or an algorithm. Customers shall be able to define the testing environment in which the network component to be tested shall be integrated. Customers shall also be able to provide configuration instructions regarding the test setup and the tests to be performed (Teagle Portal).

**Requirement ##D3.2-Cl.3 Use Cases##M.0600**

Statement: Panlab platform shall reserve and allocate all resources required to enable integration with the component/service/protocol to be tested and create a testing experimentation field in which the interoperability/conformance/certification testing can be performed.

Justification: For interoperability/conformance/certification testing to be supported with regard to network components/procedures, QoS procedures, protocol implementations etc., Panlab platform must support the creation of an appropriate testing facility. Once the Customer’s VCT request has been analyzed (Orchestration Frontend) and an Orchestration Specification has been generated by Teagle, it should be converted into a sequence of service requests issued towards the various PTMs (Panlab Testbed Managers) which, in turn, shall reserve the resources and setup the test environment according to the information received.

**Requirement ##D3.2-Cl.3 Use Cases##M.0650**

Statement: Teagle portal should maintain a Test suite repository

Justification: Teagle portal should be able to maintain test suites from past testbeds in order to achieve testing efficiency to avoid duplicating the same work when new tests must be hosted on the Panlab platform. This repository will contain test suites provided by Panlab partners. The maintenance includes: add, delete, and modify test suites, goals achieved and probably results from previous executions.
Requirement ##D3.2-Cl.3 Use Cases##M.0700
Statement: Teagle portal should provide a search service of the Test suite repository.
Justification: A Panlab customer should be able to retrieve and reuse test suites from past testbeds by making a simple search on past test suite specifications. The results of this search should be, for example: test definition, testing tools, test suite and past results. Confidentiality of the data stored in the repository should be taken into careful consideration. Customers should be classified according to their confidentiality and privacy classification.

Requirement ##D3.2-Cl.3 Use Cases##M.0750
Statement: Teagle portal should provide a generic search service of the available testing tools.
Justification: A Panlab customer should be able to retrieve available testing tools according to their search criteria via the Teagle portal. The results could be: topology, physical locations, availability, legal issues, etc.

Requirement ##D3.2-Cl.3 Use Cases##M.0800
Statement: Teagle portal should maintain a certification database of test suites
Justification: A Panlab customer should be able to certify his provided service. Teagle portal should be able to maintain different certification processes and should provide to the customer all the necessary test suites that his service must pass in order to be certified. Thus a certification process could be associated with certain test suites

2.1.2 From Underlying Architecture (D2.1), Existing Testbed Repository and System Analysis (D3.1)
Requirements stemming from D2.1 and the overall architectural requirements as well as the existing testbed repository (http://www.panlab.net/testbed-repository.html) are outlined.
As D3.1 [REQ.3] is currently a snapshot of what is described in the testbed repository, the requirements produced coincide with those of the testbed repository.

Requirement ##D3.2-Architecture##M.0850
Statement: Teagle Portal shall support the establishment of Service Level Agreements (SLA) between Customers and the Panlab Office.
Justification: Prior to Testbed instantiation, a SLA shall be established between the Customer and the Panlab Office through the Teagle Portal. The SLA shall have the form of a binding contract between both parties and will be used to analyze customers’ requests and ensure that the requested resources/testbed facilities are provided to customers. Contractual aspects such as testbed cost, access rights and ownership of test results shall be included in the SLA. A VCT request shall be constrained by an established SLA.

Requirement ##D3.2-Architecture##M.0900
Statement: Repository should be capable of storing Customer information
Justification: A User/Customer Database should exist as part of the Panlab repository. It shall store subscriber and account details and will enable the authentication, authorization and control of access to Panlab resources during the initial test environment configuration phase, the testing phase, the monitoring phase and the test result collection phase. Confidentiality of the data stored in the repository should be taken into careful consideration. Customers should be classified according to their confidentiality and privacy classification.

Requirement ##D3.2-Architecture##M.0950
Statement: Teagle should include a Resource Reservation System

Justification: VCT availability should depend on the availability of testbed resources. Teagle should provide information to customers regarding the availability of VCTs based on the availability/status of resources. A Resource Reservation System shall uniquely identify resources registered with the Federation and keep track of their availability. It shall have the ability to reserve resources for a certain amount of time (calendar based reservation) during which other reservations should not be allowed. For VCT requests, the Orchestration Frontend shall resolve dependencies between resources.

**Requirement ##D3.2-Architecture##M.1000**
Statement: Teagle shall incorporate an intelligent Orchestration Frontend.

Justification: A VCT request may include various resources belonging to different Panlab Partners of the federation. The VCT generation tool may not always be able to resolve dependencies between resources. An Orchestration Frontend shall analyze VCT requests; resolve all dependencies and pre-/post- conditions between different resources to generate an Orchestration Specification. In case the VCT generation tool generates an Orchestration Specification, the Orchestration Frontend shall simply forward it. The Orchestration Frontend shall also communicate with the databases storing the Information and Data Models.

**Requirement ##D3.2-Architecture##M.1050**
Statement: Repository should be capable of storing Test Result information

Justification: A Test Result Database should exist as part of the Panlab repository. It shall store test results and test traces in an organized manner according to a common information model. Stored test results shall be used for further processing (report generation) or as a value-adding information for the Panlab platform. Access to the database shall be subject to pre-defined access rules (Constraints/Policies Database). Test results shall be uploaded to the repository, residing within Teagle, through the T1 interface. Uploading is performed by the Test Results Management module of the Panlab Testbed Manager (PTM) belonging to the Panlab testbed that produced the Test Data (SOAP unified messages).

**Requirement ##D3.2-Architecture/System Analysis##M.1100**
Statement: Databases storing Shared Information and Data Models shall be part of the Panlab repository.

Justification: Due to the heterogeneous nature of testbeds, Panlab should adopt a Shared Information Model. This will provide a structured framework used to store definitions of entities, their attributes and the relationships between entities, in a common, understandable way. The model will allow for components to be searched, discovered and provisioned by an automated system. Shared Information and Data models (subsets of the Shared Information Model) should be stored in Databases that will be part of the Panlab Repository.

**Requirement ##D3.2-Architecture/System Analysis##M.1150**
Statement: PTMs should allow for interconnection with 3rd party components/resources.

Justification: The Panlab platform shall support the integration of components/resources provided by customers. PTMs receiving control messages and instructions from Teagle regarding the installation of specific test environment set-ups including 3rd party components, shall correspond to the implementation of specific resource configurations and shall support connectivity between testbed and Customer components.

**Requirement ##D3.2-Architecture##M.1200**
Statement: A Constraints and Policies Database should exist as part of the Panlab Repository
Justification: Panlab Office shall have the capability to apply rules (policies) regarding access to different systems. The Policies Database shall include rules regarding the access rights of Panlab Customers and Partners to Panlab resources and databases (e.g. Tests Results Repository). Access to a system (full/limited/restricted) shall be determined in accordance with the customer’s/partner’s profile (User/Customer Database). Confidentiality of the data stored in the repository should be taken into careful consideration. Customers should be classified according to their confidentiality and privacy classification – for instance Major Players vs. SME’s.

Requirement ##D3.2-Architecture/System Analysis##M.1250
Statement: Panlab Platform shall have the possibility to provide Testing Tools in the form of a resource.

Justification: Reusable Testing Tools reduce customer expenses that would arise from potential investments in such tools and enhance testing efficiency (offer several testing capabilities). Testing Tools shall be provided by Panlab Partners and Panlab Customers shall have the opportunity to utilize them on demand. From the PTM’s point of view, Testing Tools should be considered as regular testbed resources. The Resource Reservation System shall uniquely identify Testing Tools as resources registered with the federation and keep track of their availability.

Requirement ##D3.2-Architecture/System Analysis##M.1300
Statement: Panlab federation shall provide support for the set-up of a Testing Tool in the form of a service offered to Panlab customers.

Justification: For cases in which cooperation with Panlab testbed experts is required for access to a Testing Tool and for assistance regarding the implementation of a specific testing configuration, the federation shall provide specialized personnel from the respective Panlab partner testbeds in the form of a resource. A customer shall be able to request and ‘book’ assistance, along with the Tool, through the Teagle Portal. Panlab partner contact persons could be individually stored in a Reservation System in order to allow for tracking of their availability and for a calendar based reservation.

Requirement ##D3.2-Architecture/System Analysis##M.1350
Statement: Teagle Portal should advertise the tools and capabilities of the Panlab platform to Customers.

Justification: Teagle Portal should provide the means to promote the facilities offered by the Platform to Customers (available Testing Tools & Test Suits, testing capabilities, ability to host a certification process, etc.).

Requirement ##D3.2-Architecture/System Analysis##M.1400
Statement: The Panlab platform shall guarantee confidentiality of the results regarding the compliance of a system to specific certification requirements.

Justification: To enhance privacy and security, the Panlab office shall ensure limited access to test results related to the Certification Process. Access to sensitive data shall be controlled by means of rules defined within the Constraints and Policies Database.

2.1.3 From FIRE Week

This section enumerates the requirements for future testbeds as outlined in the discussions that took place during FIRE Week at Les Cordeliers, UPMC Paris Universitas in September 2008[REQ_4]. The focus for discussion was Testbeds for Future Internet. Relevant participants from the area were invited to present before a brainstorming workshop on the topic ‘How to federate’.

For the purpose of this review, focus is directed at a subset of the content which is directly relevant to the work of this deliverable. While other material is relevant to the project in general, this subset
directly impacts and comments on the area of Resource Description and Specification. The presentations that will be focussed on are those by Max Otts [REQ_5], Miguel Ponce de Leon [REQ_6] and Tanja Zseby [REQ_7]. There are also some notes from a panel session chaired by Milon Gupta of Eurescom and Martin Potts of Martel.

Experiments, defined by researchers, are seen to provide the specification for the testbed required. These experiments have a number of requirements. They must be Controllable, Comparable and Repeatable. For these to be satisfied, they must be described unambiguously using abstractions, models and relationships. Ideally, experiments are supported by a controlled environment with reference data and observation tools to record and verify accuracy.

If the experiment is suitably defined, this specifies the experimental requirements for the testbed. However, the testbed must also provide control, descriptions and monitoring. Control permits experimental variants to be adjusted suitably to expose edge cases. Descriptions enumerate the resources and people associated with a testbed for a given experiment. Monitoring provides information on performance, events, accuracy and overall satisfaction for a given experimental testbed.

2.1.4 From Future Internet Assembly

FIREworks was tasked to call for use scenarios for the FIRE facilities for the FIA event in Madrid in December 2008.

PII D1.1 was presented in Madrid [REQ_1]. It was described in this context that Panlab customers (targeted users) could be one of the following users:
- Enterprises in the ICT domain wishing for specific reasons to run tests locally
- Local community or evaluators interested to run/use the services for evaluation
- Panlab federation partner offering the virtualized services/components.

