FI-CoDE

Technical Description

Introduction

The goal of this document is to describe at different levels of detail what the FI-WARE Developer Community and Tools (DCT) will deliver. The primary goal is to offer a multi-functional development environment enabling the development and management of the applications built to address the needs\challenges of the Future Internet and based on the adoption and integration of the FI-WARE project results. In this respect, exploiting the experiences and knowledge that many developers worldwide already have with Software Development Kits (SDKs), Integrated Development Environments (IDEs) and Collaborative Development Environments (CDEs), this time these environments will not be offered in isolation but as an integrated one. The challenge, then, will be to provide a single access point to the Future Internet developers offering at the same time:

- a powerful, agile and complete development suite;
- the possibility to rely on the support of the community;
- the possibility to test, deploy and monitor the final results anywhere and anytime.

This will be the FI-CoDE: a set of tools, code samples, documentation, compilers, headers and libraries the developers can use to create Future Internet applications and services based on the FI-WARE results (e.g. FI-WARE Compliant Platform Products).

Relevant Information

In this paragraph is reported the relevant information that is used or taken into account for the definition of the solutions that will be produced by the DCT. All that information is extracted from the Description of Work (DoW) or from the FI-WARE High Level Description (HDL) where it's possible to find the whole set of definitions of the most frequent terms used in the project. Here they are reported only extracts with the most significant aspects highlighted and necessary for the comprehension of the document itself.

Roles

Here they are reported the main roles to be considered for the definition of the high level scenarios (FI-CoDE Use Cases).

- **FI-WARE Application/Service Provider:** A company or organization which develops FI applications and/or services based on FI-WARE GE APIs and deploys those applications/services on top of FI-WARE Instances.
- **FI-WARE GE Provider:** Any implementer of a FI-WARE GE. The open and royalty-free nature of FI-WARE GE specifications will allow parties other than partners in the FI-WARE consortium to develop and commercialize products that are in compliance with FI-WARE GE specifications.
- **FI-WARE Instance Provider:** A company or organization which deploys and operates a FI-WARE Instance.

Assets

Here are listed the assets that have been identified and that are relevant for the definition and realization of the outcomes (see *Major Outcomes* chapter) of the DCT.

- **FI-WARE Generic Enabler (GE):** A functional building block of FI-WARE. Any implementation of a Generic Enabler (GE) [...] supports a concrete set of Functions and provides a concrete set of APIs and interoperable interfaces that are in compliance with open specifications published for that GE.
    - Signature and behavior of operations linked to APIs (Application Programming Interfaces) that the GE should export. Signature may be specified in a particular language binding or through a RESTful interface.
    - In charge of GE Providers.
- **FI-WARE Compliant Platform Product:** A product which implements, totally or in part, a FI-WARE GE or composition of FI-WARE GEs in compliance with open specifications published for that GE.
    - It contains the client component that allows the interaction with the FI-WARE Compliant Platform Product itself. This component is,

programming, language dependent and may be specific for that particular FI-WARE Compliant Platform Product implementation.
    - In charge of GE Providers.
- **FI-WARE Instance:** FI-WARE Instances are built by means of integrating a concrete set of FI-WARE compliant Platform Products, and typically a number of complementary products to gain differentiation on the market or to enable monetization (e.g. Specific Enablers, integration with own Operating Support Systems, Billing or Advertising systems).
    - In charge of FI-WARE Instance Provider.
- **FI-WARE Testbed:** A concrete FI-WARE Instance operated by partners of the FI-WARE project that is offered to Use Case projects within the FI-PPP Program, enabling them to test their proof-of-concept prototypes. The FI-WARE Testbed is also offered to third parties to test their Future Internet Applications although support to them is provided on best-effort basis.
    - ~~Provided by Experimentation Support, Integrated Testing and Validation activity.~~
- **Future Internet Application:** An application that is based on APIs defined as part of GE Open Specifications. A Future Internet Application should be portable across different FI-WARE Instances that implement the GEs that Future Internet Application relies on, no matter if they are linked to different FI-WARE Instance Providers.

It is worth to note that inside the rest of the document the term asset will be used to avoid repeating one, or more, of the previously listed elements. This choice goes in the direction of simplifying the readability of the document itself.

Major Outcomes

The major outcomes in terms of tools and methods are listed below with a brief description. Also these items may be included in the meaning of asset inside the rest of this document.

- **Collaborative Development Environment (CDE):** The integrated environment that provides the most common features to manage and develop a software project. It includes tools and services such as ticketing system, version control system, wiki, mailing list, programming language SDKs, DBMS, web server, application server.
- **CDE Manager:** The service that allows to create and manage CDEs in a virtual infrastructure (based on a cloud infrastructure). It's possible to start creating CDEs from predefined templates or getting empty virtual machines to manage by yourself. This is the main entry point for who wants to develop a Future Internet application based on a FI-WARE Instance.
- **Integrated Development Environment (IDE):** The main tool adopted by the developers to produce their solutions. It will include a set of predefined plug-ins to extend its functionalities.
- **Catalogue:** The identified entry point where to publish and search for FI-WARE Compliant Platform Products.
- **Deployment Tools:** The tools that will be in charge of the Future Internet

Application deployment into the selected CDE.

- **Test & Validation tools:** The tools that will be in charge of the Future Internet Application testing and validation along the application life-cycle. Also some of the tools can be used to test and validate GEs. This enables FI App developers to pick GEs with a tested quality and build on them.
- ~~**Forge:** The integrated environment that provides the most common collaboration features (e.g. version control system, task list, document management, release management, forum). The forge tool is in charge of providing the Collaborative Development Environment (CDE) for the FI-WARE GE Providers and FI Application/Service Providers roles activities.~~
- **Learning Management System (LMS):** The service that support the delivery of learning objects, used in the context of the project to provide e-learning sessions related to FI-WARE as a project/initiative, GEs, FI-CoDE platform (how-to-use).
- **Methodology:** Given the complexity and the novelties associated to implementing FI Apps based on FI-WARE, FI-CoDE should come with a set of methods which will support the developers to build such applications.

This basic set of tools will be completed case by case by complementary elements (e.g. supporting scripts, documentation, compilers and methods).

To describe the main functionalities provided by this environment we provide Use Case diagrams and related scenarios together with architectural and technological views supported by textual descriptions for each specific architectural component.

FI-CoDE Architecture

Overview

To support the described Use Cases, FI-WARE provides the FI-CoDE platform that is an harmonized set of tools and methods composed by both clients and servers components.

From the highest point of view the FI-CoDE platform is composed by the elements (also represented in the next figure):

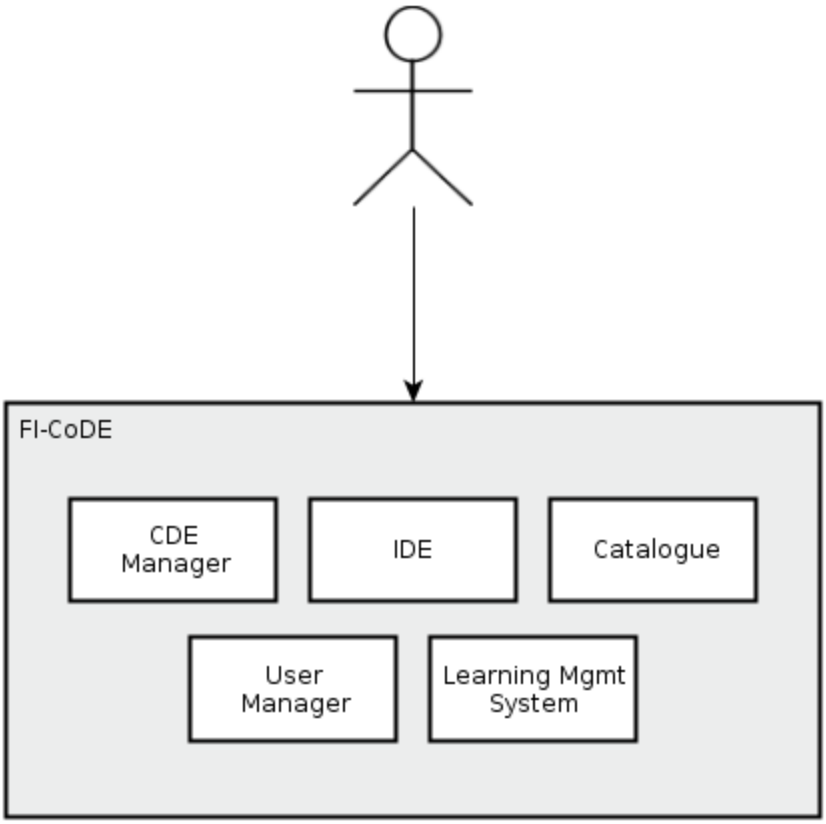
- CDE Manager;
- IDE;
- Catalogue;
- User Manager;
- Learning Management System.

Each one of these is detailed in the remainder of this chapter.

The actors that have been identified for the interaction with the FI-CoDE platform are the Project Manager (PM), the Technical Manager (TM) and the Developer (DEV).

To the end user of the FI-CoDE platform will be presented by an homogeneous interface able to merge the various services and, at same time, allowing the user to move from one service to another without the need to login again. This user experience is achieved, for the first aspect, adopting a strong front end integration, while for the second aspect,

by putting in place a Single Sign On mechanism based on a common User Management System in charge of maintaining one single identity for all the integrated services of the platform.



FI-CoDE Overview

Assumptions

To implement and validate the proposed FI-CoDE platform, some assumptions has been taken, and they affect more the project developed using the FI-CoDE platform than the platform itself. These assumptions are reported here:

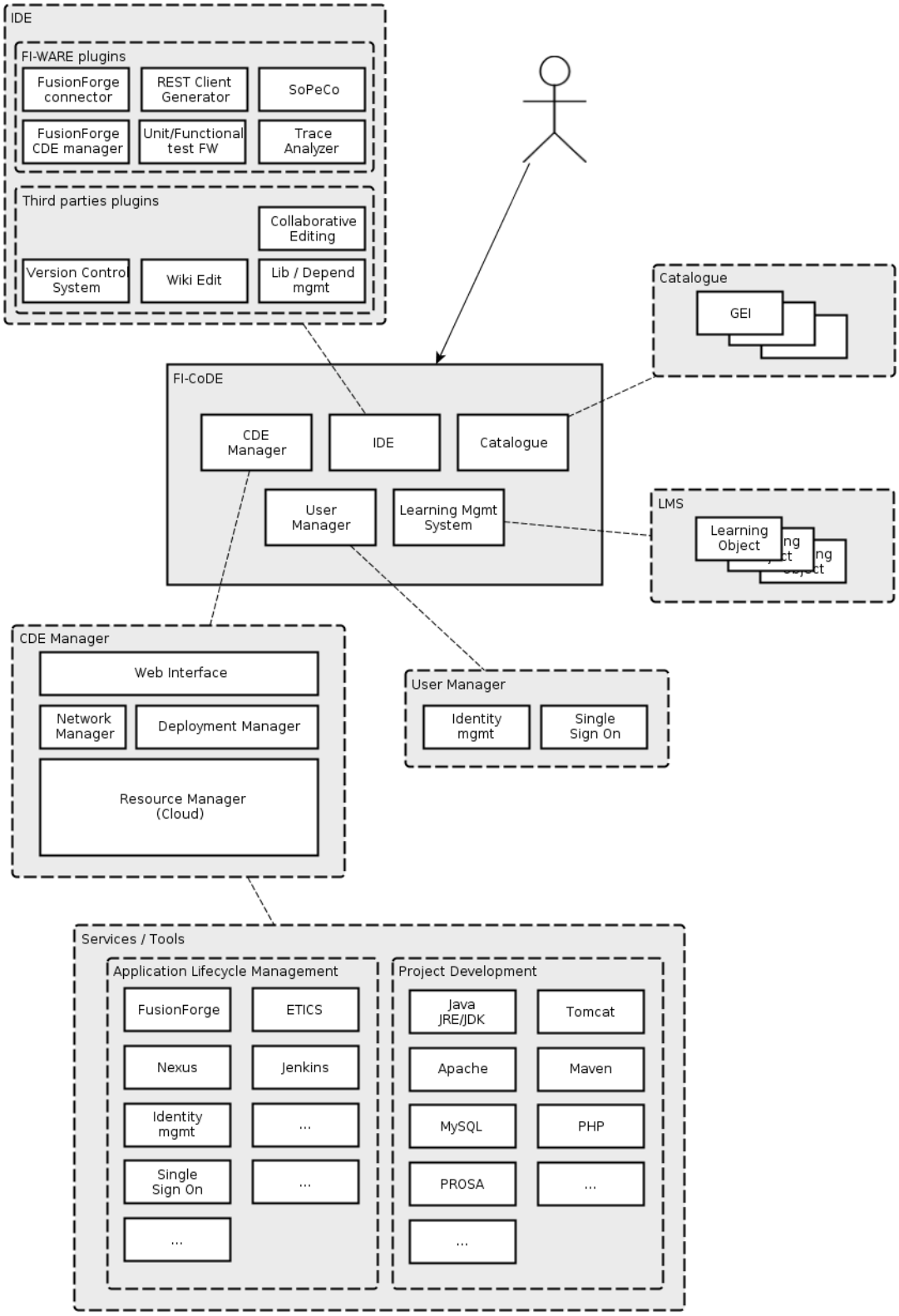
- All FI-WARE Compliant Platform Products are assumed to be e-Services (e.g. RESTful, Web Services, …);
- The project (FIApp or GEI) is based on Java programming language for implementing business logic and interaction with GEIs.

~~the technology baseline identified is composed by:~~

- ~~Eclipse IDE + Plug-ins (more details on a dedicated section);~~
- ~~Forge solutions:~~
  - ~~FusionForge (as FI-WARE reference implementation)~~
  - ~~QualiPSo Factory (development still in progress)~~
  - ~~GitHub (to be evaluated)~~

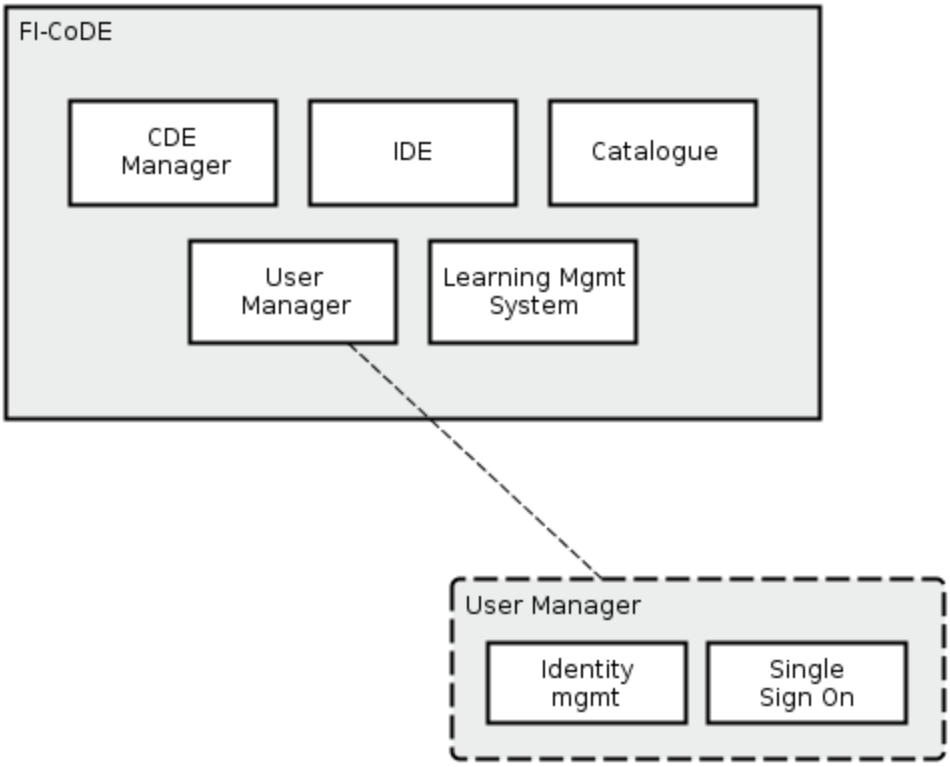
- ~~Drupal content management system (as FI-WARE Catalogue reference implementation)~~

The next sections of this chapter elaborates in detail about every macro area reported in the previous “FI-CoDE Overview” figure and highlighted in the next figure “FI-CoDE Architecture”.