Other scenarios presented in this seminar came from Fraunhofer FOKUS, BT, SEA, Self-NET, SO4ALL, C-CAST and MANA.

NESSI presented their Requirements Report. This Deliverable D10.1 describes scenarios collected and a list of requirements derived and prioritized from them in order to provide the necessary guidance to the work performed by WP7. This was in respect to the design of the reference architecture of the Open Service Framework. This work has thus been in close cooperation with WP7 in order to subsequently ease the mapping of requirements collected to layered (functional) architectural view promoted.
3 Alignment with Architecture

In this section, the initial gathered requirements from Section 2 are applied to the architecture from WP2 and the work carried out in T3.1 [REQ_3]. This is where abstract requirements and use cases start to map to the proposed architecture.

3.1 Summary of Underlying Architecture (in D2.1)

This section presents a summary of the underlying architecture as described in D2.1 [SUA_1]. The aim is to provide the reader with an understanding of the work this deliverable is building upon and how this work will be further advanced.

The architecture detailed in [SUA_1] (Figure 1) defines four main conceptual components:

- TEAGLE and support functions
- The Panlab Testbed Manager (PTM)
- The Interconnection Gateway (IGW)
- The Testbed Components

The first component listed, TEAGLE and support functions is not described in detail in D2.1 as this work has been allocated to its own work package (WP3), which this document is a part of. In D2.1 the TEAGLE was described for completeness of the infrastructure specification and also to outline the interfaces between TEAGLE and the other architectural components (primarily the PTM). In short, the TEAGLE provides a user interface for the PII testbed federation to the Customer. It allows the customer to view all available testbed resources and construct a VCT request that satisfies their unique testing requirements. TEAGLE will be described in depth in later sections of this document.

The PTM is a major component of each PII testbed. It essentially provides the link between the customer’s request coming from TEAGLE and the individual testbed components that realize this request. The PTM exposes a Web Service interface towards TEAGLE. This interface allows TEAGLE
to issue a provision or configuration request for the customer after interpreting their initial request based on resources available, SLA constraints, etc. The PTM provides the backend support to implement this request. It essentially translates the provision/configuration messages sent from TEAGLE into resource specific commands that the individual testbed components can understand and execute.

The PTM also provides other functionality besides its primary message translation role. This functionality relates to resource management. It can implement a resource discovery service that detects changes that occur to the testbed resources. For example, when a new testbed resource is added to a testbed or a resource becomes unavailable for any reason (i.e. maintenance upgrade, resource goes offline due to a local network failure, etc.) the PTM communicates these updates to TEAGLE, which in turn reflects the change towards the Customer. The PTM provides a monitoring service for all testbed resources connected to it. This facilitates the customer’s need for validation by ensuring the testbed behaves correctly and that it has been setup properly. Finally, the PTM facilitates the configuration of a testbed federation involving multiple testbeds. It achieves this by instructing the IGW to communicate provision and configuration commands sent from TEAGLE to the other involved testbeds through their own IGWs.

The IGW is another mandatory component that all testbeds must contain at least one. Its purpose is to connect testbeds and components inside these testbeds to establish the required testbed federation setup requested by the Customer. The IGWs will be instructed by the PTMs through SIP/IMS protocols to create/delete/configure connections between the involved testbeds and their components. VCTs will be used in PII to provide the support for fulfilling the customer’s requested testbed setup. The VCT solution will be delivered by establishing one VPN per customer to connect the required resources from the various testbeds. By using independent VPNs for each VCT request, security will be inherently enforced as the VPNs are completely independent from each other.

The final conceptual component described in the overall PII architecture is the testbed components that form the Resource Plane of each testbed. A testbed component is a single logical entity within the infrastructure offered by a testbed for the use of PII Customers. No specific technology limits are placed on these components by PII; however each testbed component must be individually controllable by the PTM. In addition to the primary functionality the testbed component provides in the Resource Plane, all testbed components must provide:

- Individual identification
- Addressability and be accessible from the PTM
- Controllability by accepting provisioning and configuration commands
- Optional monitoring capabilities to inform the PTM about its current state and convey test results back to the PTM.
4 State of the Art

4.1 Introduction

The state of the art section lists existing approaches and technologies that will be taken into account when designing and implementing the Panlab repository.

4.1.1 Repository

This section examines potential relationships between basic information required to be stored and database types which may support this.

4.1.2 Information Categories

Following the initial examination of the work carried out in Panlab I and the areas identified in both Use Cases (WP1) and procedures (WP4), the following information categories have been identified.

4.1.2.1 Resource Reservation

Once resources are discovered and registered with PII, the ability and system to reserve them for VCTs is required. As VCTs are temporal in nature, a calendaring system may be suitable here. The resource reservation system must be able to uniquely identify a resource, provide systems with the ability to reserve that resource for a set period of time and not allow other reservations of that same resource for the same time.

4.1.2.2 Orchestration Persistence

In the context of the PII project, orchestration definitions are needed to coordinate and automate the fulfillment of a user requesting resources available in one or multiple testbeds. Reuse of orchestration implies having a place to store and retrieve them. We expect a federation of testbeds to offer such a repository.

Typical information expected in this repository, apart from the storage of the script itself is:

- Date of creation of the script
- User or company that created
- Access rights for other users

In addition, it may be useful to attach the initial user request (if existing) that was responsible for the script generation. This makes it possible to use the repository as a cache for the orchestration (no need to regenerate a script if the request has already being interpreted).

Another aspect to be considered is the fact that there can be two levels of script: a specification script - purely functional - with no dependencies to a given technology like SOAP for communication or Java for programming and an executable script derived from the specification. In this case we have the question whether it is useful to store the two versions.

4.1.2.3 PII Logging

PII logging is concerned with capturing relevant events that occur during the provisioning/configuration stages of implementing the customer’s VCT request and also while the VCT is in operation. Logging will be necessary to verify each stage of setting up a VCT before progressing to the next stage. Other logging situations include; operational verification of the customer’s requested VCT before handover to the customer and automatic event logging generation when a link to a resource fails or the resource itself fails. The level of event capturing will have to be determined in PII so as to acquire enough information when a relevant event has occurred in the system but not to the extent that the system is burdened with processing unnecessary informational events. Different technologies are discussed in the next few sections that could be used to support this logging functionality.
Log4j is an open source project of the Apache Software Foundation and is widely used in large projects such as JBoss and Hibernate. Log4j is a java based logging framework for logging application debugging messages [PL_1]. It has been designed not to slow the application’s performance despite the addition of extra code into the application. Log4j enables logging to be turned on or off at runtime without having to modify the application binary. The log4j environment is fully configurable programmatically. However, it is far more flexible to configure log4j using configuration files. These configuration files can be written in XML or in Java properties format.

There are three main components to log4j which are defined in the configuration file: loggers, appenders and layouts. Together these components can be used to generate log messages according to the message type and level, and can also be used to control at runtime how these messages are formatted and where they are reported to. The main advantage to using a logging API rather than simply inserting print statements into the application’s code is the ability to enable or disable particular log statements. The loggers (logical log file names) in log4j perform this capability by allowing a logging level to be set (OFF, FATAL, ERROR, WARN, INFO, DEBUG, ALL). The logs output destination is determined by the log4j appender. Multiple appenders can be attached to a logger so the log message can be outputted to several locations. There are numerous appenders available in log4j so the logs can be easily outputted to locations such as the console, files, remote UNIX Syslog daemons, etc.

The final piece of log4j’s framework allows the user to customize the format of the log messages. This is achieved by attaching a layout to the appender specifying the desired formatting details. Many log viewing tools are readily available for log4j. Chainsaw [PL_2] is another open source Apache project that is a Java based tool specifically developed for viewing and analyzing log4j logs. Another such tool is called log2web [PL_3], which is open source and web based but has fewer features than Chainsaw.

In comparing log4j to similar logging APIs; Java Sun’s java.util.logging (JUL) [PL_10] package was considered. JUL is architecturally very similar to log4j using three main components that really only differ by name. For example, loggers in log4j are called ‘logRecords’ in JUL and similarly appenders in log4j are called ‘handlers’ and layouts are ‘formatters’. However, log4j has many advantages over JUL. Firstly, log4j emerged in 1998 and is considered a more mature framework, [PL_4]. As it is an open source project log4j has been developing and improving at a faster pace than JUL. JUL contains four concrete handler implementations which are adequate for basic logging, allowing writing to a file, a buffer, a socket and a console. Log4j has over a dozen appenders that cover writing logs to most destinations, such as to a syslog receiver, to email, etc. Log4j also has many more formatters available than JUL, which contains just two; XMLLayout and SimpleLayout. Log4j includes corresponding formatters for JUL’s two but it also contains three more; TTCCLayout, HTMLLayout and the PatternLayout. The PatternLayout is the most flexible and commonly used as it can be configured using string conversion patterns, similar to the syntax used in C’s printf function. JUL has similar functionality to log4j but as a whole, log4j contains many more additional features and functionality [PL_5].

Syslog is another message forwarding protocol that allows a machine/device to send event notification messages across an IP network to a message collector. The term ‘syslog’ can be used to describe the protocol that is used to transfer the messages; the application to send or receive the messages as well as the log messages themselves. Syslog was originally developed for UNIX systems and has since become the standard logging solution for both UNIX and Linux systems with many other implementations available for most other operating systems. In 2001, the IETF formally documented the Syslog protocol in an attempt to standardize the protocol and address its security issues. This work is documented in RFC 3164 [PL_6]. Syslog has since been standardized by the Syslog working group of the IETF [PL_7].

Syslog is based on a client/server architecture. The syslog enabled device or application sends a small text based message containing log information to the syslog receiver, which can also be called the syslog server, syslog daemon or syslogd. The log messages are less than 1KB and can be encrypted using SSL/TLS although this is not part of the syslog protocol. Syslog messages were originally sent via UDP but reliable delivery of syslog over TCP can also be achieved as outlined in RFC 3195.
Syslog is supported by a wide variety of devices and receivers thus making it easy to integrate logging data from these different systems into a central repository.

### 4.1.2.4 Customer and User Information

This repository shall be able to maintain customer profile and login information. Users should be able to log-in and administer their account and profile data. Authentication and authorization processes shall be supported by the repository on behalf of other functional TEAGLE blocks. For example resource configuration/access rights and charging models might depend on the customer identity. The repository needs to support this type of policy restriction and policy administration processes and optimally offer authentication and authorization as a service to other TEAGLE components such as the VCT tool.

Also, some Customers need to be sure that their information will be collected, stored and available only for their own purposes, if they require so. This requirement supports the need and importance of customer classification and authentication processes and it should also be taken into careful consideration in WP4 (Quality Assurance).

### 4.1.3 Databases

Teagle architecture requires databases for storing various types of information such as user profile, testbed configurations, testbed status, testing results, composed models of virtual customer testbeds (VCT), etc.

The main requirements to be used in database selection are:

- Fast retrieval of data, for example, during searching and VCT composition
- Fast synchronisation of data over a network between the testbed local cache and Teagle repositories
- Fast query during searching
- Relative good performance during updating/storing data
- Reliability (an essential part of Teagle) – high availability required
- Scalability
- Commercial DB or open source

#### 4.1.3.1 Relational Databases

Relational databases have standardized the way that data is stored and processed. This is due to the fact that they allow large extendable databases to be created and manipulated quite easily. Relational databases are managed by Relational Database Management Systems (RDBMS). There are numerous RDBMS on offer with both proprietary and open source licences. Some of the most popular proprietary systems are offered by Oracle, Microsoft (Microsoft SQL Server) and IBM (DB2). There are also many open source systems available with the most commonly used being MySQL, PostgreSQL and Firebird.

RDBMS offer all the characteristics that Teagle requires from a storage repository, including fast querying of data, replication and partitioning for reliability and performance. RDBMS are also generally highly scalable through an easily extendable architecture. All of the above mentioned RDBMS naturally have varying degrees of support for the requirements of Teagle and choosing one over another can be quite difficult unless strict requirement specifications are available. Until these are available, working from the general high level requirements, the databases listed should all be able to support these requirements in some manner. Some of the main requirements that need to be specified are the hosting platform, the method of connectivity (JDBC, ODBC, etc.), replication support, backup requirements, data type support, etc.
Column-oriented RDBMS are an alternative to the traditional RDBMS described above, which are row-oriented implementations. Column-oriented RDBMS are deemed to be more I/O efficient for read-only queries, as they only read the columns they need, whereas row-oriented systems read all the columns. They also offer increased compression compared to row-oriented implementations due to the fact that only a single data type needs to be compressed for each column. These two attributes make column-oriented RDBMS ideal for data warehousing systems, whereas, row-oriented RDBMS tend to suit transaction processing systems where frequent reading and writing of data occurs. Again, many commercial and open source products are available, with Sybase[RDB_7] and Vertica[RDB_8], among the leading commercial products and C-Store[RDB_9] and LucidDB[RDB_10] being examples of open source products.

4.1.3.2 XML Native Databases

In addition to Native XML databases (NXD) there are XML-enabled databases. The real NXD databases have been designed to process XML-based information and store it in XML format which may result in a performance improvement over XML-enabled relational databases. XML databases use different types of query methods such as XPath, XQuery and XML:DB API. XPath can be used to query documents or collections of documents. XQuery extends XPath providing statements such as For, Let, Where, Order and Return. XML:DB API (or XAPI) defines a common access for different XML data stores in a similar way that Java Database Connectivity provides access to different relational databases.

IBM SolidDB is an example of an XML enabled database. It is a distributed relational database with an XML processing plugin. SolidDB can be run on a Linux environment and it supports automated synchronisation of database installations over the network. This could be used to update the Teagle repositories and the caches of different testbeds using the native format of the database.

4.2 Information and Data Models

An information model symbolizes the concepts, relationships, constraints, rules, etc. for a domain of interest. The main purpose of an information model is to model managed objects at a conceptual level, independent of any specific implementations or protocols used to transport the data [INF_1], i.e. information models are platform independent. Data models on the other hand map the data described in an information model to a specific technology or platform.

A number of information and data models were evaluated for use in the project. These included the Common Information Model (CIM), the Shared Information Data Model (SID) and DEN-ng (Directory Enabled Networks-new generation). The CIM [INF_2], which is standardized by the DMTF was ruled out from an early stage of the project, as it was considered to be too specific a model. The CIM is concerned with modelling elements of an IT environment, e.g. operating systems, networks, storage, etc. Whereas, in PII it is foreseen that many more concepts will need to be model in order to develop a federated testbed system. For this reason the other two models, the SID and DEN-ng were considered. These models are specifically designed to address issues within the telecommunications industry and hence focus more on the concepts and issues most likely to arise in PII. In developing a federated testbed, the use of information and data models will facilitate managing the large amount of data involved within the testbed federation.

4.2.1 Shared Information Data Model (SID)

The Shared Information Data Model (SID) [SID_1] is part of the TeleManagement Forum (TM Forum) solution frameworks. These frameworks called the New Generation Operations Software and Systems (NGOSS) framework [SID_2], allow communication service providers to enhance their business support processes and systems. NGOSS can help providers to achieve lower operational costs and reduce the time to market for new or updated services and products. Operators achieve this using the NGOSS toolkit, which provides specifications and guidelines that enable service providers to better understand and redevelop their business processes, business support systems and easily integrate
information flows. Typically, an operator’s software base is extremely fragmented with applications coming from various suppliers, equipment vendors and internal development. These numerous data sources that are modelling the same processes can result in information duplication and create a data sharing challenge. The NGOSS framework and the NGOSS SID in particular provide the ability to integrate this fragmented data into a clear framework.

The SID provides the Information Framework component for NGOSS, which is a key component, as it provides an information and data reference model and a common language from a business and system perspective. Another important component of NGOSS is the Business Process Framework called eTOM (enhanced Telecom Operations Map) [SID_3]. eTOM describes the business processes employed in the telecommunications industry. The SID and eTOM model the same subject matter but from different view points. When eTOM is used in conjunction with the SID, a link can be created between the business and Information Technology (IT) groups that allow both of these groups to understand the business processes in a level of detail that suits their needs. This model must be flexible enough to allow people and systems with different information needs use it. It should also be robust enough so that it can be used to translate the information needs of different users, using different terminology and having different understandings of the same concepts. Hence, the business group understand the definitions of the business processes and these definitions are still detailed enough to be used for software development. In summary, the eTOM provides a view focused on the business processes, whereas the SID provides an entity view of the system, i.e. the ‘things’ that the business processes act on, their characteristics and relationships [SID_4].

The key benefits of using the SID include:

- Reduced time to market
- Reduced cost of integration
- Facilitation of new technologies
- Reduced management time and cost

The SID is a federation of a number of different industry models, with major inputs coming from the DEN-ng model and ITU-T recommendations. The SID is not simply an amalgamation of these models but it has been carefully constructed to produce a model consistent with the eTOM.

The SID business view addresses the need for shared information/data definitions and models. The definitions in the business view focus on business entity definitions and their associated attribute definitions. A business entity is a thing of interest to the business such as a customer, service, product, etc., while its attributes are facts that further describe the entity. The SID organises these business entities into a layered model with eight domains at the top level. Each of these eight information domains broadly aligns with the eTOM business process framework [SID_5]. Within each domain there is a high degree of cohesion between the business entities and loose coupling between the different domains. This representation allows for the business problem to be broken into manageable pieces and allows resources to focus on a particular area of interest, e.g. for a business process that you are automating it’s possible to identify the shared information and data that is needed to support that process.

The eight domains of the SID’s top level view are further broken down to contain several Aggregate Business Entities (ABE’s) [SID_5]. As the SID evolves these current ABE’s may be refined as they are more explicitly defined. Also, some of the domains are a work in progress and are due to be completed in future releases. The SID business entities and models are packaged in a series of documents and modelled in Rational Rose UML-based models. Detailed information can be found on each ABE by looking up the relevant addendum for each domain.

The SID uses standard software patterns to improve the quality of the model and ensure that it is reusable and extensible for future requirements. These patterns capture common relationships and occurrences of physical connections and structures. The four main patterns used in the SID are:

- The Specification Pattern
- The Composite Pattern
• The Role Object Pattern
• The Abstract Superclass Pattern

The focus of this deliverable is to describe and store the individual testbed components from each testbed provider in a meaningful way so that these components can be discovered, catalogued, searched and configured/provisioned by an automated system. This task will amount to a large quantity of data that can quickly become a burden on the system if not controlled correctly. The SID model is a possible solution to handle this requirement as it provides a structured information framework that can be used to store clear definitions of entities, their associated attributes and the relationships between different entities. Also, the SID implicitly contains an innate common language that facilitates data sharing and interoperability. This means that the different descriptions and terminology used to describe the testbed components by the various testbed providers can be mapped and transformed to share a common vocabulary. This ensures a common understanding of the terms used and hence facilitates communication by allowing interfaces to be developed between the different components of the PII architecture. Different data models can be subsets of the SID common information model thus making them interoperable see figure 2. The SID is highly extensible so future testbed components can be easily added. Finally, as the SID model details common business processes, these could be used as the basis for implementing key components of the PII architecture, such as, billing customers, provisioning testbed components, etc.

![Information Model Diagram](image)

**Figure 2:** Inferring interoperability by using data models that are subsets of the information model

### 4.2.1.1 IPSphere

The IPSphere Framework which recently became part of the TMForum delivers a business layer for rapid service delivery, including advanced support for IP services [IPS_1]. The framework simplifies service management by defining specifications that help to automate the offering, purchasing and provisioning of service components between multiple stakeholders.

IPSphere uses service templates to describe the technical and commercial parameters of the service and the method in which elements must be selected to create it. These service templates are the backbone of the framework and are generally conformant to the SID model. Service templates define

- the relationships between services and resources
- how to control the processing associated with services and elements
- the policy for commercial exchange of services
- the relationship between the stakeholders involved.

The FP7 Federica [IPS_2] project is using IPSphere service templates to provide users with slices of its network for experimental research. It may be possible for PII and Federica to collaborate by linking
resources through a mapping of their IPSphere service templates to the information model that PII will employ. Further investigation will have to be carried out at a later stage of the project.

4.2.2 DEN-ng

DEN-ng is an object-oriented information model that describes the business, system, implementation, and deployment aspects of managed entities and their relationships. It was intended to enable the construction of autonomic networks and was originally designed to meet the needs of the TMF NGOSS architecture [DEN_1]. DEN-ng like the SID model is defined using UML and is also a federation of models. In fact, the SID is based on an early version of DEN-ng (v3.5), whereas the current version of DEN-ng is v6.6.4. The fact that the SID is based on DEN-ng means that the models have many similarities and originally, developments in DEN-ng were submitted to the SID team to be integrated into the SID model. However, a change in the direction of both models has developed and they have diverged to some extent. DEN-ng is currently being standardised in the ACF [DEN_3].

DEN-ng focuses more on knowledge management than simply modelling entities. It uses classification theory to ensure that information about class instances is separated from how classes and other objects are classified. While the SID also employs a degree of classification theory it is less strictly enforced as in DEN-ng. The first and fundamental classification of DEN-ng separates ManagedEntities from Metadata and Values as shown in figure 3.

The DEN-ng policy framework is far more flexible compared to the SID’s policy framework. DEN-ng has concentrated on developing its policy model to integrate with appropriate parts of other DEN-ng domain models. This means there is an inherent relationship between the policy model and the other DEN-ng domain models allowing DEN-ng to control the management of policy from a customer right through to the configuration of network devices [DEN_1][DEN_2].