FI-CoDE Architecture

User Manager

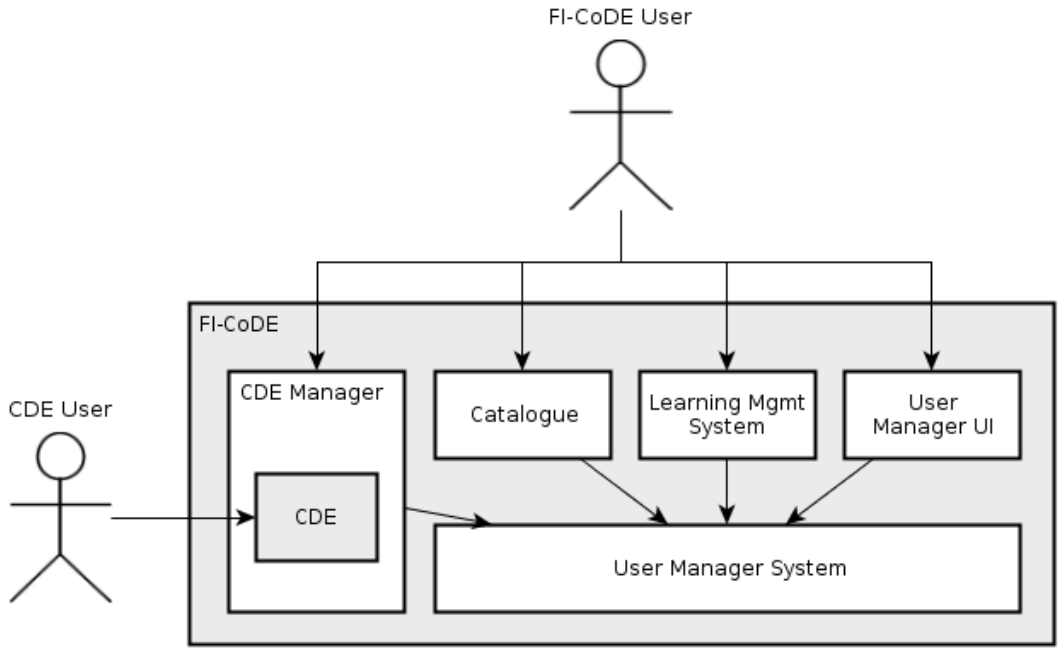


FI-CoDE - User Manager

The FI-CoDE platform provides a solution for managing user credentials (User Management System - UMS) that have to interact with all the services available in the platform, that are: CDE Manager, Catalogue, Learning Mgmt System and User Manager.

The two main types of users that are considered are:

- FI-CoDE Users;
- CDE Users.



FI-CoDE/CDE Users

For the first ones, the initial solution takes into account the possibility to access all FI-CoDE services (CDE Manager, Catalogue, Learning Mgmt System and User Manager UI) using the same user credentials. In addition to this a Single Sign On (SSO) extension is put in place in order to simplify the navigation from one service to another, without the need of typing the user and password again.

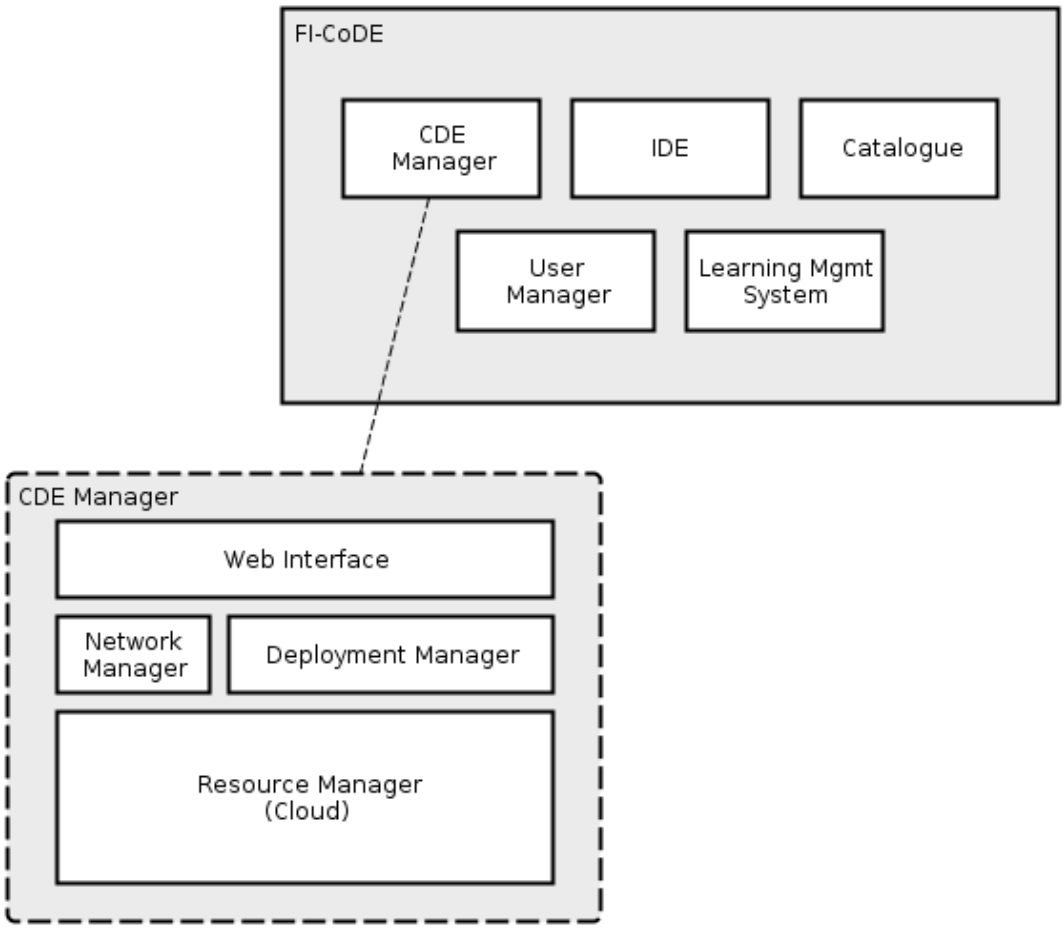
To improve the flexibility of the system it's possible to log into FI-CoDE also using an already existing account provided by the most relevant social networks.

The scenario described here considers to manage authorizations for users, and groups, within every specific service. A centralized authorization configuration will be evaluated depending on the possibility offered by each one service solution.

The second type of users (CDE Users) are those users that can access the services installed and running within an instantiated CDE. Because of these services can be considered completely independent from the FI-CoDE cervices it makes sense also to allow the possibility to set up an independent policy for managing user accounts that access the CDE services. On the other hand when a CDE is created starting from predefined templates or from a selection of services, is possible to define access rights for FI-CoDE users.

Within the UMS it's possible to define groups in order to identify accounts that, for example, are related to organizations or projects.

CDE Manager



FI-CoDE - CDE Manager

The FI-CoDE platform allows to create and manage private Collaboration Development Environments (actually virtual resources) to support, first the development and testing of FIApps and GEIs and second to bring them in production.

To fulfil these goals, and to provide the most flexible solution, every CDE can be created accordingly to three different options based on:

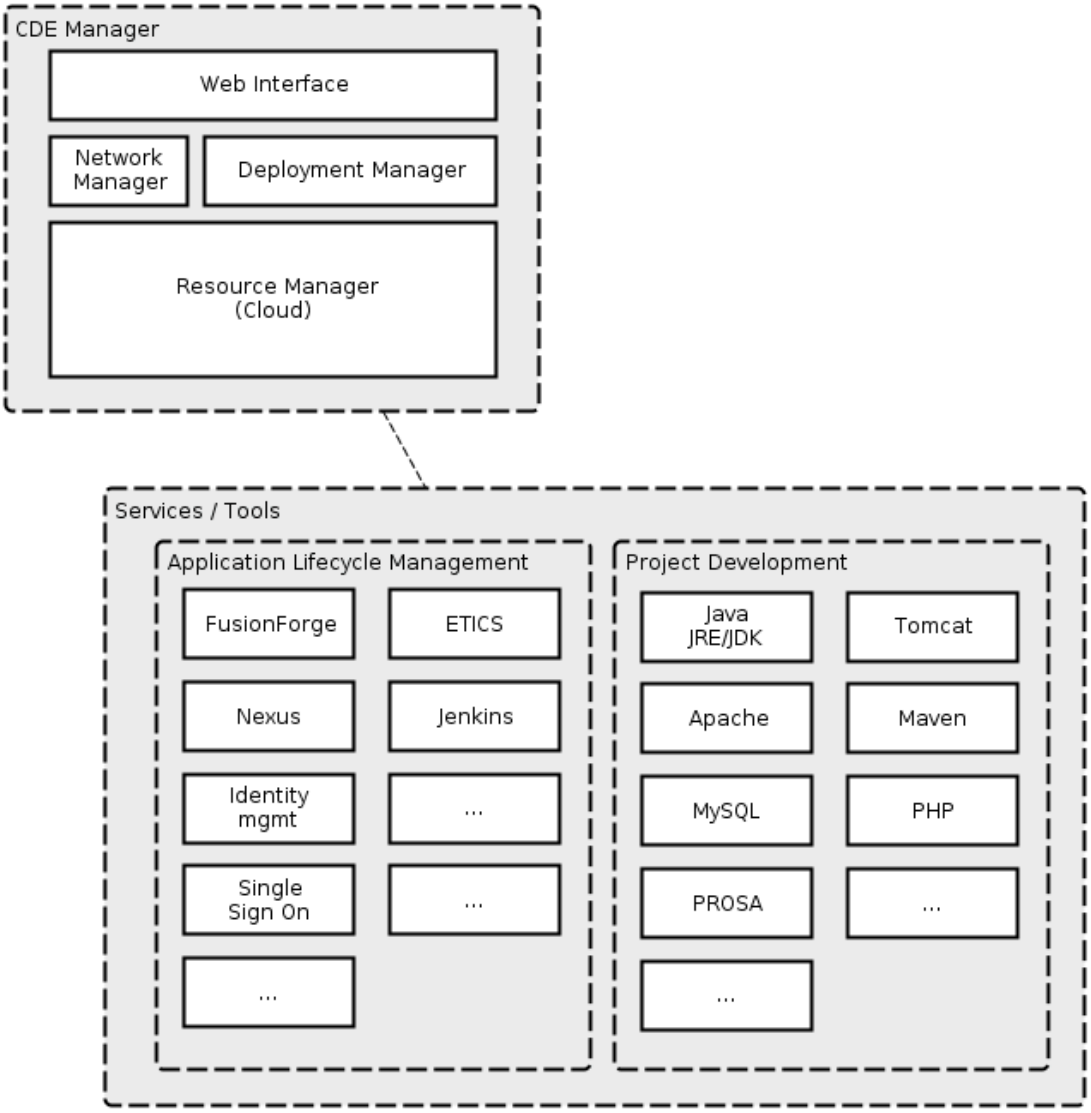
- **templates:** the user can select from a list of predefined environments the one that better fit the project needs. This list is composed by templates shared among all FI-CoDE registered users and private user-defined templates. Once the environment has been created it's possible to further customize it.
- **services:** The user selects what services to include on the CDE from the list of available services. Some default values defined for the selected services may be changed to better fit the project needs (e.g. DB/disk space). Once the environment has been created it's possible to further customize it.
- **empty environment:** The user defines the virtual resources needed by the project and once it's ready, the user proceeds to install the services. In this case the user is able to build from scratch the whole CDE.

To give a more concrete example of how a CDE may looks like here is considered a

project that intend to organize the work with two sub-environments:

- **Application Lifecycle Management:** here are contained all those services strictly adopted to manage the project along all the phases (e.g. requirements definition, tasks definition, progress tracking, user identity manager) from the concept to the release.
- **Project Development:** here are contained all those tools and assets strictly adopted during (or are relate to) the development activities.

Next figure summarize this example, also reporting some possible services for each one of the two environments.



FI-CoDE - CDE Services and Tools

Forge

To provide a more detailed example of one of the services made available for the Application Lifecycle Management environment, here is considered the Forge solution.

The Forge is the tool in charge of providing an integrated Collaborative Development Environment (CDE) to support the development of a software project. A Forge usually provides in a unique environment the following tools:

- version control system;
- bug tracking system;
- to-do list;
- mailing list;
- document management system;
- forum.

FI-WARE project adopts the FusionForge solution as first reference implementation, to be integrated into the FI-CoDE~~, while other solutions (GitHub, QualiPSo Factory) will be evaluated as additional options~~.

Note: the development of a Forge is not a goal of the Developer Community and Tools activities.

The choice of an open source solution, for a Forge, allows to get in contact with the community of developers and leaders, to understand the code base and, if/when it's the case, to contribute some updates that are driven by the FI-WARE needs and, at the same time, are general enough to be included into the official development line.

It's worth to note that a comprehensive CDE may be provided by a single Forge solution or thanks to the composition of specific tools, each one able to cover one (or more) feature of a CDE. In the following are listed some of these tools that are usually adopted by the people working in the open source environment:

- version control system (e.g. Subversion, Git);
- bug tracking system / to-do list (e.g. Bugzilla, Mantis, Trac, Jira);
- mailing-list (e.g. Mailman);
- document management system (e.g. Alfresco, Nuxeo);
- forum (e.g. JForum, Vanilla).