DEN-ng from its beginning included support for roles and patterns, which the SID consequently inherited. However, DEN-ng has developed this further using many more patterns than the SID and applies these patterns in a stricter manner. This provides inherent extensibility in the DEN-ng model and enables formal transformations through detailed semantic definitions. DEN-ng is the only model whose design philosophy was to support orchestration of services through their entire lifecycle; it does

![Figure 3: First and Fundamental Classification in DEN-ng](image-url)
this by using a set of finite state machines to model the full lifecycle of a ManagedEntity. The ManagedEntity’s behaviour can be orchestrated by using context-aware policies to determine the next state the ManagedEntity is allowed transition to. The SID is lacking in this respect as it doesn’t implement state machines. In summary, the SID is strongly focussed on representing business concepts whereas DEN-ng is more focussed on network management concepts.

4.3 Languages & Tools

4.3.1 Languages

A variety of languages were considered for specifying the testbed service descriptions that will be modelled using the information model. Traditional modelling languages such as UML through to semantically annotated languages (e.g. SAWSDL, OWL, etc.) were reviewed. A comprehensive overview of their features and weaknesses is presented and assessed. Other languages such as WSDL were also evaluated for the interface between Teagle and the PTM.

4.3.1.1 UML/Class Diagrams

The Unified Modelling Language (UML) is a visual language for specifying, constructing, and documenting the artefacts of systems [UML_1]. As a general purpose modelling language, it can be applied to all application domains and implementation platforms.

UML provides an abstract syntax for modelling concepts, their attributes and their relationships. There is a technology independent explanation of the semantics of each UML modelling concept.

There is also a set of human-readable notation elements and rules for combining them into different diagram types based on different aspects of the model. These diagram types, Structural and Behavioural, allow this. Structural Diagrams include Class, Component, Composite, Deployment Object and Packaging Diagrams. Behavioural Diagrams include Activities, Interactions, State Machines and Use Cases. There are a number of Interaction diagrams including sequence diagrams, communication diagrams, interaction overview diagrams and timing diagrams. Finally, there are a set of rules for creating tools that can be compliant with the UML2 specification.

The XML Meta Interchange (XMI) is a mapping from MOF based metamodels to XML and a Schema for validation [UML_2]. This allows UML models to be serialized as XML using XMI. It does not, however, allow diagrams to be interchanged. For this, compliance with the Diagram Interchange [UML_3] specification allows uniform interchange of diagrams.

4.3.1.2 WSDL

The Web Service Description Language (WSDL) is a protocol-, platform- and programming language independent description language for the exchange of messages between Web Services on the basis of XML. WSDL is a metalanguage that enables description of a Web Service in terms of functions, data, data-types and data exchange protocols. Predominantly, WSDL describes the exposed operations of a given Web Service, the operation’s parameters and return values.

There are six main WSDL elements in order to describe a Web Service, three attributing to the abstract definitions and three attributing to the concrete definitions. These include:

- Abstract definitions:
  - types – defining the data-types that are used to exchange the messages
  - message – an abstract definition of the exchanged data, comprised of several logical elements of which each is linked to the definition of a data-type system
  - portType – four types of message exchange modes;
    - one-way – the service receives an input-message from a client
    - request-response – the service receives an input-message and replies
    - solicit-response – the service sends a message and expects an answer
- notification – the service sends an output-message
- Concrete definitions:
  - binding – defines a specific protocol and data-format for the operations and messages
  - port – specifies an address (usually a URI) to use as a communication interface
  - service – subsumes several related ports

The World Wide Web Consortium (W3C) released the first WSDL version 1.1 [WSDL_1] specifications in 2001. WSDL version 2.0 [WSDL_2] was released in 2007. It enhances WSDL 1.1 in several ways. It adds additional semantics to the description language by allowing for accepting binding to all HTTP request methods. This greatly simplifies the actual implementation. However, WSDL 1.1 is still commonly used and software development kits often offer tools only compatible with WSDL 1.1.

In conclusion, the WSDL standard provides a syntactic description of the elements of a Web Service, describing how a client can utilize a given service. For enhanced discovery, automated service orchestration and additional descriptions of a service, e.g. in terms of costs, QoS or effects of an operation, WSDLs are not sufficient. In order to describe additional aspects of a given service, Web Service Semantics (WSDL-S), OWL-S [WSDL_3] or WSMO [WSDL_4] are more appropriate. However, ontology based service description mechanisms like OWL-S and WSMO imply much greater complexity, which usually comes with increased processing time.

4.3.1.3 RDF

Resource Description Framework (RDF) is the most basic language in the Semantic Web language stack [RDF_1], it’s aim is to provide a common framework that allows data to be shared and reused across applications, enterprises and community boundaries.

The most important feature of RDF is simplicity, so that it provides a very well understood metadata structure for information modelling based on three assumptions:

- resource: anything that can have an URI such as a web site or a web service or element
- property: a property of the thing that the statement describes
- statement: a link between a resource and its property.

Every single RDF statement can be described with a graphical representation or with XML-based serialization syntax, and is composed of three terms:

- subject: an URI that indicates a resource
- predicate: the property of the subject
- object: the value of the property.

RDF Schema (RDFS) extends the language with a new vocabulary and defines a semantic extension of the basic language that provides the necessary instruments to describe groups of related resources and relationship between resources. It adds some basic concepts such as properties, classes, subclasses, data types, constraints, containers and collections. The RDF Schema is also important for the understanding of every semantic language descending from RDF, such as DAML and OWL, because it introduces many capabilities used for describing ontologies. The main advantages of RDF Schema (and OWL) in contrast to other representation formats are their Web-based representation of domain-specific knowledge as well as support for deductive querying and reasoning.

Moreover, RDF databases are also being put to use in more practical ways, as a replacement for or supplement to relational databases in more general purpose applications, particularly those that need to model relatively abstract information or drive knowledge management systems.

RDFS can be profitably used as an alternative to XML Schemas. The extensive reasons for the RDF choice instead of XML are described in [RDF_2] and summarized as follows:

- it allows easier exchange of information between independent domains,
- it is easily extendible,
• it allows integration of independent data models developed in heterogeneous fields.

One disadvantage of RDF could be its verbosity but it is worth mentioning the alternative approach. This requires development of XML-based custom solutions and could lead to inefficiency both for compatibility reasons, as well as being unable to leverage the existing RDF tools. In fact, several open source and publicly available tools and APIs are available ([RDF_2] [RDF_3]), thus making the use of this syntax straightforward.

An additional advantage of using RDF/S is to leverage its semantic richness that is needed for supporting dynamic service discovery and composition in autonomic systems. Many examples can be found in literature; in the following a brief overview of the relevant applications of RDF/S is presented.

RDF/S can be used to structure component specifications in a more coherent way (compared to relying solely on UML models) with the ability to add semantic information in a superimposed layer. A short note [RDF_4] made available by W3C for discussion only, describes a relationship between RDFS and UML models.

In [RDF_5] is described an example on how RDF/S can be employed for integrating distributed, heterogeneous business component specifications. In a first alternative, a superimposed information space is spanned that semantically describes physical information resources but leaves them unchanged, thus achieving integration of heterogeneous specification resources that would remain unconnected otherwise. In a second alternative, the relevant information is extracted from heterogeneous specification resources and represented as RDF graphs derived from an uniform standardized component specification in UML. In this case, a UML profile and a transformation from the XMI serialization into RDF needs to be defined.

Transformations have been described in [RDF_6], which discuss a technology to support the use of UML for representing ontologies and domain knowledge in the Semantic Web. Two mappings have been defined and implemented using XSLT to produce Java classes and an RDF schema from ontology represented as a UML class diagram and encoded using XMI.

Sometimes UML has to be maintained as the primary notation. A solution to this requirement is found in [RDF_7], which presents a technique for augmenting UML with loosely coupled background knowledge and with reasoning support on this knowledge. The background knowledge is attached via the Tagged Value construct of the UML which is mapped to RDF statements. The resulting triples can be used to reason about and query the model contents. This approach is compatible with all versions of the UML because it does not require any change in existing models, profiles or tools.

RDF annotations can also be directly included into XML data models in order to better organize, interrelate and classify this knowledge, thereby increasing the aggregate value of the stored data. The simplest way to add features to XML-based applications for semantics or just for greater flexibility in information modelling, is by using Extensible Stylesheet Language for Transforms (XSLT) [RDF_8] for converting from XML to RDF syntax. Some basic examples can be found in [RDF_9].

In regard to the Web Services Description Language (WSDL) which provides a model and an XML format for describing Web services. W3C recently produced a specification [RDF_10] for a representation of that model in RDF and in the Web Ontology Language (OWL). It describes a mapping procedure for transforming particular WSDL descriptions into their RDF form, so that WSDL 2.0 documents can be transformed into RDF and merged with other Semantic Web data.

Previous attempts based on simple XSLT transformations are given in [RDF_11] and [RDF_12].

In [RDF_13], which extensively presents semantic based dynamic service composition architecture, RDF is used in a slightly different manner. That is, a WSDL (binding) file imports a WSDL file describing the functional aspects of the (pre-existing) service plus a newly-created RDF file which describes the semantics and the logical aspects of the service. This arrangement allows the existing (already deployed) Web service to migrate to semantic based service composition architecture without the need for reimplementation.
Another solution, proposed in [RDF_14], is to publish the content of a relational database using a proxy/server thus allowing data access according to RDF formalism. This is actually a mapping tool (D2R Server) which enables RDF and HTML browsers to navigate the content of the database, and enables applications to query the database using the SPARQL query language. This approach was used in [RDF_15] with the objective of creating an information federated semantic layer and a mechanism to propagate, share and validate legacy data among network management systems.

4.3.1.4 SoAML

The Service oriented architecture Modelling Language (SoaML) extends UML to support service modelling and design. It can be used for modelling both basic services, focusing on the specification of a single service without regard for its context or dependencies, and services architectures showing how multiple participants cooperate, provide and use basic services to achieve an overall business goal or process. The SoaML [SML_1] describes a UML profile and metamodel for the design of services within a Service Oriented Architecture (SOA). The profile and metamodel approach the issue of service design taking into consideration the perspectives of both the service consumer and service provider. Additionally, the perspective of system design that describes the interaction between consumers and providers in order to achieve the overall objectives is considered. Therefore, SoaML contains modelling constructs for things like contracts, services, service providers and consumers.

UML extensions of SoaML support a range of new modelling capabilities, mainly [SML_1]

- Identifying services and dependencies between them.
- Specifying services - their functional capabilities, the customer capabilities, the protocols/rules for using them and the service information exchange between customers and providers.
- Defining service consumers and providers, their interconnections, as well as the services they will consume/provide.
- Defining policies for using and providing services.
- Defining service and service usage requirements and associating them to related OMG (Object Management Group) metamodels, i.e. the BMM course_of_action, BPDM Process, etc.
- Defining classification schemes that support architectural, organizational and physical partitioning schemes and constraints.

The aforementioned modelling requirements for service oriented architectures are accommodated by the SoaML profile and metamodel, which are built upon a set of key concepts related to basic services and services architectures.