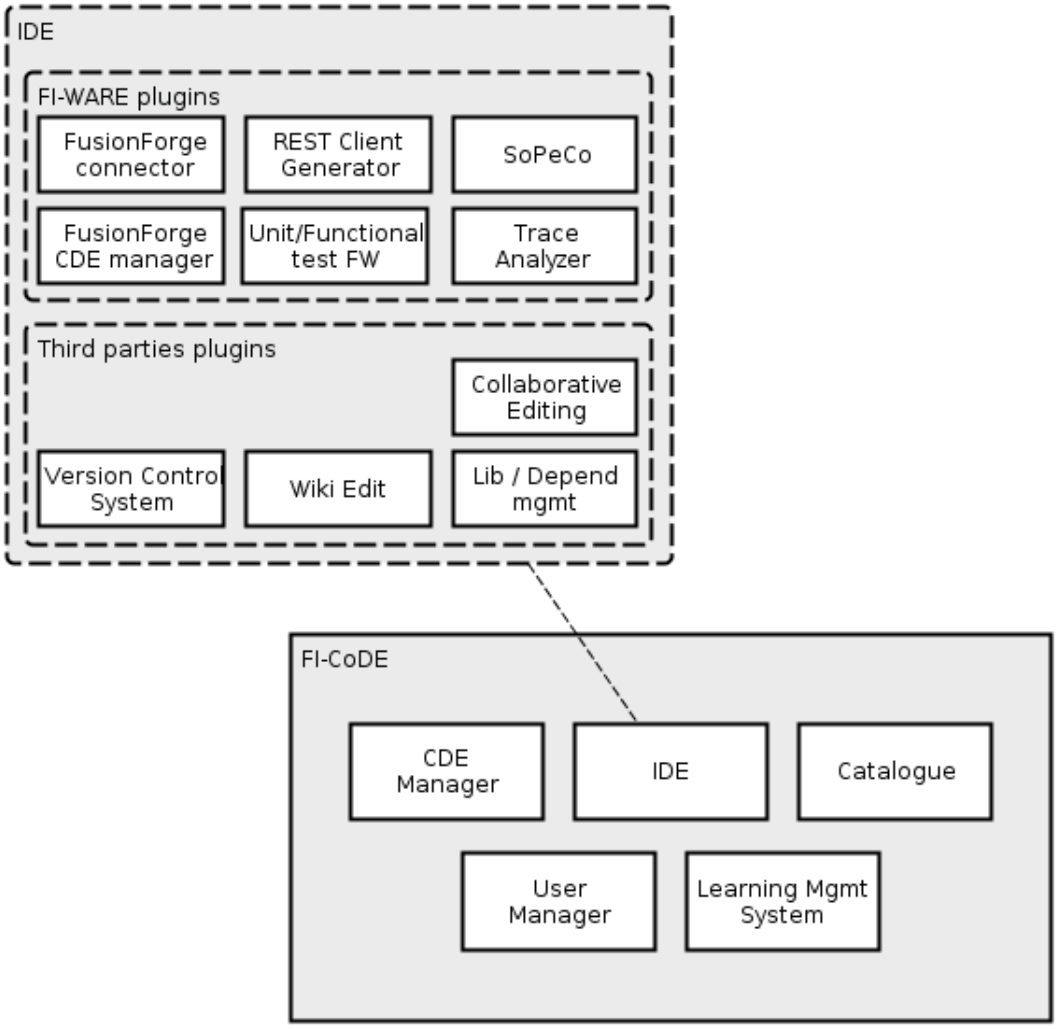
FI-CoDE IDE

The approach to the development of the FI-WARE IDE is based on the idea that the user (developer, architect, project manager, etc) can improve the productivity by having a direct contact (from the IDE) with the other tools that are part of the whole FI-CoDE (e.g. a Forge and related services). The proposed solution is designed to reduce the distance and the possible misalignment between the project manager and the developers that are working together on the same project.

The integration involves:

- Forge services;

- Catalogue;
- Test and Validation tools.



FI-CoDE - IDE

The target users of this IDE are mainly the software developers working on the FI-WARE Compliant Platform Products and the community of software developers that will build Future Internet Applications and Services on top of FI-WARE Instances.

The proposal is to produce a FI-CoDE IDE package that is based on the Eclipse IDE and already contains a set of, available, plug-ins selected according to the partners experience. In addition to these , other plug-ins, tailored to the needs of FI-WARE, are going to be developed and included into the final FI-CoDE IDE bundle.

During the initial configuration phase, the FI-CoDE IDE, will be arranged to work with a specific FI-WARE Instance. The benefit of this choice is to have directly available the list of the Generic Enablers provided by that FI-WARE Instance and be directly bound to the

Forge it eventually provides.

The decision to adopt the Eclipse IDE is because of the assumption of considering Java based solutions and because it has a modular architecture (plug-ins based) as well. An additional and non-technical reason is, of course, its consolidated stability, reputation and community dimension.

The integration of the IDE with the FI-CoDE tools allows the developer to remain better focused on the current task avoiding the continuous switching between different interfaces. On the other side the project manager may have the option to interact, for example, with the CDE features from the IDE and at the same time have a closer view to the developed artifacts.

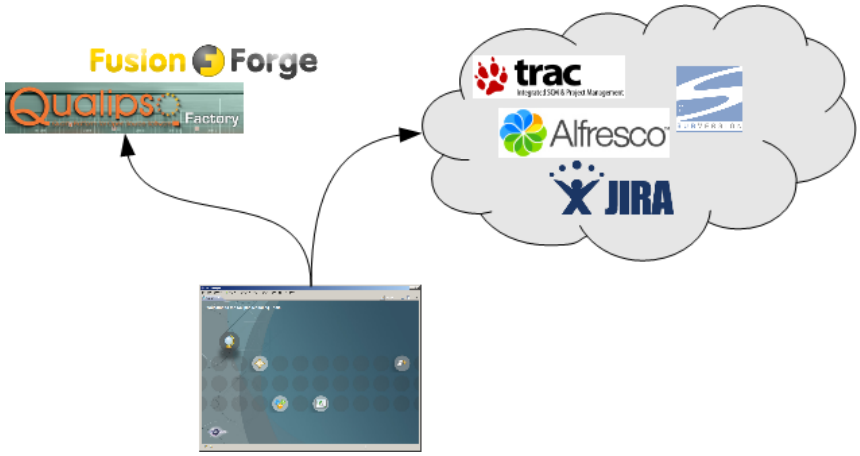
The goal is to provide through the IDE only the basic information and features of the other FI-CoDE tools and not to completely replicate the user interface of these tools inside the IDE. ~~This allows concentrating more on the added value features that will be agreed later on according to the users needs and requirements.~~

Next sub-chapters will explain in more details the characteristics of the integration between the IDE and the most interesting Forge services.

IDE-Forge: FI-WARE MetaForge

This section explains the proposed integration between the IDE and the Forge services.

The IDE will connect to the Forge features whether they are provided by as a single forge solution or as a composition of specialized tools (see Figure: CDE composition).



CDE composition

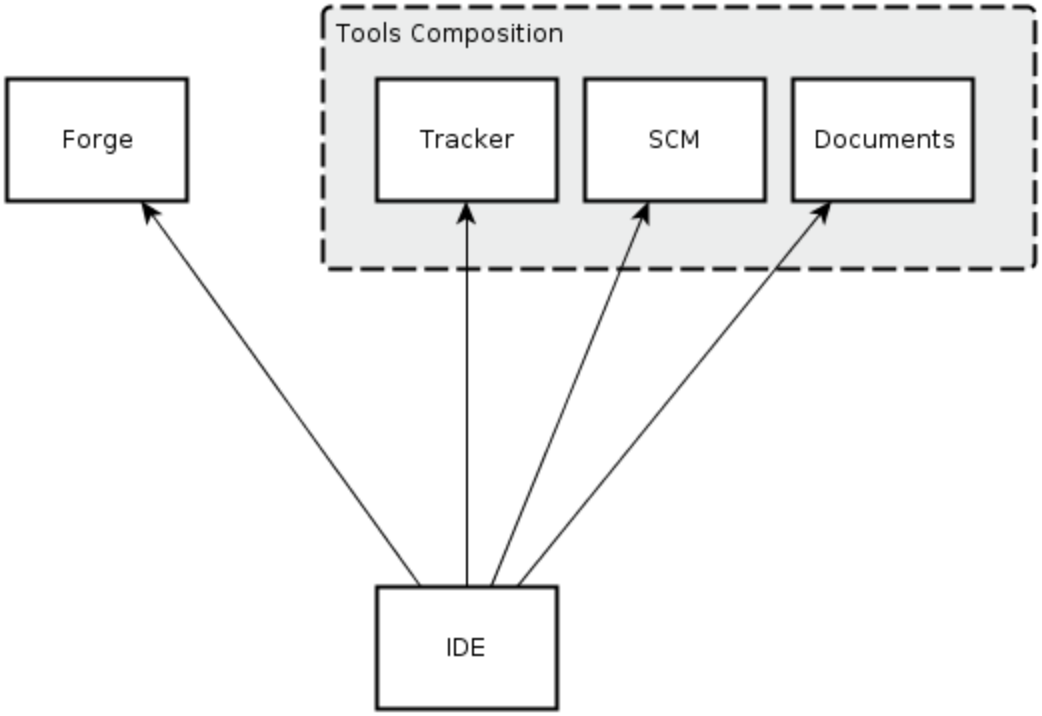
Main features provided by the integration (In Italic, the functionalities that are completely or partially provided by third party tools - e.g. Mylyn, Eclipse Plugin, etc. - and assessed during the development phases):

- *submit a new bug*
- *submit a new task*
- *associate a file to a bug/task*
- ~~cross-search: this allows to query different tools, for the same terms, and present~~

~~the results in an homogeneous way.~~

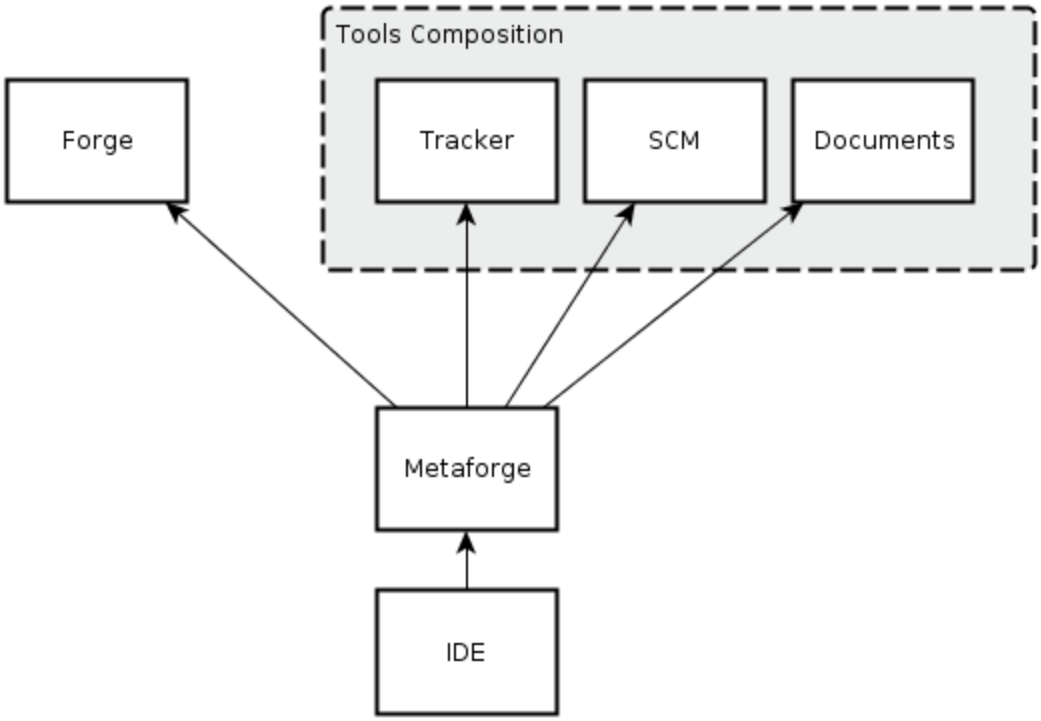
- contextual menu that, from the IDE, lets the user, starting from an highlighted text (code, docs, etc) to add a new task/ticket.
- ~~relationship management: (to be detailed after the architectural approval)~~
  - ~~create relations between resource items~~
  - ~~display relations between the resources in a mind-map style~~

The next figure (IDE - CDE interaction) highlights the two different types (Forge vs Tools Composition) of CDE that are envisaged for the integrated architecture:



IDE - CDE interaction

In order to make the solution as more flexible as possible in terms of variety of forges/tools that can be supported by the IDE integration, an additional layer is inserted between the IDE and the tools. This layer, the FI-WARE MetaForge (Figure: IDE - MetaForge - CDE), is designed to operate as an abstraction layer that, from one side, exposes the functionalities and the data using the IDE user interface, while, on the other side, it interacts with the actual installation of the tools.This approach lets the IDE user interface and the functional logic to be independent from the tools actually installed in back-end.



IDE - MetaForge - CDE

The MetaForge component is divided in three main parts:

- Data Layer
- Functional Layer
- Driver Layer

Data Layer

All the data (or resources in general) handled by the MetaForge are defined in terms of classes (task, bug, document, etc) and relative attributes.

Functional Layer

This layer implements the functionalities made available by the MetaForge, based on the resources modeled by the Data Layer. Every functionality may replicate the respective one available from the actual tool (create, modify, delete, etc.) or may be a composition of those, also involving different tool types (e.g. cross search).

This layer implements the functionalities made available by the MetaForge, based on the resources modeled by the Data Layer. Every functionality may replicate the respective one available from the actual tool (create, modify, delete, etc.) or may be a composition of those, also involving different tool types (e.g. cross search) This layer is mainly responsible to host:

- the services APIs;
- the functional processes (as service composition)

Driver Layer

The real interaction between the IDE and the tools is implemented in a driver logic style. For instance, it can be seen as the one defined for the database systems:

- database server - collaborative tool
- database driver - tool driver
- jdbc APIs - MetaForge APIs

The execution of the operations requested by the MetaForge is implemented at the level of the tool driver.

This structure allows to select the desired forge, or set of tools, with the only constraint to have available (or develop) the implementation of the MetaForge APIs.

Main benefits obtained by this integration:

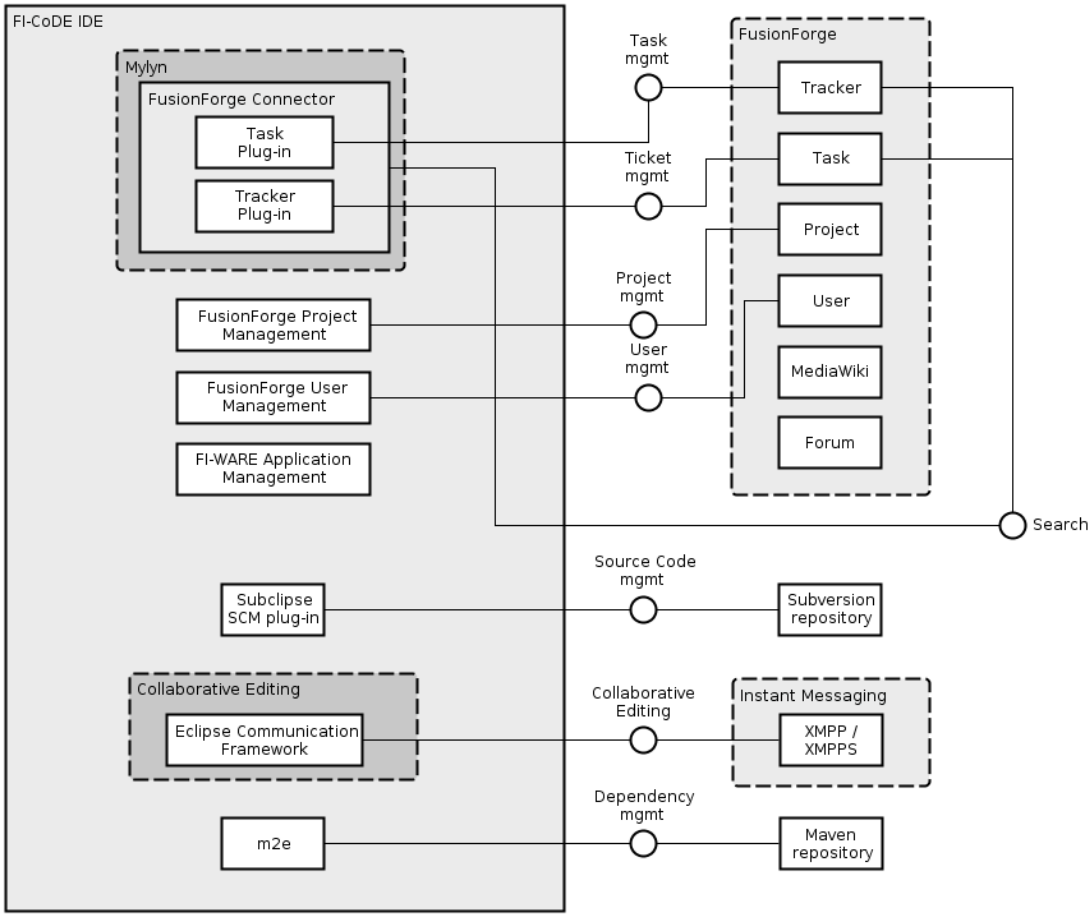
- same IDE user interface independently from the adopted forge solution (as bundle or as composition of tools), the IDE integration allows to manage the information using a common and uniform user interface;
- integrate different forges and tools only associating and configuring the driver implementation for that specific forge or tool; a tool can be integrated only providing the implementation of the required APIs.
- the focus is more on the resources and the processes that manage them, than on the tools actually installed.

IDE-CDE

This chapter describes the components that support the developer in the interaction between the Collaborative Development Environment (CDE) and the IDE. The detailed figure (IDE-CDE) represents the components that are developed by FI-WARE, or third parties and integrated in a single IDE solution.

The selected reference implementation for the IDE is the **Eclipse IDE for Java EE Developers** while the CDE is composed by **FusionForge 5.x** and relative **MediaWiki** extension.

Note: FusionForge exposes its services by means of SOAP interfaces; those interfaces that were not available by default, they've been implemented and contributed to the main FusionForge source code base.



IDE - CDE Architecture details

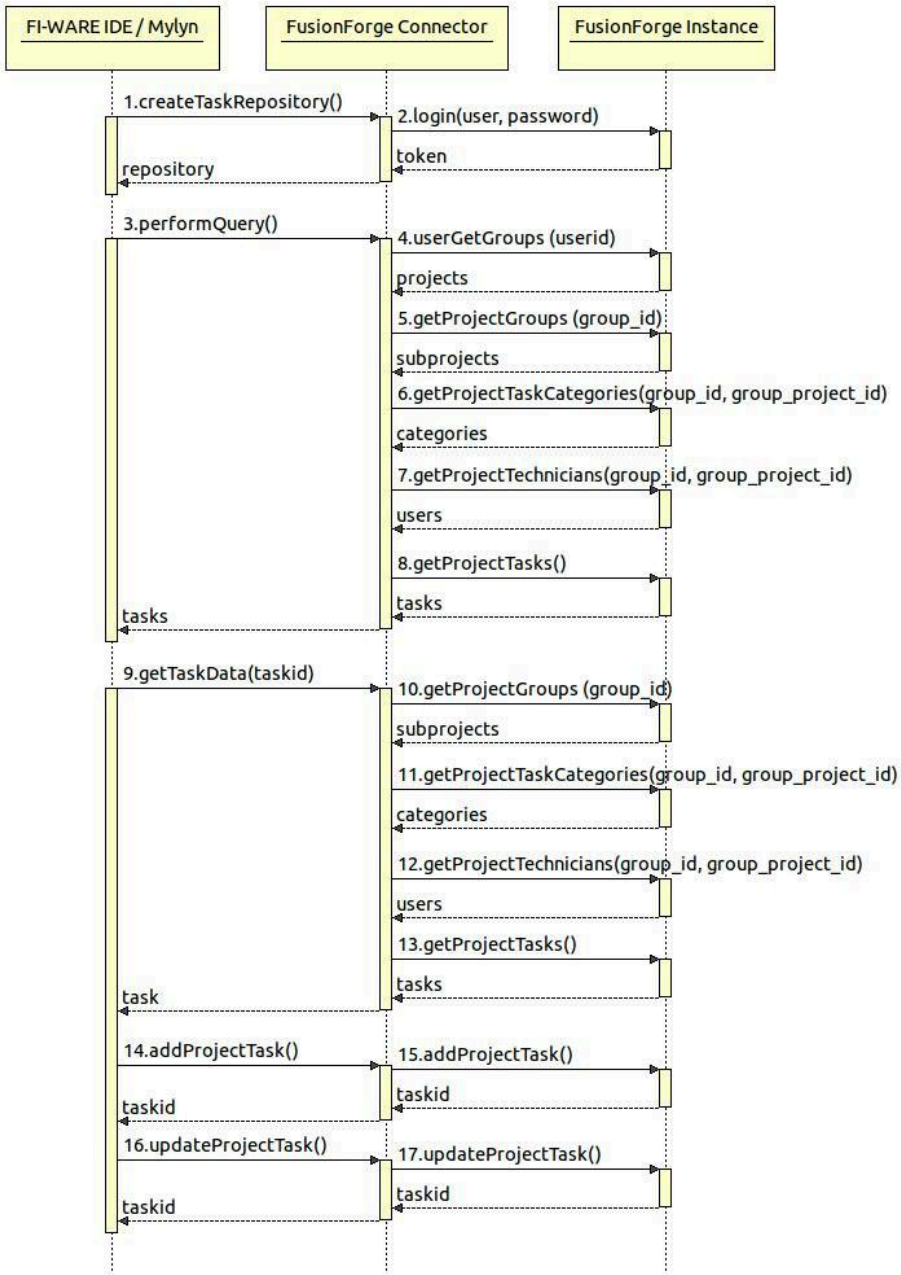
The IDE components represented in the previous figure "IDE - CDE Architecture details" have been developed in the context of the FI-WARE project except those indicated as provided by third parties. In those cases the plug-in has been selected from the leading solutions, tested, configured and integrated into the final environment in the proper way and according to the requirements.

FusionForge Connector

this component allows the developer to connect the IDE to a specific FusionForge instance in order to manage Tickets (Traker) and Tasks. It's based on the Mylyn infrastructure and the main supported features are: create a custom query to list the elements (tickets or taks); show the details of the selected element; insert and update elements. The connection with the FusionForge server is done by means of SOAP interfaces.

The next sequence diagram presents the interactions for Tickets/Tasks management. The one for the Tickets will differ from the one for the Task for those interactions that

retrieve specific attributes and fields (e.g. custom fields) that are not defined for the Task element.



Tickets/Tasks management interaction

this plug-in allow the developer to connect with the Source Code Management system directly from the IDE, and perform the usual operations (e.g. commit, update, merge). Thanks to the Mylyn extension it's possible to reflect into the commit message some information taken from the current task/ticket under execution. This plug-in is provided by third parties and all the additional information can be found at the official web site (http://subclipse.tigris.org/).

Eclipse Communication Framework

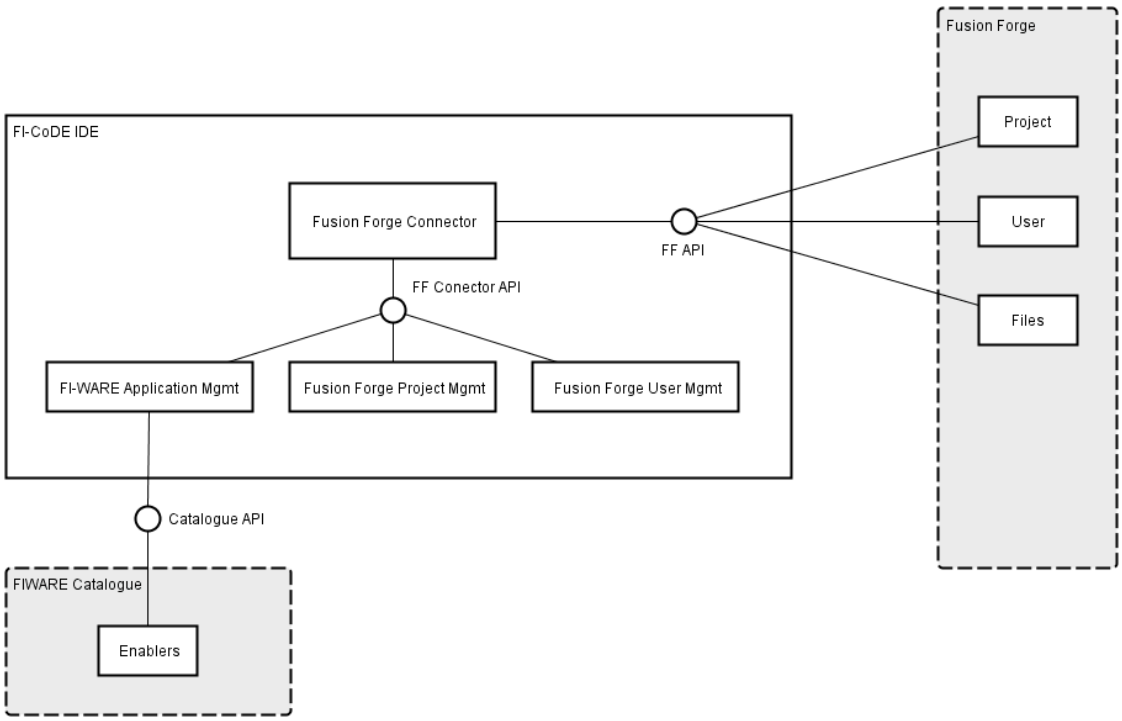
this component has been selected to support the collaborative editing of the same file at the same time by two different developers. This features relies on the communication channel provided by the most common instant messaging services (e.g. XMPP/XMPPS, Google talk). This plug-in is provided by third parties and all the additional information can be found at the official web site (http://www.eclipse.org/ecf).

m2e

this plug-in supports the developer in managing the dependencies (libraries) integrated into the project under development. This plug-in is provided by third parties and all the additional information can be found at the official web site (http://eclipse.org/m2e).

FusionForge User/Project Management

This section describes the IDE plugin that enables developers to manage users and projects of the Collaborative Development Environment (CDE) directly through the IDE. This component depends on the FusionForge connector described in IDE-CDE section, whereby the reference implementation for the IDE described in that section defines the dependencies required by this plugin. FusionForge User/Project Management uses the FusionForge Connector to interact with the SOAP methods exposed by the remote FusionForge instance linked to the IDE by configuration. FusionForge has been extended with new SOAP methods required to manage both users and projects.



FusionForge Project & User Management plugin architecture.

FusionForge Project ~~Wizard~~ Management

The developer can create, update and delete a project hosted into the FusionForge instance directly from the IDE. The developer can select the FusionForge instance where to create a new project and customize the project features, such as tools, roles, users granted to get access to the project and their roles. The connection with the FusionForge server is done ~~by means of SOAP interfaces~~ through the FF Connector.

FusionForge User ~~Wizard~~ Management

The developer can create~~, update~~ and delete a user account into the FusionForge instance directly from the IDE. The connection with the FusionForge server ~~is done by means of SOAP interfaces~~ through the FF Connector.

~~Eclipse Project Type Wizard~~ FI-WARE Application Management

This Eclipse ~~extension~~ plugin adds ~~to the predefined project types list the options to start with~~ to the predefined list of new project wizards a new one that enables the creation of a new FI-WARE Application or a new Generic Enabler implementation. This wizard enables developers to browse and select among available FI-WARE Catalogue Enablers and configure their services within the FI-WARE Application project. Other specific meta-data that are stored together with the project itself and are used by other CDT features (e.g. deployment). This plugin also enables to link FI-WARE projects and FusionForge projects, so FI-WARE project artifacts, such as deployment bundles (or application releases) can be delivered to the FusionForge project.

IDE-Catalogue

The FI-Code IDE is able to access the catalogue to gather information regarding the available FI-WARE Generic Enablers specifications and the available instances.

The Catalogue offers a REST API to perform CRUD (Create, Read, Update and Delete) operation on two main resources:

- Generic Enablers
- Enabler Instances

The response for a request on each resource contains the information stored in the catalogue including: description, contact information of the maintainers, available documentation, etc.

The authorization and authentication will be integrated with FI-WARE identity manager.

IDE-API integration: REST Client Generator

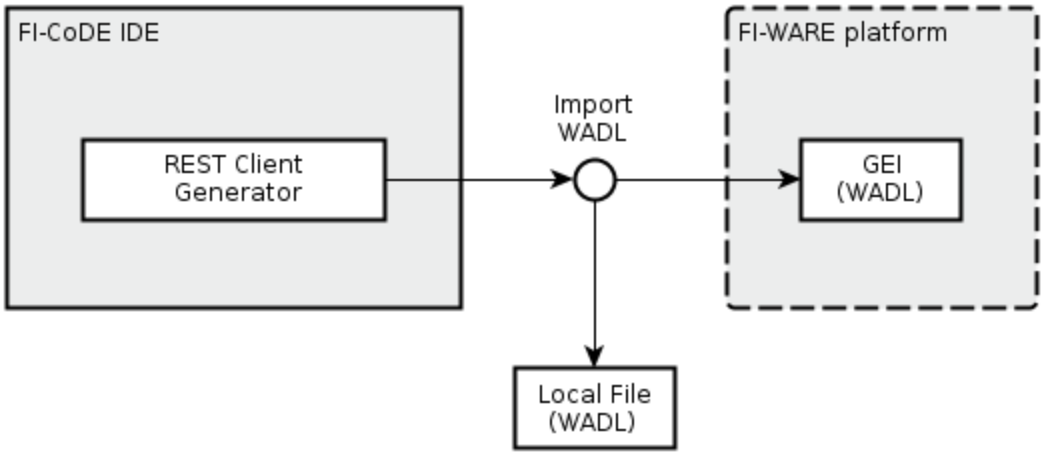
The FI-CoDE IDE supports the access to the FI-WARE Compliant Platform Products that are at the base of the development. The integration of these products into the Future Internet Applications and Services is done by means of a GE Client software component (e.g. client, driver, stub) that has to be compliant to the GE Open Specifications of the selected GEIs. This GE Client is:

- programming language dependent;
- the same for:
    - different instances of the same GEI;
    - different implementations of the same GEOS:
- open for additional customization needed by the specific project.

The GE Client package includes whatever needed by the developer to complete an entire process of development, test and deploy (e.g. dependencies, libraries, configuration files).

When the GE Client is not made available by the GE Provider, the FI-CoDE allows to generate one under some conditions:

- the generated client is for Java programming language;
    - it's based on Jersey library;
- the GEI exposes RESTful APIs
- the generation process requires the availability of the WADL file directly from the GEI endpoint (URI) or from a local file.



REST Client Generator

Learning Management System

The FI-CoDE platform provides to the end users an access to the eLearning area. In this area it's possible to browse take benefit from all the learning contents about FI-WARE, Generic Enablers and FI-CoDE itself. The paradigm of interaction with this eLearning offer is structured in two main channels:

- **recorded lessons:** or Learning Objects (LO), that are always available without any constraint in terms of fixed dates and number of attendees. They are built starting from scratch or from already available presentations (or other documentation) enriched by an audio track. These modules are intended to be SCORM compliant in order to allow the traceability of content usage for monitoring and reporting. The content of these LOs is related to FI-WARE project, GE usage and integration, FI-CoDE platform. The access to the Learning Objects can be made available directly from the FI-CoDE web interface, the Catalogue and the FI-WARE portal;
- **live sessions (webinars):** these are the more traditional synchronous sessions. They require a fixed date, the availability of a presenter and may be limited to a maximum number of attendees. All the scheduled live sessions are published in a calendar together with a brief description of the contents and the detail for connecting to the session. Apart from the typical webinar campaign, an alternative approach is to schedule a live session on demand after an explicit request for additional details, in respect to what is presented in a recorded lesson.