Specifically, regarding basic services, the SoaML considers the concept of the actual Service, where a Service is the functionality offered by a single or set of entities to others, provided that a number of terms and conditions are met, using well defined interfaces. The notion of entities is encapsulated by Participants, that can be people, organizations, technology systems or components that provide or use services. A Service Port is the point of interaction on a Participant where a service is actually produced or consumed. Participants offer capabilities through services on Ports. The type of the Service Port specifies everything needed to interact with that service. That type may be either a UML interface (for very simple services) or a Service Interface. The Service Interface describes the service functionality and usage; it also provides information necessary for service providers to implement the service. A Participant expresses its needs by making a Service Request to some other participant. The Service Request defines what services a participant needs or consumes.

The services architecture on the other hand, organizes a set of services and defines the collaboration of the Participants for a community or organization. SoaML identifies components of services architecture at two levels of granularity:

- community services architecture, which is modelled as a collaboration,
- Participant’s services architecture, which specifies the architecture for a particular Participant. Within a Participant, services architecture illustrates how sub participants and external collaborators work together.
Key concepts for the services architecture are the *Service Contract* and the *Service Capability*. A Service Contract is the actual specification of a service, including terms, conditions, interfaces and choreography between interacting participants. Service Capabilities allow for focusing on just the services, without concern for how a service is implemented and offered by a Participant. Thus, the relationship between services can be identified prior to their allocation to a particular Participant.

The scope of Service Capabilities, Service Contracts and Service Architectures is to bridge the gap between business concerns and the actual SOA solutions. They provide the means to map the business requirements of the contracts to services and service Participants that fulfil the contracts, isolating the business from the IT concerns and technologies that support SOA, like Web Services and CORBA.

The SoaML profile provides a metamodel which can be incorporated into standard UML modelling tools. Therefore, SoaML can be used with either existing UML tools or tools that offer enhanced, SOA specific, support. Additionally, the UML profile enables existing UML2 tools to effectively develop, transform and exchange services metamodels in a standard way, while providing the foundation for new tools to extend UML2 in order to support service modelling.

### 4.3.1.5 Semantic Annotations

The study of meaning has emerged as an important factor in communications, as establishing a common terminology for any particular domain promotes the effective management of distributed knowledge available for that domain. As a result, establishing semantic interoperability allows for flexible and efficient information reuse.

Semantic annotation is the enabler of semantic interoperability, due to the fact that it adds information related to the meaning definitions of the concepts used in a domain. The meaning definitions of concepts are in most cases represented in ontologies, which provide a shared vocabulary to model the domain. Therefore, semantic annotation links domain entities to ontologies. Practically, semantic annotations are considered the information about what entities appear in a text and where they do, and they are usually in the form of metadata.

In the context of Panlab, semantic annotations can be used to facilitate the automation of Web Services tasks such as service discovery, execution and composition, as they provide the foundation for a common vocabulary and definition of rules for use by independently developed services.

#### 4.3.1.5.1 Semantic Annotations for UML

A system model expressed in UML can be extended using standard UML mechanisms to include semantics expressing concepts from a given domain. The result is an annotated model, where the annotations are specified by the designer in the UML model and attached to various model elements. Using semantic annotations in a UML model establishes semantic homogeneity, as UML model contents are associated to domain ontologies.

Therefore, to include semantic annotations in a UML model, the semantic references from UML elements to ontology concepts must be modelled. This requires the implementation of UML profiles with stereotypes that enable capturing this information in a UML model extension. Examples of the aforementioned approach are described in [SAU_1] for the creation and management of XML schema vocabularies, and in [SAU_2] for annotating UML models with non-functional properties for quantitative analysis techniques used for the verification and validation of temporal characteristics of real time embedded systems.

#### 4.3.1.5.2 SAWSDL

In the world of services there is the possibility that two services can have similar descriptions while meaning totally different things, or they can have very different descriptions yet similar meaning. Whereas, WSDL specifies a way to describe the abstract functionalities of a service and concretely how and where to invoke it. It does not include semantics in the description of Web services.

Resolving such ambiguities in Web services descriptions is an important step toward automating the discovery and composition of Web services.
The Semantic Annotations for WSDL and XML Schema (SAWSDL) W3C recommendation defines mechanisms for adding semantic annotations to WSDL components. SAWSDL does not specify a language for representing the semantic models, e.g. ontologies. Instead, it provides mechanisms by which concepts from the semantic models that are defined either within or outside the WSDL document can be referenced from within WSDL components as annotations. These semantics when expressed in formal languages can help disambiguate the description of Web services during automatic discovery and composition of the Web services. [SAW_1]

Based on the above design principles, SAWSDL defines new extensibility attributes to WSDL 2.0 elements to enable semantic annotation of WSDL components.

Bibliography [SAW_2] suggests the following four types of semantics:

- **Functional semantics.** A formal description of the service’s functionality is crucial for efficient service discovery and reuse
- **Data semantics.** A formal description of the data the service exchanges is crucial for interoperability
- **Nonfunctional semantics.** Formally defined service-level agreements and quality-of-service attributes are crucial to service providers’ efforts to differentiate themselves from their competitors
- **Execution semantics.** Formally modelling the Web service’s runtime behaviour and exceptions is crucial for ensuring that services execute correctly and for supporting runtime exceptions

Currently, SAWSDL has direct support for functional and data semantics but service providers can incorporate nonfunctional and execution semantics with the WS-Policy framework (www.w3.org/2002/ws/policy/).

A critical problem for Panlab federation service is finding testbed services that they can reuse across entire test processes. A functional description of the service, along with a discovery mechanism that can leverage the description could increase the reuse in testing services.

SAWSDL’s attributes can link operations to functional descriptions in domain models; for example, the testbed service provider can annotate all operations that implement a specific X Interface with an identifier representing the term. Using a discovery mechanism, users can locate all testbed services in the Panlab federation that implement the X Interface and reuse them as required.

**4.3.1.5.3 OWL**

OWL stands for Web Ontology Language and it is defined by the W3C Web Ontology Working Group [OWL_1]. It is aimed to be the standardized and commonly accepted ontology language of the Semantic Web.

Ontology languages allow users to write explicit, formal conceptualizations of domain models. The main requirements of an ontology language [OWL_1] are:

- well-defined syntax; this is a necessary condition for machine processing of information
- formal semantics; the meaning of knowledge precisely. To allow people to reason about the knowledge such as, class membership, equivalence of classes, consistency and classification.
- efficient reasoning support; allows to check the consistency of the ontology and the knowledge to check for unintended relationships between classes and automatically classify instances in classes.
- sufficient expressive power; the language can be supported by efficient reasoners while being sufficiently capable to express large classes of ontologies and knowledge
- convenience of expression

OWL would be an extension of RDF Schema, in the sense that OWL would use the RDF meaning of classes and properties and would add language primitives to support the richer expressiveness
required. Unfortunately, simply extending RDF schema would work against obtaining expressive power and efficient reasoning.

Efficient reasoning support and convenience of expression for a language as powerful as a combination of RDF Schema with a full logic seem unobtainable. Thus, OWL is defined as three different sublanguages each geared toward fulfilling different aspects of this full set of requirements:

- **OWL Full.** The entire language is called OWL Full and uses the OWL primitives. It is fully upward-compatible with RDF both syntactically and semantically. The disadvantage of OWL Full is that the language has become as powerful as to be un-decidable, dashing any hope of efficient reasoning support.
- **OWL DL (Description Logic).** It is a sublanguage of OWL Full that restricts how the constructors from OWL and RDF may be used thus regaining computational efficiency. The advantage is that it permits efficient reasoning support but the disadvantage is that it loses full compatibility with RDF.
- **OWL Lite.** This is a further restriction that limits OWL DL to a subset of language constructors. The advantage is that it is both easier for users to grasp and for tool builders to implement. The disadvantage of course is restricted expressivity.

OWL builds on RDF and RDFS and uses RDF’s XML-based syntax because RDF does not provide a very readable syntax. Other syntactic forms for OWL have also been defined:

- An XML syntax that does not follow RDF conventions and is thus more easily read by human users
- An abstract syntax, used in the language specification document that is much more compact
- A graphic syntax based on the conventions of UML

OWL documents are usually called “OWL ontologies” and are RDF documents. An OWL ontology may start with a collection of assertions and may import ontologies whose content is assumed to be part of the current ontology. It might also define class elements, property elements, property restrictions, data types, etc.

OWL could be used as an OWL representation of the information model. Such an example is given for DEN-ng in [OWL-3].

### 4.3.2 Tools

#### 4.3.2.1 Introduction

This section outlines the candidate tools identified at the current stage of analysis and design.

#### 4.3.2.2 StarUML

StarUML is an open source implementation of the UML graphical notation, available since 2005 from sourceforge at http://staruml.sourceforge.net/. It implements almost all the popular diagrams defined by UML 1.4: class diagrams, use-case diagrams, activity and state machine diagrams and sequence diagrams. The tool is known for being fast, stable and of good quality in terms of documentation and ease of use.

One of the major strengths of the tool is its openness for extensibility: it provides a complete customization mechanism that allows users to define their own set of diagrams for domain-specific modelling as well as to extend the pre-existing UML concepts (through the standard UML profile facility). The tool offers COM APIs for exploiting the models from external scripts. It is thus possible to write a specific serialization of a given model using JavaScript, Visual Basic, Python or any other script supporting the COM technology.

Among the week points of the tool are:
• The tool does not support the enhancements and new diagrams introduced in UML2 standard. In particular the UML component diagrams are not supported.
• No evolution of the tool since 2005. The community remains active in the assistance forum and new modules are contributed from time to time. However it seems there will be no upgrade of the core functionality of the tool.
• There is no explicit connection with Eclipse and Eclipse/EMF tools. Export facilities need to be written explicitly to that end.

To summarize, the tool appears as an excellent alternative to commercial tools for exploiting the basic functionalities of UML through graphical editing and/or through scripting.

4.3.2.3 Eclipse UML 2 Plugin

Eclipse UML2 [ECU_1] is a project based on the EMF (Eclipse Modelling Foundation) [ECU_2] implementation of the UML 2.x OMG metamodel for the Eclipse platform in Java. The aim of UML2 is to support the development of UML based applications. UML2 does not provide a graphical view of the UML model but is more focussed on developing a robust implementation of the UML syntax.

For a graphical representation of a UML model Eclipse provides the UML2 Tools project [ECU_3]. This consists of a set of Graphical Editing Framework (GEF) based editors for viewing and editing UML models. The project is currently in the incubation (validation) phase. Currently, the UML2 Tool supports most but not all of the UML 2 model diagram types.

Eclipse UML2 also relies on EMF to provide support for code generation. Generating code involves a number of steps; firstly, an intermediate Ecore model (.ecore) has to be created. This model is a cut down version of the UML model, which contains the basic definition of the classes, their attributes and relations. EMF also creates a Genmodel that provides information about how the code should be generated. These two models allow EMF to generate code for the UML model. There are a number of ways to provide the UML model definition to EMF. These include an UML model that is EMF compatible, an XMI document, using annotated java interfaces with model properties or providing an XML schema representation of the model.