The Learning Management System allows to specify the policy of access for every published content, this way it's possible to have public contents, or restricted contents available to registered users only.

The main entry point for the LMS is the FI-CoDE platform, while other useful places are

the FI-WARE Catalogue and the FI-WARE main web site.

FI-WARE Catalogue

The catalogue is a container for assets published by asset providers and made available to developers that can browse the catalogue to find the specific assets they are interested in and download or use those assets. It provides features for asset publishing, browsing of assets, community for discussions around these assets and references to the appropriate support functionalities available for each asset.

Catalogue deployment view

~~There will be a need for two different instance types of the catalogue. First there is the instance catalogue, of which there will be one for each FI-WARE instance. The first such will be the catalogue for the FI-WARE testbed. This instance catalogue will contain:~~

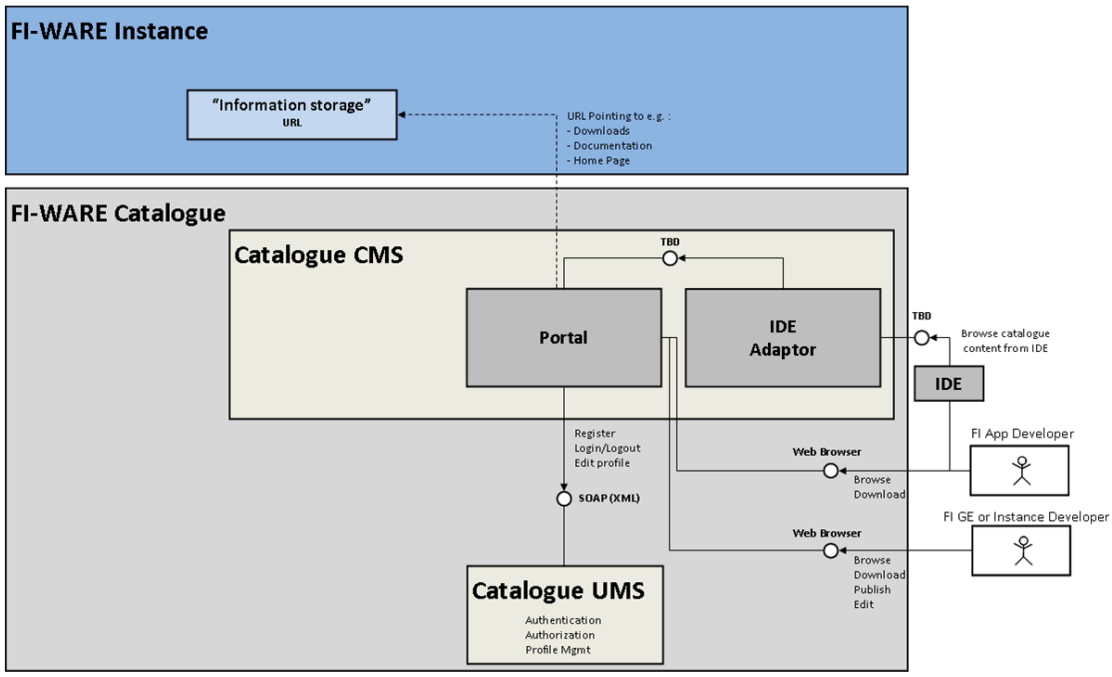
- ~~GE open specifications~~
- ~~FI-WARE compliant platform products (including clients where available)~~
- ~~Instance specific enablers and clients~~

~~Secondly there will be the central FI-WARE catalogue, which will contain the same GE open specifications as the instance catalogues. This catalogue instance type will be populated by each of the instance catalogues, such that it has a listing or database of each GE open specification that has been uploaded to an instance catalogue as well as references to the instance catalogues where the GE implementation can be found. As new generic enablers are added to an instance catalogue they will automatically be added to the central catalogue as well. Instance specific enablers are not automatically added to the central catalogue. For the initial deployment in the FI-WARE testbed we will most likely combine the two instance types above, and make the FI-WARE testbed catalogue act as both the central repository for GEs and as the testbed instance catalogue.~~

The catalogue will be deployed as part of the FI-WARE Testbed and expose the assets available in this testbed. The catalogue enables users (such as e.g. FIApp or GE implementations developers) to browse and interact with content as well as with FI-Ware GE and instances providers. The users can, in addition, also publish new GE implementations, info about GE Implementations and edit them. The catalogue is based on two main components:

- Catalogue Content Management System (CMS) - a portal including an adaptor for integrating with an IDE
- Catalogue User Management System (UMS)

The CMS/portal includes a publishing system and stores published content making it available for browsing. The IDE adaptor provides an interface to the IDE that enables browsing of catalogue content. The UMS takes care of registration, authentication, authorization and profile management. It is role based and it is possible to configure different roles and provide them different rights to the catalogue.

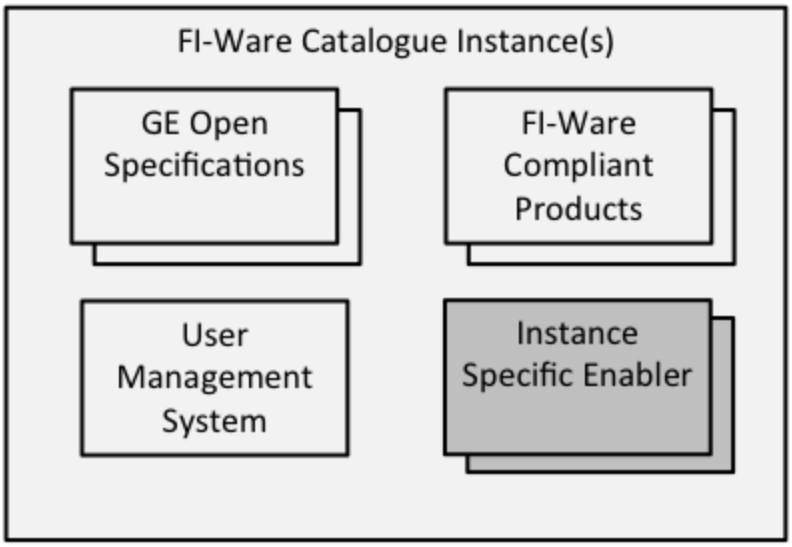


Catalogue Architecture

There will be a single instance type for the catalogue that will be present on each FI-WARE instance. The catalogue for each instance will contain:

- GE open specifications (or reference to)
- FI-WARE compliant platform products (including clients where available)
- Instance specific enablers and clients

Where the first two will be shared among all the catalogue instance while the third will be separately populated by every FI-WARE instance.



Catalogue Instances

Proposed solution

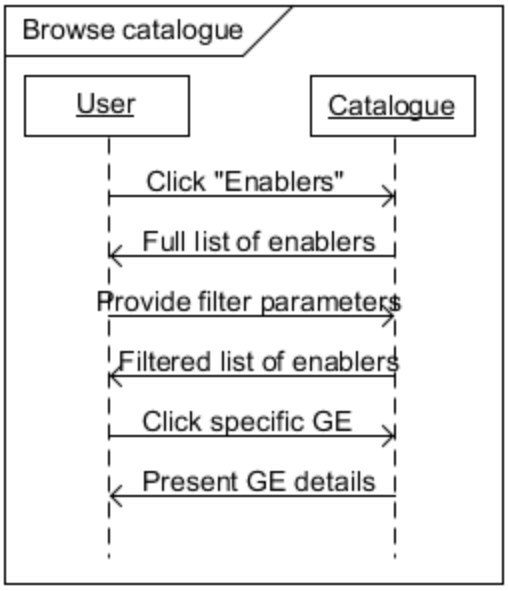
The Catalogue architecture is based on the existing Ericsson Labs developer community and the experience from running that. However there will be some modifications made to provide a better fit to the needs of the FI-WARE Catalogue. The core of this solution is the Drupal CMS with extensions and adaptations made to better enable the specific functionality required for this system.

Catalogue functionality

The following chapters provide more details on distinct parts of the Catalogue functionality, including the asset publishing, community functions and user management.

Browsing the catalogue

The next diagram shows an example sequence of invocations necessary when a user browses the catalogue:



Browsing the catalogue

Asset Publishing

The Catalogue environment is in charge of publishing and making available the various assets that can be used to build FI applications and new assets. ~~More specifically:~~ The catalogue will be configurable but it will by default contain the following basic information for any FI-WARE instance (e.g. FI-WARE Testbed):

- Generic Enabler Open Specifications
- Downloadable files (e.g. GE implementation or reference to the API and related clients)
- Support Documentation
- Links to the FI-WARE instances where the GE Implementation instance is available (e.g. the FI-WARE Testbed)
- Community - blog and forum (optional)
- ~~FI-WARE compliant platform products~~
- ~~Documentation~~
    - ~~FAQ~~
    - ~~Terms of use for:~~
        - ~~Generic Enabler Open Specifications~~
        - ~~FI-WARE Compliant Platform Products~~
        - ~~Test clients~~

This means that the FI-WARE Catalogue contains both GE Open Specifications as well as references to the FI-WARE instances where an implementation of those GE Open Specifications can be found (i.e. the FI-WARE Testbed).

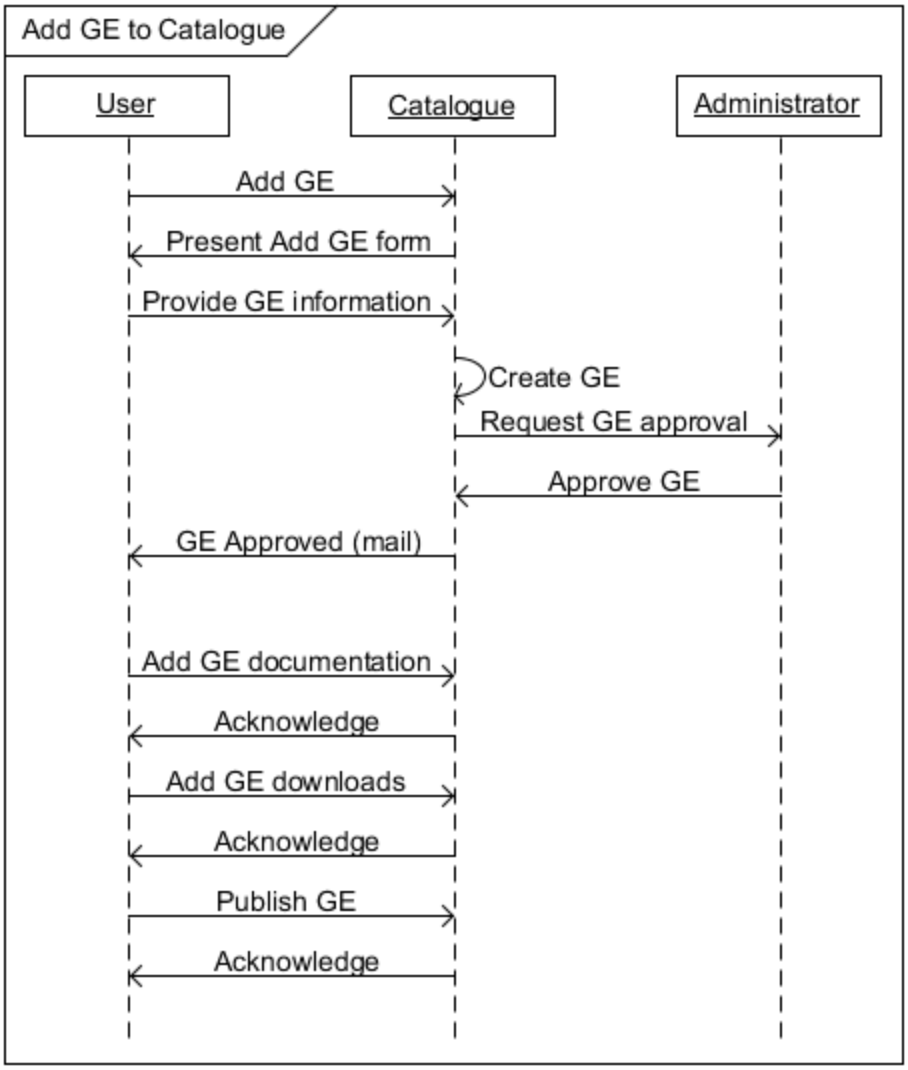
~~The FI-WARE Catalogue contains open specifications as well as references to the instance catalogues where an implementation of those specifications can be found. Publishing in the catalogue is done in the instance catalogue (as opposed to the central catalogue), and can be done either through the IDE or directly in the catalogue using web-based forms. All projects who wish to publish into the Catalogue need to align to specified requirements on:~~

Publishing is done directly in the catalogue using web-based forms. There will be two different publishing flows: one for adding new Generic Enablers (specifications, documentations, etc.) and one for adding references to FI-WARE instances. Whoever wishes to publish into the Catalogue needs to align to specified requirements on:

- Technical readiness (test&verification; …)
- Documentation
- Categorization and keywords to enable search

The publishing process includes verifying that requirements are met, for this reason, the publishing of any GE implementation requires approval by an administrator. The publication of a reference to a FI-WARE instance is instead automatic and the reasoning is that the community can itself take care of quality assurance of instances: they will simply not use instances that are not working well. ~~The process can be more or less automated; this will be specified in more detail later on.~~ The FI-WARE Catalogue access policy is defined in order to regulate the access to the artefacts and related resources. Parts of the Catalogue will be visible to anyone, while parts (typically downloads and API key requests) are only available to registered users. The Catalogue will contain a document specifying the rules for publishing and access.

The following diagram presents all the sequence of methods invocation necessary to upload into the FI-WARE catalogue all the information about a GE: open specifications, implementation, documentation, etc.



Adding a GE to the catalogue

Support

There is a back-end tracker system to handle support requests, which distributes requests to the right GE provider. This is not considered a part of the catalogue and will have to be handled by separate external request tracker system. The catalogue will link to this system.

Developer Community

The catalogue also includes maintaining, monitoring and supporting the community of

developers who use, develop or are just generally interested in the assets available in the Catalogue and/or the FI-WARE Instance as such. Keeping the community ‘alive’ and active is necessary in order to attract developers and drive usage. It cannot be left only up to the community itself, but some moderator activity needs to be continuously on-going. Developer community maintenance is largely not a technical issue, and so will be described in more detail in the methodology supporting the adoption of the FI-WARE SDK, here is defined only the technical tools to support it.

- News and blogs
- Developer fora/discussions
- Subscriptions

The catalogue will include linking to social media channels, to allow developers and the community responsible to share, promote and provide feedback.

User Management

The catalogue itself requires some kind of user management. For this we propose to use a central user management system that is based on Ericsson Developer Account, the user management system used for Ericsson Labs. The advantage of this central user management system that will be bundled with the central FI-WARE Catalogue is that it will provide single sign on functionality for users as they go between different catalogue instances as they browse. One typical use case is for a user to start out in the central catalogue, find a GE that he is interested in and then follow the links to the different FI-WARE Instance Catalogues where the GE is implemented to find an instance that seems interesting to work with. Having to register and log in to each separate instance would make the process much more complicated for the user. However, for the case where an FI-WARE instance provider wants to set up his own user management separate from that provided by the central FI-WARE Catalogue this is also supported. As the Drupal CMS is used as the foundation for the catalogue it is easy for the specific implementation to either pick another third party user management solution or to use the built-in Drupal user management.

Optional catalogue components

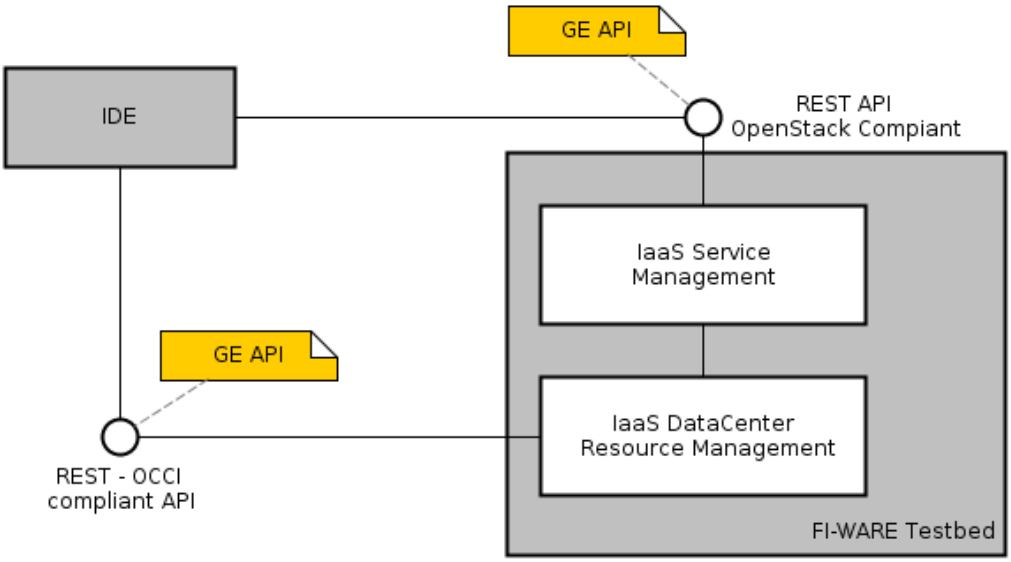
This chapter contains optional catalogue components that have been discussed as features that we may or may not want to include as a part of the catalogue deliverable. As no firm decision has been made yet the components are included here as a reference, and once decision has been made the section will either be included in the previous sections or deleted from the document.

API key handling

The API key system as used in Ericsson Labs can be used by an API developer to allocate a key to each user of the API and provides facilities to limit the usage of the API to a certain number of requests or a certain request rate per API key. This is a service

that can be provided by the Catalogue. It is optional for API developers to use the service or choose one of their own (this can be regulated in the publishing rules by the FI-WARE Instance owner). This would provide asset providers with a tool with which they can limit or throttle the access to their API. The reasons for doing this could include managing server load, blocking users that abuse the service or providing premium (paid for) services to high load users.

~~Deployment Tool - TO REVIEW~~



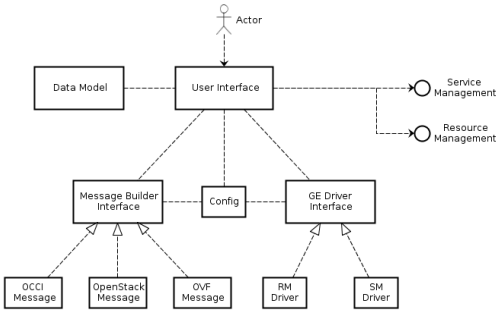
~~Deployment Architecture Overview~~

- ~~IaaS DataCenter Resource Management uses OpenStack as back-end solution~~
- ~~exposes a REST and OCCI compliant API~~

- ~~IaaS Service Management~~
- ~~uses OpenStack as back-end solution~~
- ~~exposes OpenStack API~~
  - ~~this API will be the one defined as GE API~~
  - ~~the full implementation will be ready by Sept/Oct 2012~~

- ~~IDE~~
- ~~interacts with RM for single VM mgmt~~
- ~~interacts with SM for multiple VM mgmt~~

IDE Model

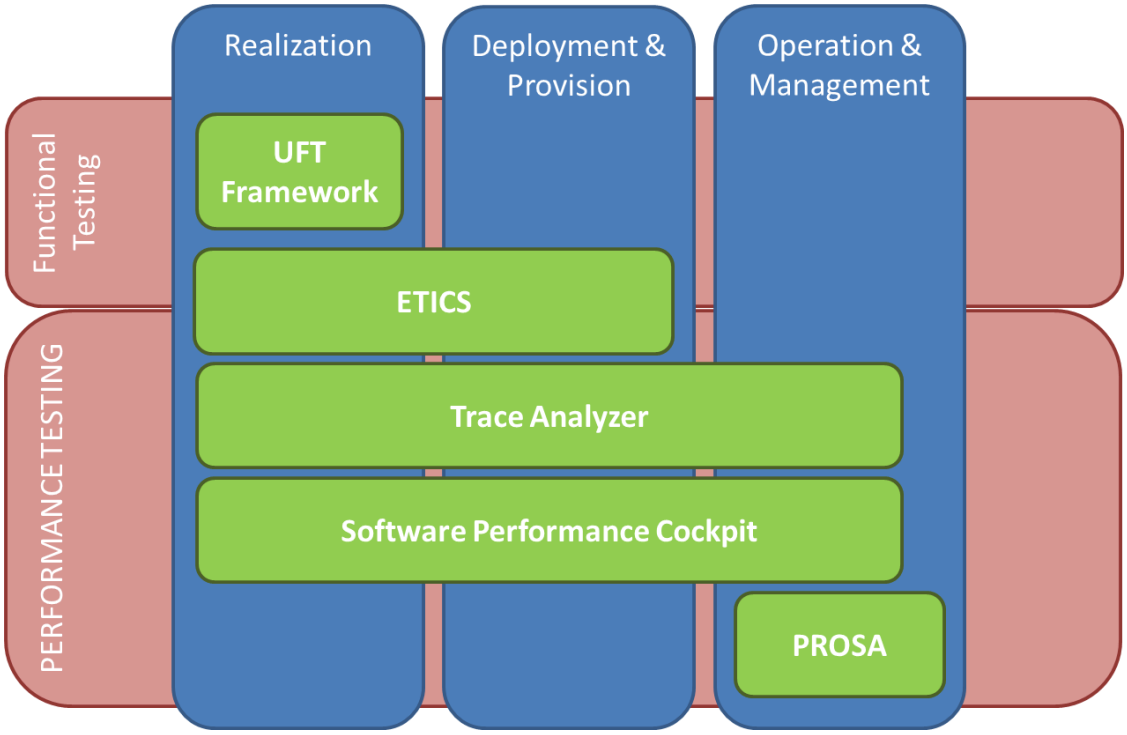


Architecture Overview

- **User Interface**the user interfaces, wizards and Eclipse views, used by the Actor to interact with the Cloud Hosting GEs. They use the *Message Builder Interface* and *GE Driver Interface* to interact with *SM* and *RM* end points.
- **Data Model**
  - the component that stores the structured information managed by the user and exchanges with the GEs
- **Config**
  - this part maintain the configuration that defines if the User Interface is interacting with a SM or a RM and the format of the exchanged messages.
- **Message Builder Interface**
  - interface used to translate the information from (and to) the *Data Model* to (and form) the appropriate format defined in the *Config* component (may depend also on the GE Driver implementation). The implementations of this interface cover different formats such as OCCI, OpenStack and OVF.
- **GE Driver Interface**
  - interface used to interact, in terms of operations to call, with the GE defined in the *Config* component. The abstract operations defined by the interface are mapped to the specific operations provided by the *SM* and *RM* GEs end points.

Testing and Validation

While we can assume that state-of-the-art testing will be performed on FI-WARE applications under development,t This activity will focus on testing activities that are specific to the nature and architecture of FI-WARE applications and instances. These are characterized by multiple distributed components and services that will not be complete before they are deployed. Thus, testing approaches suited for FI-WARE applications must span different stages of the development process: development time (i.e, realization), deployment time (i.e., deployment & provision), and run time (i.e., operation and management). With these demands, different testing tools technologies can be defined in the context of FI-WARE as shown in Figure: *The Testing Approach*. In order to focus the activities performed by the DCT team, different testing approaches will be considered one after another to ensure substantial results for single approaches.



The Testing Approach

The following assets are provided ~~by the partners~~ for testing:

- **Unit and Functional Testing framework** (UFT framework) is a framework that aims to simplify the creation of tests for validating units of code and/or the functionality of more complex components. UFT framework can be adopted during the development phase of FIApps or GEIs indifferently.
- **Trace Analyzer** is a tool that aids in performance testing during development, deployment and runtime. The **Trace Analyzer** tool helps with collection and analysis of data across different layers (h/w, os, middleware, services). The tool provides visualizations for the collected data, allows filtering and aggregating the collecting data, highlights performance anti-patterns, and identifies bottlenecks in the execution.
- **ETICS** is a service-oriented system to define, execute and analyse a large range of software testing cases, from static code analysis (e.g. documentation coverage, PMD metrics) to unit testing to deployment and functional testing in a multi-node environment created specifically for the test execution.
- The **Software Performance Cockpit** is a tool for the definition and execution of complex performance test scenarios. Furthermore, it comes with rich mechanisms for analysing results by statistical means. Based on input from other project activities/teams and use cases, the tool will be extended to better cover distributed FI-scenarios. The tool can be used during development, deployment and runtime.
- **PROSA** is ~~a result of the EU project S-CUBE~~ a tool for continuously monitoring and visualizing the QoS data of constituent services in a composition during runtime. PROSA can be configured to invoke the service by online testing to

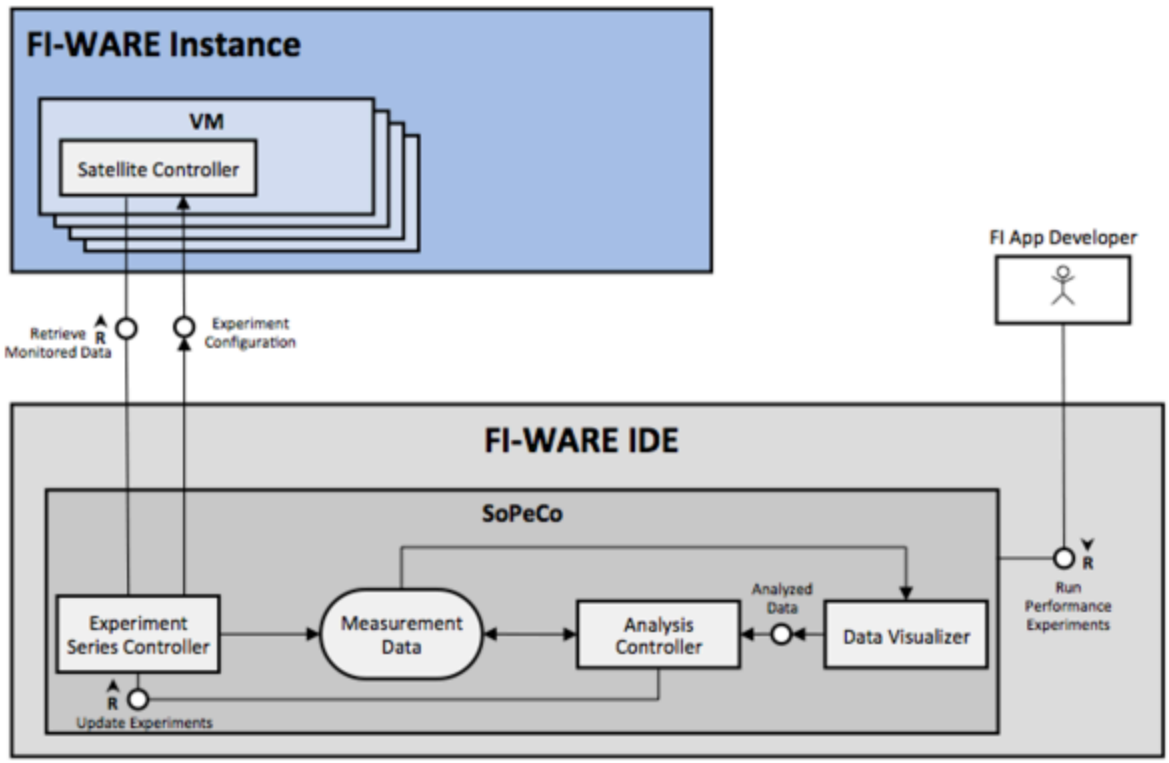
gather additional QoS when needed.~~analyzing the quality of monitoring data gathered at run time and improves it if necessary by invoking online tests.~~

~~Thus the DCT team will focus on performance and QoS analysis and test first.~~ Putting them altogether, the UTF framework can be used to perform unit and functional testing during development. At development time and deployment time, Trace Analyzer will be provided for aggregating, filtering and visualizing monitoring data gathered from testbeds and target environments. This will aid developers in providing applications which use resources in an optimal way. Furthermore, the Performance Cockpit supports ~~the analysis of design decisions but also for~~ planning feasible service level objectives for some given services or components by developing a measurement controller for the initial application (or components of it), which would allow for early feedback during development time. So, in addition to functional aspects, the developer gets early feedback on the non-functional requirements (e.g. performance). At run time, a continuous surveillance of the performance of all distributed services constituting a FI-WARE instance is desirable. ~~The related data can also be handled by~~ This is supported by SoPeCo and in a limited way by Trace Analyzer. In addition, it is necessary to consider the quantity ~~quality~~ of the QoS~~-related~~ data here. This is possible ~~with~~ by ~~S-CUBE~~ PROSA which allows defining ~~requirements for~~ the number of necessary data points, monitors their availability ~~of according data~~, and invokes tests in order to ~~generate~~ collect ~~necessary~~ additional data points through the testbeds or target environments.

Software Performance Cockpit

Main Components

SoPeCo facilitates systematic performance evaluation of software systems. The high-level architecture of SoPeCo, including its four components in the context of FI-WARE, is shown here. The FI App developer can use SoPeCo to define experiment configurations, and then systematically run those experiments, gather the results and use the analysis controller and the data visualizer to view the outcome.



SoPeCo Architecture in FI-WARE

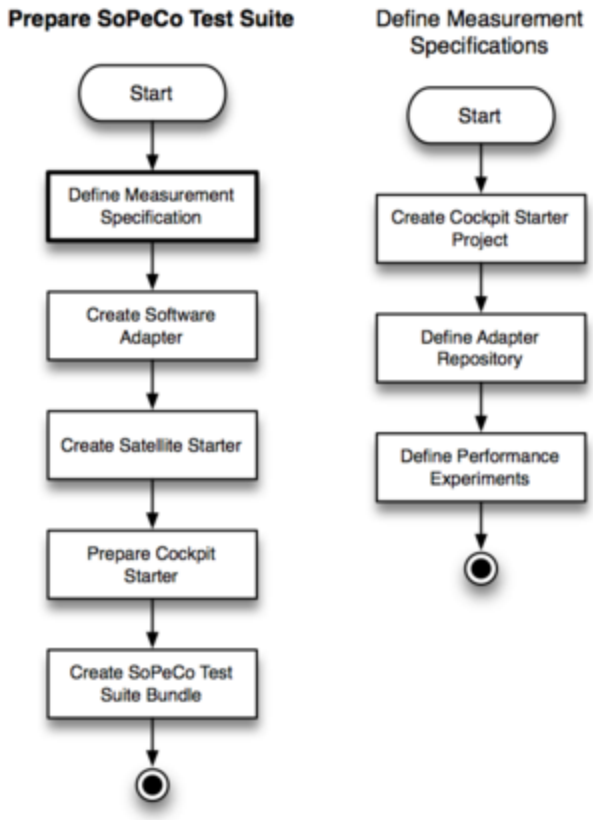
Prepare SoPeCo Test Suites

The process of preparing SoPeCo test suites is summarized in the following two diagrams.