Once the UML model has been loaded and the Ecore model and Genmodel have been generated EMF relies on two tools for generating code, JET (Java Emitter Templates) and JMerge (Java Merge). JET is a generic template engine that can be used to generate SQL, XML, etc, but Java source code is the most common. These templates are customizable hence they can generate the code required for specific needs. The standard Java code outputted by EMF will include an interface and a corresponding implementation class for each class in the model. The interfaces contain getters and setters for each attribute and references of any related classes. EMF can generate efficient code for all modelling concepts such as two-way references, multiplicity, etc. and any user customization will not be lost if the model is regenerated, as EMF can distinguish the difference between generated and user written code.

4.3.2.4 Papyrus

Papyrus is a sub-project under the Eclipse Model Development Tools (MDT) project. While MDT aims to provide an implementation of industry standard metamodels and tools for developing models based on those metamodels. Papyrus aims to provide an integrated, user-consumable environment for editing models based on UML and other related languages such as the Systems Modelling Language (SysML). The tools features of UML and SysML are organized into different topics which are: Diagram editors, Diagram editors integration, EMF Search integration and EMF Validation integration.

The Papyrus project intends to be a focal point for UML and SysML providing an integration and extension means for diagram editors, profile design and support mechanisms and generic modelling tools such as search and validate. The project aims at flexibility and openness that should allow diagram extensibility so that diagrams can be completed in an iterative way. The project also provides
a customizable GUI in order to support the definition of tools like new palettes, properties views and new specific outlines.

For UML diagrams, Papyrus focuses on the MDT UML2Tools project which is a set of editors based on the GMF, for viewing and editing UML models and aims to integrate the features of this project into the MDT one. It also introduces extensions to the UML2Tools to maximise collaboration with MDT.

The papyrus sub-project itself acts as a container of other finer-grained operating projects which are listed as follows:

- The Backbone project. This is a multi-diagram container providing an infrastructure that allows to integration of any kind of diagrams coming from existing tools and modellers.
- The backbone connectors project, focuses on the creation of connectors to re-use existing editors such as the MDT UML2 Tools, the EMF/UML2 Tree Editor, Papyrus v1.x and TopCased v2.
- The UML Diagrams project provides graphical editors for UML Structure Diagrams, UML Behaviour Diagrams and Profile Diagrams (these diagrams should cover 100% of the official UML2 specifications). The UML Structure Diagrams are composed of Class Diagrams, Component Diagrams, Composite Structure Diagrams, Deployment Diagrams, Object Diagrams and Package Diagrams while the UML Behaviour Diagrams are composed of Activity Diagrams, Use Case Diagrams, State Machine Diagrams, Sequence Diagrams, Communication Diagrams, Timing Diagrams and Interaction Overview Diagrams.
- The SysML Diagrams project. Aims to provide graphical editors for all SysML diagrams introduced in the OMG SysML specifications.
- The Facilities Project intends to provide facilities to allow integration of Papyrus with other Eclipse projects.
- The UML-Diagram Interchange (UML-DI) project aims to ensure interoperability with other UML tools through import and export procedures.
- UML Profiles. Papyrus will host specific tooling on top of OMG and standards UML Profiles. These Profiles will be contributed to the MDT project.

The key features of the Papyrus project are:

- full UML2 standard compliance,
- compliance with Eclipse UML2,
- compliance with the Diagram Interchange standard which enables exchange of UML documents between different software tools,
- extensible architecture to allow users to add new features such as new diagrams,
- inclusion of facilities to support development of UML2 profiles,
- support for nested profiles,
- provisioning of facilities to write OCL (Object Constraint Language) constraint.

At present, a set of add-ons for Papyrus are available. Examples of add-ons are model transformation tools, code generators and UML profiles. The UML profiles available nowadays are the MARTE (Modelling and Analysis of Real Time and Embedded systems) UML profile, the SysML UML profile, the CCM (CORBA Component Model) profile, the LwCCM (Lightweight CCM) UML profile and the EAST-ADL (EAST Architecture Description Language) profile. The MARTE Papyrus plug-in implements the OMG specification of a UML profile for MARTE providing support for the different steps of the development process, which are specification, design and validation/verification. The UML profile for SysML is a new modelling language for system engineering. The papyrus plug-in for CCM and LwCCM implement the OMG specifications of the UML profile for CCM and LwCCM respectively.
Finally, EAST-ADL is an architectural description language to model automotive embedded systems and the homonymous papyrus add-on allows describing East-ADL2-compliant models. It aims to implement all the EAST-ADL2.0 stereotypes.

As code generators, Papyrus includes an add-on for a Java code generation framework based on UML models and an add-on for a C/C++ code generator. The Java code generator is powered by Acceleo. The C++ code generator relies on JET. The C code generator is based on Acceleo.
5 Specification

5.1 Introduction

While section two dealt with gathering requirements from various sources, this section deals with formalising those requirements. The requirements gathered in section two are translated into atomic items and matched to components identified in the architecture. This assigns responsibility to those components.

5.2 Requirement Index

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Statement</th>
<th>Primary Components Involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>###D3.2-Cl.1 Use Cases###M.0100</td>
<td>Teagle Portal shall incorporate a VCT request designer which will enable Customers to request/define a desired testbed facility or topology.</td>
<td>Teagle Portal Creation Client User Management Search VCT Tool Resource Registry</td>
</tr>
<tr>
<td>###D3.2-Cl.1 Use Cases###M.0150</td>
<td>Teagle Portal shall present Customers with feedback on the availability of requested resources, testing facilities or testbed topologies.</td>
<td>Teagle Portal Calendar Search Resource Reservation Resource Registry</td>
</tr>
<tr>
<td>###D3.2-Cl.1 Use Cases###M.0200</td>
<td>Software Images of testbed components should be provided in the form of a resource stored in a Central Repository.</td>
<td>Teagle Portal Search Resource Registry</td>
</tr>
<tr>
<td>###D3.2-Cl.1 Use Cases###M.0250</td>
<td>Panlab federation shall provide support for the set-up of an SME local testbed in the form of a service offered to Panlab customers.</td>
<td>Teagle Portal Additional Tools/Support Tool Search Calendar User Management Resource Registry Testbed Reservation</td>
</tr>
<tr>
<td>###D3.2-Cl.1 Use Cases###M.0300</td>
<td>For an outsourced testbed environment, test results should be gathered exclusively by the customer without intervention of the Panlab Office.</td>
<td></td>
</tr>
</tbody>
</table>

Class 2: Test of Services and Applications

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Statement</th>
<th>Primary Components Involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>###D3.2-Cl.2 Use Cases###M.0350</td>
<td>The Panlab Office and the Panlab federated testbed shall be able to provide users/user groups for testing purposes in the form of a resource.</td>
<td>Teagle Portal Additional Tools/User Group Search Calendar</td>
</tr>
<tr>
<td>Use Case ID</td>
<td>Description</td>
<td>Components and Tools</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>M.0400</td>
<td>Teagle's VCT request designer shall also enable Customers to request testing environments over which specific services developed by third parties (not belonging to the federation) could be tested (sometimes concurrently in different environments).</td>
<td>Teagle Portal, Creation Client, Search, VCT Tool, Resource Registry</td>
</tr>
<tr>
<td>M.0450</td>
<td>Teagle’s VCT request designer shall enable Customers to request testing environments over which specific components (network components, terminals) or applications developed by third parties (not belonging to the federation) could be tested (Support for User Driven Innovation).</td>
<td>Teagle Portal, Creation Client, Search, VCT Tool, Resource Registry</td>
</tr>
<tr>
<td>M.0500</td>
<td>Teagle Portal shall provide the means for Customers to view current/past test results and logs.</td>
<td>Teagle Portal, Visualisation Tool, User Management, User/Customer, User Roles, Test Results</td>
</tr>
<tr>
<td>M.0550</td>
<td>Teagle Portal (VCT request designer) should enable customers to define and configure the environment in which interoperability, conformance and certification testing shall be performed.</td>
<td>Teagle Portal, Creation Client, User Management, VCT Tool, Orchestration Frontend, Orchestration Engine, Teagle Gateway, PTM</td>
</tr>
<tr>
<td>M.0600</td>
<td>Panlab platform shall reserve and allocate all resources required to enable integration with the component/service/protocol to be tested and create a testing experimentation field in which the interoperability/conformance and certification testing can be performed.</td>
<td>Teagle Portal, Search, Calendar, Resource Registry, Orchestration Frontend, Orchestration Engine, Resource Reservation, Orchestration Engine, Teagle Gateway, PTM</td>
</tr>
<tr>
<td>M.0650</td>
<td>Teagle portal should maintain a Test suite repository</td>
<td>Teagle Portal, Testbed Configuration</td>
</tr>
<tr>
<td>M.0700</td>
<td>Teagle portal should provide a search service of the Test suite repository</td>
<td>Teagle Portal, Search, Testbed Configuration</td>
</tr>
<tr>
<td>Task</td>
<td>Description</td>
<td>Related Components</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>D3.2-Cl.3 Use Cases</td>
<td>Teagle portal should provide a generic search service of the available testing tools</td>
<td>Teagle Portal Search Resource Registry</td>
</tr>
<tr>
<td>D3.2-Cl.3 Use Cases</td>
<td>Teagle portal should maintain a certification database of test suites</td>
<td>Teagle Portal Resource Registry</td>
</tr>
<tr>
<td>From Underlying Architecture (D2.1), Existing Testbed Repository and System Analysis (D3.1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3.2-Architecture</td>
<td>Teagle Portal shall support the establishment of Service Level Agreements between Customers and the Panlab Office.</td>
<td>Teagle Portal Additional Tools/SLA creator? Testbed Configuration repository</td>
</tr>
<tr>
<td>D3.2-Architecture</td>
<td>Repository should be capable of storing Customer information.</td>
<td>Teagle Portal User Management User/Customer</td>
</tr>
<tr>
<td>D3.2-Architecture</td>
<td>Teagle should include a Resource Reservation System.</td>
<td>Teagle Portal Search Calendar Resource Reservation</td>
</tr>
<tr>
<td>D3.2-Architecture</td>
<td>Teagle shall incorporate an intelligent Request Interpreter.</td>
<td>Repository Gateway Resource Registry VCT Registry Orchestration Frontend Policy Enforcement Orchestration Engine</td>
</tr>
<tr>
<td>D3.2-Architecture</td>
<td>Repository should be capable of storing Test Result information.</td>
<td>Test Results PTM</td>
</tr>
<tr>
<td>D3.2-Architecture/System Analysis</td>
<td>Databases storing Shared Information and Data Models shall be part of the Panlab repository.</td>
<td>Information Model Data Models</td>
</tr>
<tr>
<td>D3.2-Architecture/System Analysis</td>
<td>PTMs should allow for interconnection with 3rd party components/resources.</td>
<td>Teagle Portal Resource Registry Information Model Data Models Teagle Gateway PTM</td>
</tr>
<tr>
<td>D3.2-Architecture</td>
<td>A Constraints and Policies Database should exist as part of the Panlab Repository. Confidentiality of the data stored in the repository should be taken into careful consideration. Customers should be classified according to their confidentiality and privacy classification – for instance</td>
<td>User Management Users/Customers User Roles Policy Enforcement Information Model</td>
</tr>
</tbody>
</table>
5.3 Operational Aspects

This section will try to cover key operational aspects that emerge from Panlab concepts and affect Teagle’s architecture in terms of activity diagrams. Three major activities will be described here in detail. Teagle services involved during the fulfillment of these activities are also presented.

5.3.1 Activity 1: Customer creates a testbed and an SLA is established

The activity “Customer creates a testbed and an SLA is established” is given in figure 4. The result of this activity is the establishment of an SLA for the customer.

Customer logs into the Teagle portal, where Authentication and Role Assignment occur. At this point, authentication services and role assignment use the Users and User Policy data repositories. For this scenario it is assumed the user gets a role of a customer TestbedUser.

After log in, the TestbedUser might proceed with the activity of Set a new environment and Create an environment in order to create the testbed. Portal services like the Creation Client, Creation Environment and Visualization tool are utilized during these activities.

The TestbedUser might proceed with just the Search Resources activity and leave the activities of Set a new environment and Create an environment for later. During the Search Resources activity the TestbedUser enters different criteria or just browses through different categories. Different activities are followed to fulfill this request. The Search tool is used to search for matching resources. After that, certain criteria of the user might lead to the activities Check Resources Availability, Check Calendar and Check Partner Constraint (Policies) where the service of Search utilizes repositories such as the Resource Registry, Information and Data models and the Portal’s Calendar service.

Then the TestbedUser after successfully finding the necessary resources proceeds with the activity Request/Add resources to their testbed. Again, different services and repositories are utilized here like the Visualization Tool.

The TestbedUser after finishing with adding resources continues with the activity User finished adding resources. Teagle follows the Check Resources Availability activity to inform the user of calendar availabilities of the whole testbed and all resources. Then the Testbed user with the activity User requests/Sets configuration marks the configuration that is needed by each resource in order to properly run the tests. Again, this triggers Teagle’s Check Configuration activity, which tries to inform the user of any improper configuration or policy violations as early as possible.
The fulfilment of the parallel activities of *Check Resources Availability* and *Check Configuration* triggers the activity *Verification and Approval of Availability and Configuration*. During this activity which might also include human intervention, the proper configuration and availability is verified and approved by the Teagle service and/or Panlab office representative.

*Figure 4: Customer creates a testbed and an SLA is established*
After this a pre-reservation of resources occurs in the activity *Pre-Reserve Resources*. This activity proposes available calendar days in order to book the resources. This process might be time consuming and even require some human intervention.

Finally, the activity *Propose Schedule of Testbed based on all resources availability* just informs the user and maybe other partners of the Proposed Schedule. This activity is rather iterative until a final decision is made on schedule.

Upon completion of all of the above parallel activities, a proposal comes to the user which happens in the activity called *User Accepts Proposal*. The success of this activity triggers the following three parallel activities:

*SLA Preparation* activity. Might occur with tools offered by the Teagle service

*Reserve resources*. Resources are actually reserved and partners are informed automatically from the system.

*Schedule Test*. Calendar services are updated and partners might be informed automatically by the system.

The completion of the above three parallel activities leads to the *SLA Establishment* activity. The Teagle service manages this and presents it to the TestbedUser.

### 5.3.2 Activity 2: Provisioning according to a SLA

The activity “Provisioning according to a SLA” is given in figure 5. This activity might happen on a scheduled date.

The system starts with the activity of Process SLA. During this activity the system might check services for availability, status, health, etc. [maybe we need a HealthCheck service]. This triggers three parallel processes, which might happen even asynchronously: *Infrastructure Provisioning*, *Capacity Provisioning* and *Setup/Configure the Monitor Infrastructure*. (Details on these are given in D2.1).

Each one of the activities is fulfilled by following the activities:

*Prepare VCT Requests*. Teagle’s service VCT starts the preparation of VCT Requests.

After the preparation of the requests, the *Interpret VCT Requests* takes place which will transform the request to certain Orchestration specifications. Orchestration Frontend will be the service which will fulfil this activity. Upon termination of this activity the configuration could be stored for future use in the activity Store Configuration via the Repository gateway.

Then the *Orchestration* activity takes place where all the previous Orchestration Specification is prepared for execution. Orchestration Frontend and Orchestration Engine services will handle this activity.

Finally, during the *Execute* activity and after having the orchestration specification, a sequence of service requests is forwarded to PTMs. This activity utilizes the Orchestration Engine service.

The completion of all these activities triggers the activity, *Quality Assuring and Verification* and *Updating the Teagle’s repository with Testbed Approval*.

Finally, the Testbed is ready to be given to the customer. Teagle on the final activity *Handover Tesbed to customer* informs the customer of the successful configuration, any QA results and possible any configuration problems.
Figure 5: Provisioning according to a SLA
5.3.3 Activity 3: Partner Interaction

The activity “Partner Interaction” is given in figure 6 below.

This activity takes place when a partner wants to offer his resource(s) to the service. After login to the Teagle service, the activity *Manage Testbed offerings* starts. This activity allows Teagle’s Portal service to offer a means for a partner to manage his resources. In order for a partner to offer a service, five parallel activities must be fulfilled:

- **Define resource availability.** Define calendar days/times where the resource will be available. This can be done through Teagle’s portal interface by utilization of Calendar repositories.
- **Define policies.** Define policies for using the resource. This can be done through Teagle’s portal interface by utilization of Entity Policies repositories.
- **Describe resources.** Describe the resources (based on provided templates). This can be done through Teagle’s portal interface by utilization of the Information Model and the Resource Registry repositories.
- **Supply means of configuration.** Supply configuration details of the resource. This can be done through Teagle’s portal interface by utilization of the Information Model and the Resource Registry repositories.

![Figure 6: Partner Interaction](image-url)
The final three activities trigger the QA process of having the resource under acceptance and passing Qualification Tests. As a result, the QA process updates Teagle’s database and gives the green light for accepting the resource.

5.4 Entities

This section describes the entities that play a part in Teagle as a whole. From this point of view the descriptions of the entities is introductory and serves to give context for the supporting repository elements.

![Teagle Internal Architecture Diagram]

**Figure 7: Teagle Internal Architecture**

5.4.1 Teagle Portal

The Teagle Portal is the front-end component of the Teagle and the first point of contact for PII users. Its primary function is to provide a web portal to allow users to access the PII testbed federation. The Teagle Portal primarily supports two basic user types, PII customers and PII partners. The typical use case scenario for a PII customer involves facilitating the customer to create a testbed that realizes their unique testing requirements. The Teagle Portal is the gateway to fulfilling this request as it
allows the customer to view or search a full list of the testbed resources available throughout the federation and select the resources required. The process was described in detail in Section 5.3.1. The Teagle Portal will also be accessed by the PII partners to manage their testbed offerings. Initially, the partners will have to install the testbed resources they are supplying to the federation through the Portal. This involves describing the resources in a manner compatible with the DEN-ng information model. It is anticipated that this process will be automated through automatic resource discovery processes. The PII partner can also manage the usage policies/constraints and define the availability of the resources they are supplying to the PII federation. See Section 5.3.3 for more details on this process. This data is stored in the Resource Policies and the Resource Reservation repositories respectively.

Finally, the Teagle Portal will provide support for the customer by supplying the contact information of a representative contact from the partner testbed(s) involved in the customer’s VCT. These contacts can assist the customer with technical issues regarding setup. This satisfies Requirement ##D3.2-C1.1 Use Cases##M.0250. The Teagle Portal comprises of five functional components that are described in the following sections.

5.4.1.1 Search
The Search Tool allows the PII Customer to find testbed resources that suit their testing requirements. The tool accepts search requests from the PII Customer for specific resources or the user can browse the resources in a catalogue style. To search for resources that match the search request, the tool will have to access the Resource Registry via the Repository Gateway interface.

5.4.1.2 Calendar
The Calendar component will manage resource reservations thus satisfying Requirement ##D3.2-C1.1 Use Cases##M.0150. It will comprise of a calendaring system that will allow testbed resources to be reserved for the time period that the customer requires to test their system. The reserved testbed resources will be stored in the Resource Reservation repository.

5.4.1.3 User Management

D4.1 [ENT_1] section 1.1 has defined four high level user types:

- The PII office
- A PII partner
- A PII customer
- A PII end user

These different user types play different roles in the PII environment. The User Management component will provide a means to treat each of these users differently by managing their authorization and level of access to the system. The User Management component may be accessed by other Teagle functional components to verify user authentication and access permissions before an operation is executed. The User Management component interacts and stores this data in the User/Customer repository.

5.4.1.4 Visualisation Tool
The Visualisation Tool will provide support for the presentation of different types of data. For example, it may entail a graphing tool to enable a PII customer view their testing results.

5.4.1.5 Creation Client
The Creation Client will allow the PII customer to create their testbed graphically. It is envisaged that this tool will have a drag and drop user interface where the customer can very simply select the testbed resources they require to meet their testing needs. These resources can then be organised in the desired topology in a drag and drop style. Each testbed resource and the links between the resources can be
individually configured through the user-friendly UI. This entity satisfies Requirement ##D3.2-C1.1 Use Cases##M.0100 and Requirement ##D3.2-C1.3 Use Cases##M.0550.

5.4.1.6 Additional Tools

This section is for future tools that may be added to the Teagle Portal as required. Examples may include a Monitoring tool or a Report generator.

5.4.2 Repositories

5.4.2.1 Repository Gateway

The Repository Gateway is the single access point for the Teagle repositories thus it acts as a gate-keeper directing all requests towards the correct data service. It may be implemented using an Enterprise Service Bus (ESB) architecture.

5.4.2.2 VCT Registry

The VCT Registry will store the currently active VCT configurations within the PII environment. The VCTs will have a unique 32-bit identifier so they can be easily retrieved for further testing at a later date even if the VCT has been torn-down since the initial testing period.

5.4.2.3 Resource Registry

The Resource Registry will be a repository to store the details of each testbed resource supplied by the testbed owners. This information will be stored based on the information model (the DEN-ng model) as described in section 4.2.2. The testbed resources will have an identifier to uniquely address the resources within the different testbeds of the federation.

5.4.2.4 Resource Reservation

The Resource Reservation repository is used by Teagle to track the availability of testbed resources. The repository will be used in conjunction with the Calendar component to enable testbed resources to be allocated for the time period specified in the SLA. It will also provide a method to mark the resources as reserved for a defined period of time and ensure that data concurrency and consistency prevail. This system will also have a requirement to handle different time zones. The Reservation system is outlined in Requirement ##D3.2-C1.3 Architecture##M.0950.