The user has to first define measurement specifications, which further details into defining the interface of the software adapters (in terms of input/out parameters of interest) and the specification of the experiments. For each experiment, a set of input parameters of interest, value variation strategies for each parameter, and the output parameter of interest are identified.

After defining the measurement specifications, a software adapter will be developed that serves as the interface between the target system (software under test) and the SoPeCo framework. Software adapters are called by the framework to run single experiments on the target system. In order to perform test on remote machines, satellite starters are developed that act as a service between the framework and the machine on which the performance is monitored. SoPeCo satellites then pass on the requests to the corresponding software adapters. In simple configurations where one has only one adapter per physical machine, the satellite and the adapter can be combined into a single component.

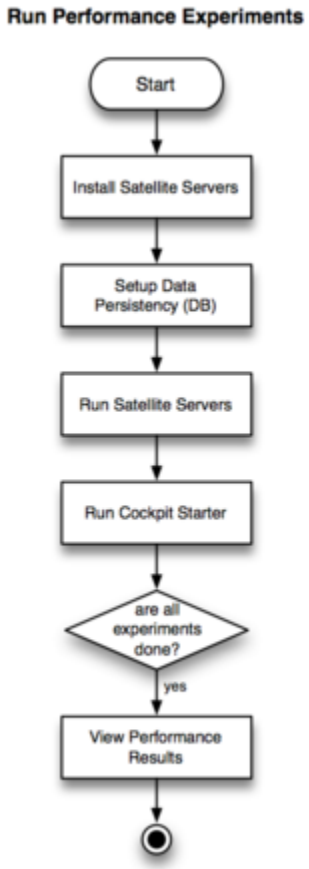
Finally, the developer bundles up these pieces into re-usable performance test packages.



Prepare SoPeCo Test Suites

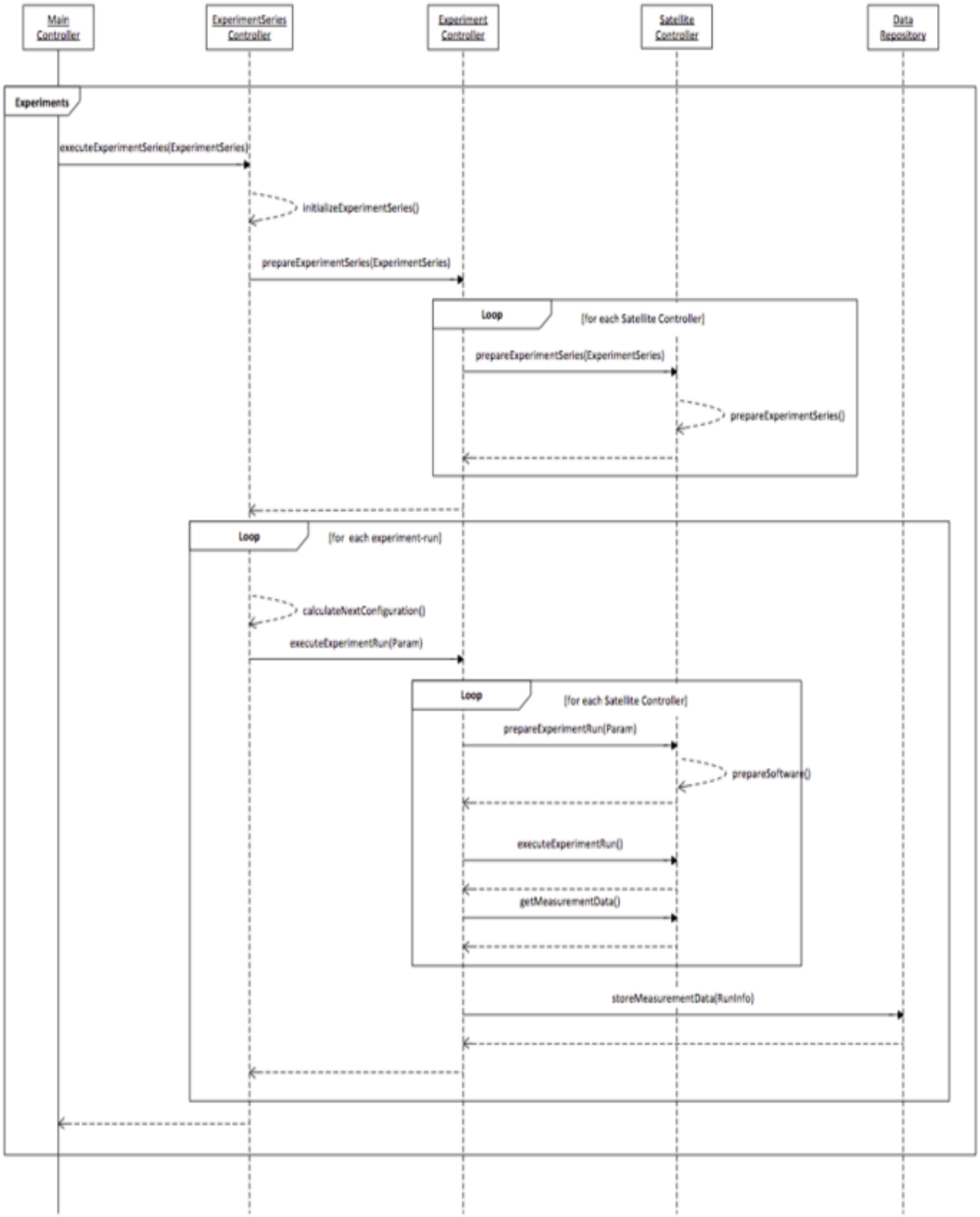
Run Performance Test Suites

In order to run performance tests, satellites have to be installed and running on the target machines. If needed, a database can be set up and the cockpit starter can be configured to use the database for data persistency. Next, the cockpit starter will be executed. It automatically communicates with the satellites and commands the execution of experiments. Once the data is gathered, the data visualizer component can be used to observe and analyze the results.



Run Performance Experiments

To better understand the concept of experiments, the following sequence diagram further explains how the experiment executions are handled in SoPeCo.



Execute Experiments

Trace Analyzer

Trace analyzer plug-in has two major modules: the first one is the Trace Modeler that reads the trace, generates internal model of the trace and provide data on demand. The second one is trace analyzer that visualize the data in the trace

model according to the selected view, or statistical query.

Trace Modeler

- *buildTraceModel(trace)*
- *getViewData(view_identifier)*
- *getRecordDetails(record_id)*
- *getStatistic(parameters)*

Trace Analyzer

- *analyzeTCP(interface, port)*
- *analyzeOS()*
- *analyzePthreads()*
- *selectView from the list: {TCP_sessions, TCP_packets, OS_CPU, OS_mem}*
- *getRecordDetails(record_id)*
- *getStatistic(query)*
- *zoomIn(interval)*
- *zoomOut*

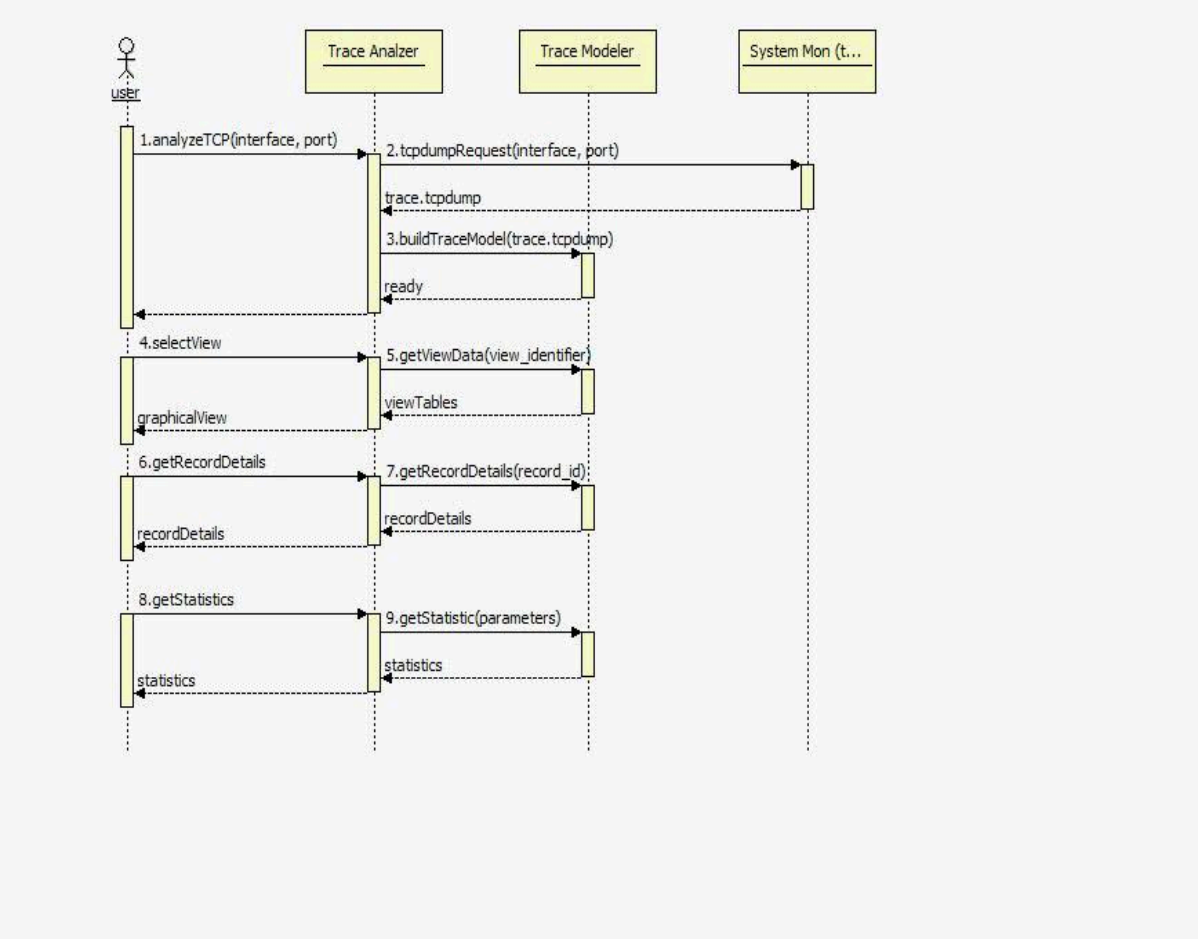
System Monitoring

In order to collect monitoring data Trace analyzer uses the tcpdump and sar tools in the following way:

- tcpdumpReq(interface, portNum) is implemented by
  tcpdump –i <interface> port <portNum>

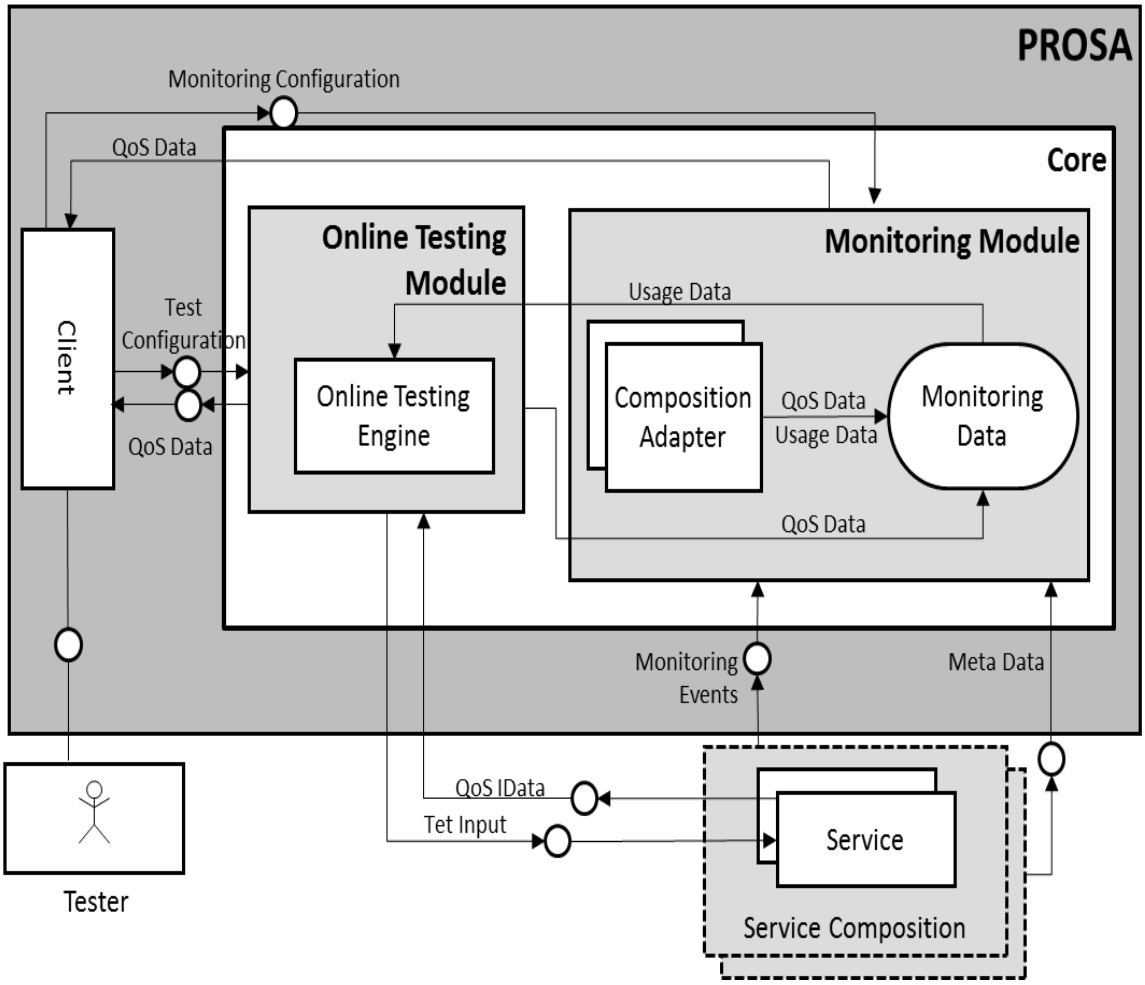
- sarReq(intervals) is implemented by
  sar –A <intervalLen> <intervalNum>

The following sequence diagram describe one of Trace Analyzer usage scenario.