5.4.2.5 Resource Policies

Resource Policies provide the ability for the testbed owner to set terms and conditions regarding the use of the testbed resources they are supplying to the PII testbed federation. Such policies will need to be stored in a repository (Resource Policies) and made available through the Repository Gateway for the utilization of the Policy Evaluation component. An example of such a policy can occur when a testbed owner may not want to allow a resource they are offering to be used by a particular type of customer. Another example of a policy might consider the business model behind a testbed federation. For instance, PII partners might be able to use other partners’ resources under different conditions than regular customers (e.g. at a reduced cost). The resource policy repository satisfies Requirement ##D3.2-Architecture##M.1200.

5.4.2.6 User/Customer

User/Customer is a repository that stores a user profile for each user of Teagle. Each user once authenticated should be able to view and administer their account and profile data. A User profile will contain standard data such as Name, Company, etc. to validate the user’s identity. This fulfils Requirement ##D3.2-Architecture##M.0900.
5.4.2.7 Testbed Configuration

Testbed configurations will be stored within Teagle on behalf of PII customers in the Testbed Configuration repository, satisfying requirement ##D3.2-C1.3 Use Cases##M.0650. Storing the testbed configurations allows the customer to reproduce their testing environments for further testing at a later date and to allow reproducibility of experiments. VCTs are private to each PII customer but it may be advantageous for PII to share commonly used VCTs thus allowing other PII customers to use these testing configurations to facilitate reuse and allow for easy setup.

Storing the testbed configurations involves storing the VCT request and possibly the corresponding orchestration specification. Each VCT will be assigned a unique numeric handle that will never be reused so VCTs can be re-established even after teardown. To reinstate the same testbed configuration the orchestration specification is all that is required. However, if any of the resources used in the original testbed configuration are unavailable, the original orchestration specification will not be usable. In these situations it has been foreseen that the VCT request will be required to generate a new orchestration specification. Thus, storing the VCT request may also be necessary.

5.4.2.8 User Roles

A separate repository from the User/Customer repository may be used to store user roles that can be associated with the different user types. This repository called User Roles will allow a more flexible level of control over the system regarding access rights.

5.4.2.9 Test Results

The customer’s experimentation results can be stored within a repository (Test Results) in Teagle. This functionality is provided to facilitate customer’s that may not have the resources to store this large amount of data. Legal ownership of these test results will have to be agreed beforehand. This satisfies Requirement ##D3.2-Architecture##M.1050 and part of Requirement ##C1.2 Use Cases ##M.0500.

5.4.2.10 Information Model

DEN-ng has been chosen as the Information Model to represent the testbed resources in a manageable way. It could also be used as the framework for implementing many other necessary concepts within the PII environment such as Policy, User Roles, etc. The DEN-ng model will be represented in UML [ENT_2]. The necessity for an Information Model is specified in requirement ##D3.2-Architecture/System Analysis##M.1100.

5.4.2.11 Data Models

The Data Models are closely linked to the information model as they represent a specific technology implementation of the information model. The data models will be represented in XML [ENT_3] or RDF [ENT_4] and are outlined in requirement ##D3.2-Architecture/System Analysis##M.1100.

5.4.2.11.1 Orchestration Frontend

The Orchestration Frontend transforms the PII customer’s VCT request for a specific testbed configuration into an orchestration specification. The Orchestration Frontend is composed of three sub-components and is specified in detail in D3.3 [ENT_5] and satisfies Requirement ##D3.2-Architecture##M.1000.

5.4.2.12 Policy Enforcement

This component will allow any resources that have been selected as a result of a customer’s VCT request to be validated so policies associated with their use are not violated. It will thus need access to the Resource Policies repository.
5.4.2.13 Creation Environment

This *Creation Environment* is one of two components of the VCT tool. It is the backend component for the *Creation Client* and will allow for manual intervention if the *Request Processor* fails to generate a valid orchestration specification.

5.4.2.14 Request Processor

The *Request Processor* is the second component of the VCT Tool. It automatically translates the VCT request into an orchestration specification that can be outputted to the *Orchestration Engine*. In simple cases, the VCT request may not need any translation, so in these cases the *Request Processor* simply forwards the VCT request to the *Orchestration Engine*. However, in more complicated requests the *Request Processor* may need to resolve dependencies and pre-/post- conditions between different resources. These cases will require a certain level of intelligence in order to generate an orchestration specification. The orchestration specification will follow a standard such as BPEL [ENT_6] or UML [ENT_1], further information can be found in PII D3.3 Testbed Orchestration Service Specification [ENT_5].

5.4.3 Orchestration Engine

This is the final functional component in the Teagle to process the flow of the customer’s VCT request. The *Orchestration Engine* accepts the orchestration specification from the *Orchestration Frontend* and generates executable code that in turn is forwarded in a sequence of service requests to the PTM via the *Teagle Gateway*. These service requests will be sent as Web Service calls using SOAP [ENT_7] messages described by WSDL [ENT_8] documents.

5.4.4 Teagle Gateway

The *Teagle Gateway* is a public interaction component enabling access to/from the Teagle. It will primarily be used to communicate with the PTM over the T1 interface and provides interfaces to the *Orchestration Engine* (TG) and the *Repository Gateway* (REP), see figure 7.

5.5 Reference Points

This section details the reference points connecting Teagle components to the Repository and details the internal reference points within the Repository. Where appropriate, proposals for specific protocols and technologies are made.

5.5.1 External Reference Points

The external reference points connect the repository to other Teagle components. The figure 7 shows these links and the current architecture of components. Based on this architecture and the requirements placed on the repository, the nature of these reference points are detailed below.

5.5.1.1 REP1

This generic interface to the repository should provide a standards-based access point to data in the repository. The stores of data consist of XML databases, Relational Databases, serialized XML files and other types of sources. The structure of the information is based on the generic information model. The information model provides a platform independent means of describing the information and associated relationships. Teagle components operating on this data will typically only need to access a subset of this information at any given time.

The requirements for this reference point are based on the proposed design of the components accessing it. As can be seen in the architecture figure 7, the Portal, the VCT tool in the Orchestration Front End and the Teagle Gateway all use this reference point to access the Repository Gateway.
The Portal and more importantly, the sub-components of the Portal, use this reference point for object retrieval and persistence. These sub-components include the searching facilities, user management, resource reservation scheduling and visualisation of resources. The VCT tool uses this reference point for handling VCT creation, design and providing descriptions of the testbed resources which compose the resulting testbed. The Teagle Gateway uses this reference point to process incoming registrations of federated testbed partners, testbed resource registrations from these partners and handle testbed resource status events. To account of the differing development platforms of the various components, data objects could be represented in XML using a cross-platform data definition specification.

The components supporting this reference point will need to abstract data specific operations such as querying, updating, transaction and concurrency management. Thus a typical database management system would not be sufficient here. However, Object-Relational Mapping or Service Data Object [REF_1] functionality would be more appropriate as these functions are provided and extended. In summary, the REP1 reference point should provide web service based access to XML representations of data objects aggregated from multiple data sources in the Repository and manage the handling of these objects as operations are carried out on them.

5.5.1.2 REP2

The reference point labelled REP2 is a management interface for the policy evaluation entity of the repository. This interface should allow the creation, updating, deleting and viewing of policies in the repository. It should use a standards-based protocol for formatting and exchanging this information. The Policy Enforcement component is used to evaluate access requests to testbed resources that could be used in composing a VCT. These requests are evaluated based on the identity and affiliation of the user in question, the category of testbed resource being requested and the access policy attached to the testbed resource by the federated testbed partner.

As it is envisaged that the policy enforcement would use the Open Mobile Alliance Policy Evaluation [REF_2], Enforcement and Management Architecture specification [REF_3] as a basis for its operation, it is likely that this reference point would use the Extensible Mark-up Language Configuration Access Protocol (XCAP) [REF_4].
6 Conclusion

As a guide to developers of the Teagle Repository, this section outlines the identified requirements and the technologies best suited to addressing them.

Many of the persistence based requirements (M.0500, M.0600, M.0650, M.0750, M.0800, M.0900, M.1050, M.1100, M.1200) specify the need for a database. We outlined two such candidates in the form of Relational Database Management Systems and XML Databases. XML Databases are advantageous for specific tasks such as integration with XCAP for user configuration and policy management and integration with Orchestration systems for storing automation scripts. Relational Databases provide known integration with existing persistence layers and object-relational mappers.

While databases may address the storage needs of components, there are also supporting requirements for information management. When Teagle components interact with information, it is presumed that issues such as concurrency and transaction handling are implicitly dealt with. While some databases provide frameworks to do this, it is neither inherent nor consistent. For this reason, it is recommended that a data access layer be used to ensure that this consistency is provided.

The components dependent on the Repository also require certain higher level services. The Teagle Portal contains a number of components that may require service composition from the basic data services to realise the business logic for these user interfaces. To this end, the information in the Repository should be made accessible through services (e.g. web services) to enable easy integration with other Teagle components and compositions. Services such as Resource Reservation, Orchestration Scripting, Testbed Resource Registration and Logging should be used and re-used to form and extend the Teagle implementation.

Many of these services are the realisation of platform specific data models which in turn are based on the underlying information model used to describe testbed resources and their relationships. This information model should be manageable and extendible through similar services.

While this specification has tried to remain technology agnostic, a common characteristic of identified tools has been the support or use of Java. This implies that Java forms a homogeneous environment for development. While this may be the case, the recommendation for web services based inter-component communication should allow for different implementation languages and development environments without impacting on integration.
References

Panlab II website - http://www.panlab.net/

Objective References


Requirements References

[REQ_1] PII D1.1 Use Case Descriptions
[REQ_2] PII D2.1 Technical infrastructure specification
[REQ_3] PII D3.1 System Analysis Report

Summary of Underlying Architecture References

[SUA_1] PII D2.1 Technical infrastructure specification

PII Logging - References


Relational Databases References


Information and Data Model References

SID References

IPSphere References

DEN-ng References

UML References

WSDL References
[WSDL_1] W3C WSDL 1.1 - http://www.w3.org/TR/wSDL
[WSDL_2] W3C WSDL 2.0 - http://www.w3.org/TR/wsd120/
[WSDL_3] OWL-S - http://www.w3.org/Submission/owl-s

RDF - References
[RDF_1] http://www.w3.org/RDF/
SoAML - References


Semantic Annotations - References


SAWSDL - References


OWL - References


Eclipse UML2 - References


Entities – References

[ENT_1] PII D4.1 ‘Testbed quality description: Criteria and metrics’

W3C Extensible Markup Language (XML) specifications, http://www.w3.org/XML/Core/

Resource Description Framework, W3C website, http://www.w3.org/RDF/

PII D3.3 ‘Testbed Service Description Specification’


W3C SOAP Version 1.2 specification, http://www.w3.org/TR/soap12-part1/

W3C Web Services Description Language (WSDL) specification, http://www.w3.org/TR/wsd1

Reference Points – References


[REF_2] Open Mobile Alliance - http://www.openmobilealliance.org