Trace Analyzer - usage scenario sequence diagram

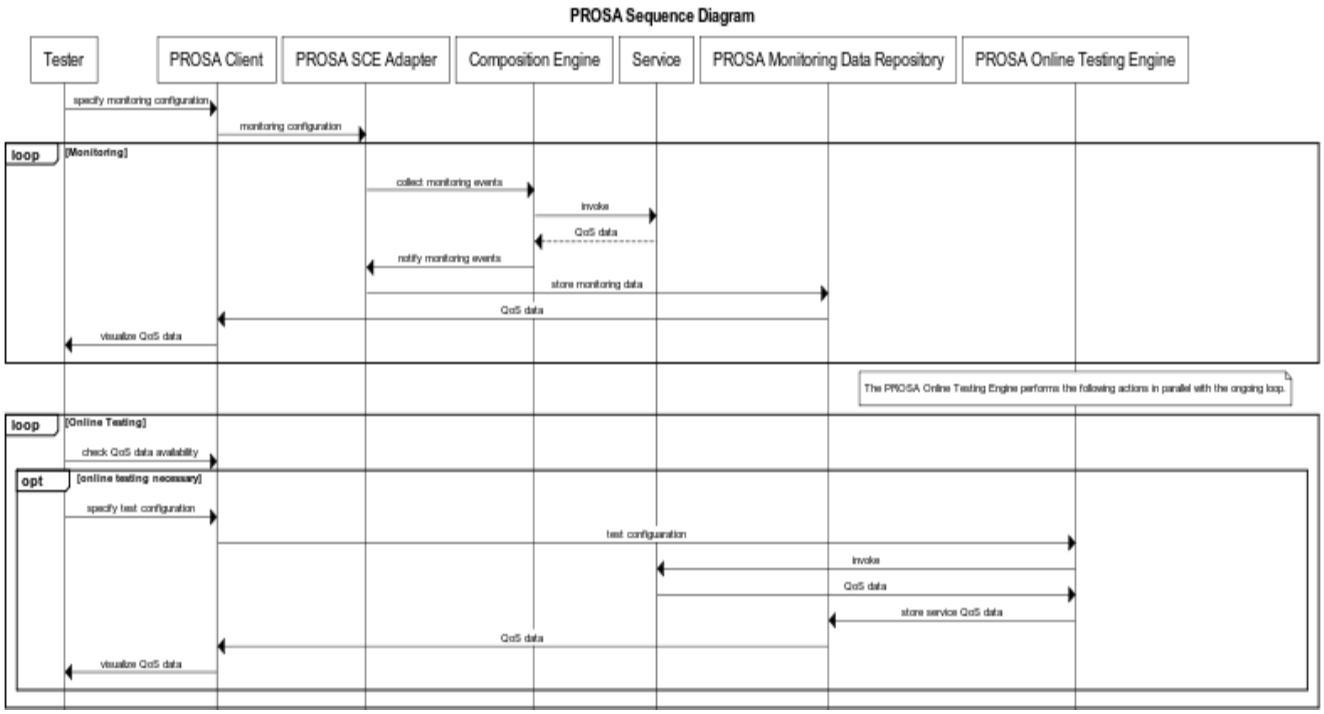
PROSA



The Architecture of PROSA

The above figure shows the architecture of PROSA. The key components of PROSA are the Client, the Online Testing Module and the Monitoring Module. The Client is used to specify parameters for connecting PROSA with service composition engines (SCE; e.g., Apache ODE). It is also used to specify the online testing configuration (e.g., service, input, test rate) for testing constituent services of the compositions (e.g., BPEL). The Monitoring Module interfaces with the composition engines to collect Meta Data such as deployed compositions and Monitoring Events resulting from executing the them. Monitoring data (e.g., QoS Data) obtained from the collected events are stored in the Monitoring Data repository. The PROSA Client visualizes the QoS Data. Based on the Test Configuration, the Online Testing Module invokes the constituent services of a composition for collecting additional QoS data when needed. The collected QoS Data is stored in the Monitoring Repository.

The following sequence diagram shows the interactions during the usage of PROSA.



PROSA - sequence diagram

Deployment Tool

ETICS is a solution for software development and testing lifecycle management. The ETICS system is designed to simplify the development process and realize complex testing scenarios while improving the quality of software. ETICS offers a complete range of testing tool for checking various aspects of software and to write and execute customized tests.

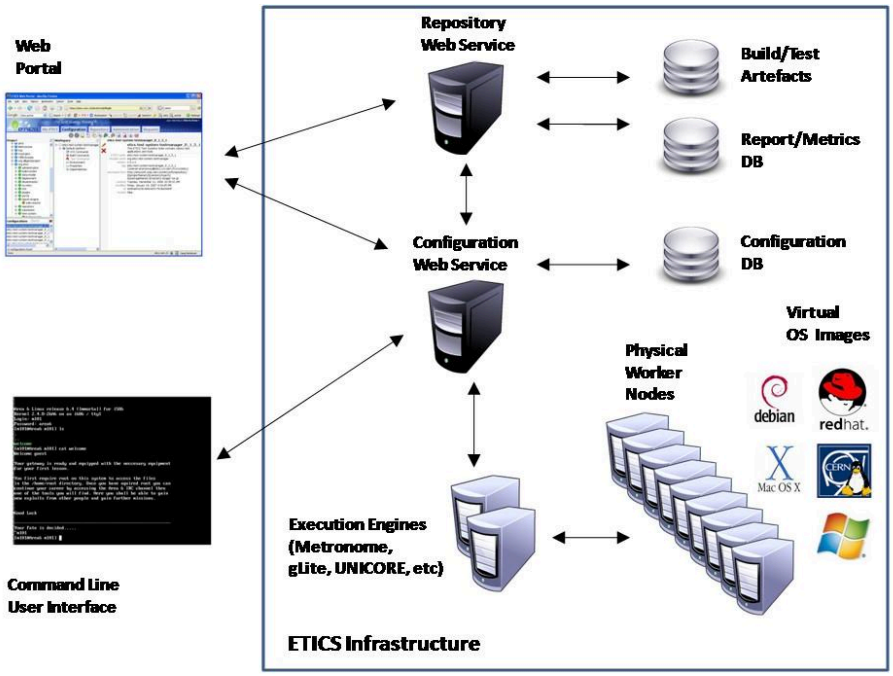
Main functionalities offered by ETICS are:

- **Configuration** and management of different versions and dependencies of software projects and sub-components. Through ETICS each component of a software project can be modelled in detail specifying source code repository path and/or tag, building and packaging commands, building environment, dependencies;
- **Remote Build** of software on remote computational resources. Users can submit builds (from the portal or using web services interface) and the ETICS system will allocate the required computational resources to perform the build. Results (logs, reports, artifacts) will be made available on the ETICS Repository;
- Set up **complex testing scenarios** (e.g. distributed deployment scenarios). ETICS allows to define multi-node testing environments specifying software to

install and testing scripts for each node. ETICS will then executes tests reserving the required number of machines from its pool and automating the execution of testing scripts. At run-time ETICS also provides a synchronization mechanism on all nodes involved in the test;
- Continuous **quality checking** of software. At every build/test, ETICS executes several specific plugins to execute source code analysis and collect measures on code quality. Aggregated reports as well as raw data is made available in the ETICS Repository;
- Automatic creation of **distribution** packages in different formats (e.g. rpm, build, tgz) on the basis of the selected platform.

ETICS can be used, at different levels, by users with different roles in a project. a) Developers use ETICS to configure components, manage dependencies, check software quality, automate build and test activities. b) Release Managers can configure and integrate different components in a unique project release. ETICS also provides a repository that can be used as a distribution site for packages. Finally, ETICS provides c) Project Managers with a vision of the overall project status (components build/test success rate, software quality improvements).

ETICS architecture is based on four key elements (see figure below):
- **Portal**: this is the main access point to ETICS. It allows users to configure their software, submit builds and tests, analyse reports, download artifacts. It interacts with ETICS web services to fulfil the actions requested;
- **BuildSystem Service**: this is the core of ETICS System. It is a Web Service that implements all configuration, build and test functionalities;
- **Execution Engine**: this component consists of a connector between BuildSystem Service and a pool of distributed computational resources (different technologies can be used as backend: condor, metronome, gLite, cloud-based resource allocators). Computational resources are used by ETICS to run build and test jobs;
- **Repository Service**: This component holds reports, logs and artifacts of completed builds/tests. It can be accessed by a web services interface as well as through the Portal.



ETICS Infrastrucure

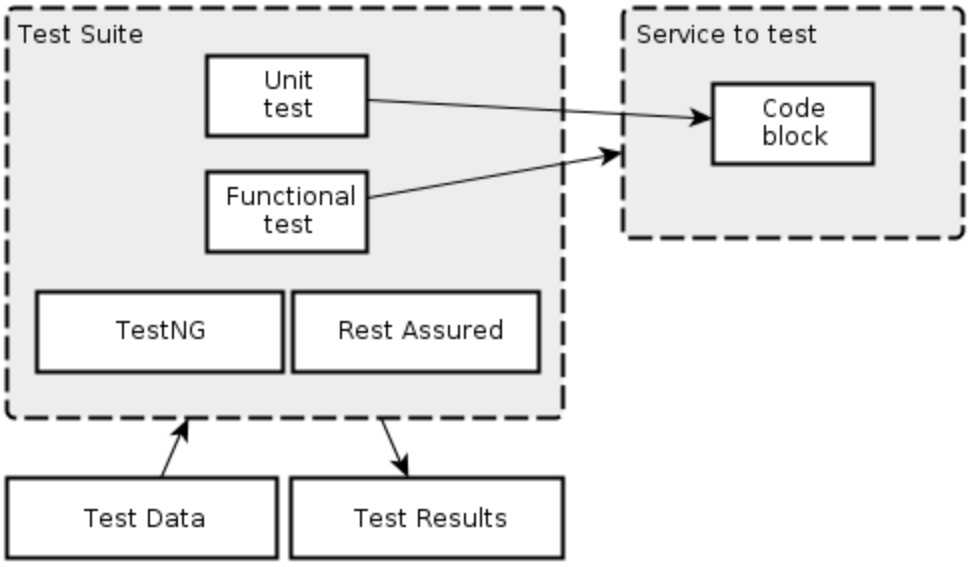
Given its service-oriented and distributed architecture, ETICS well suits projects with geographically distributed teams and/or big projects made of several different modules difficult to build and test properly all together.

UFT Framework

The Unit and Functional Testing (UFT) framework aims to integrate under a single testing project all the testing activities that are implemented in order to validate basic and simple code snippets or more complex functions that are treated as black boxes.

The two main aspects that can be tested using this framework are:

- Unit test: the FIApp provider or the GEI provider can set up and execute this sort of tests in order to validate the internal behaviour of the component;
- Functional test: the FIApp provider or the GEI provider can set up and execute test to validate the functional behaviour of a component or a service. This may be the case for validating the services implemented by a FIApp or made available by a GEI.



UFT Framework

The relevant constraint for developing the test cases using this framework is that the tests have to be implemented in Java language.

The source code of these tests can be properly organized in packages by nature (e.g. unit, functional) and they can be executed in different order or by groups once the source code of the tests is properly annotated. This allow to easily implement different testing scenarios from the smoke tests to massive testing of single components.

The two main components on which this testing framework is based are TestNG (http://testng.org) and Rest Assured (https://code.google.com/p/rest-assured/). Here are summarised the main characteristics of these components and also the benefits derived from their adoption:

- **Annotation:** thanks to the annotation feature of TestNG it's possible to organize the single tests in groups/suites and also to define dependencies between test cases;
- **External test data:** TestNG provides the capability of retrieving the test data to use, from an external source. This way it's possible to keep the test logic separated from the test data. The major benefit of this is that the test data can be edited by domain experts without the need of changing the test logic. The UFT framework provides an utility to retrieve test data from a spreadsheet file;
- **Dedicated support for RESTful services:** The Rest Assured library is used to implement the test cases to validate the behaviour of a RESTful web service. Thanks to it's notation, it's possible to define complex calls and response evaluations. It supports, out of the box, the parsing of XML and Json messages;
- **IDE integration:** Thanks to the dedicated plugin for the IDE (e.g. Eclipse) TestNG tests can be executed obtaining relevant information from the results. The main adoption of this feature is during the test suite development;
- **CI integration:** the major Continuous Integration services allows to execute the test suites created using TestNG. In this use case the results collected are also

presented, usually in a web page, for an easier inspection and for sharing them among the team members.

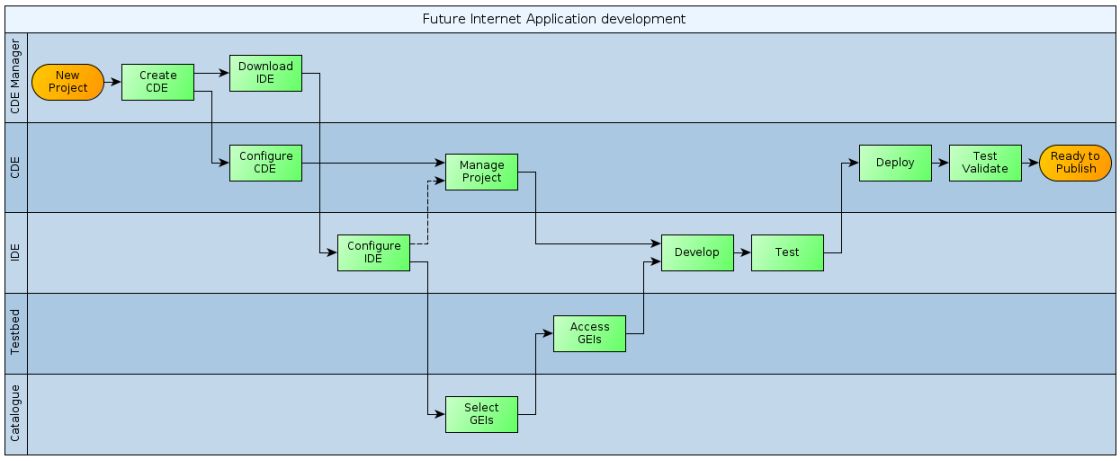
Use Cases

Introduction

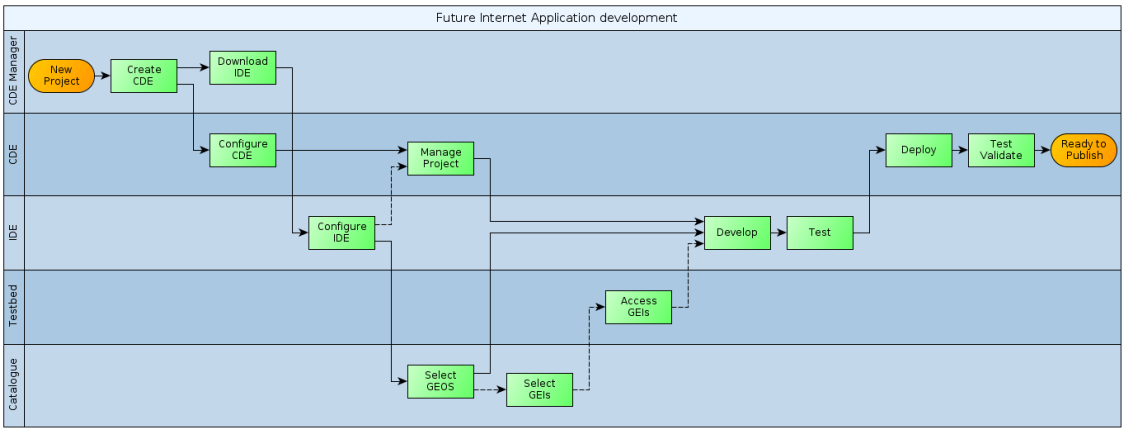
The high level scenarios that will be supported by FI-CoDE are the development of FI-WARE Compliant Platform Products and of Future Internet Applications and Services. The next two figures display the main aspects of both the scenarios.

Legend:

- dashed line: optional path
- dashed boxes: not provided by the FI-CoDE



Future Internet Application development



FI-WARE Compliant Platform Product development

Actors

The development cycles presented in the two high level scenarios (FIApp and GEI development) are executed by some well identified actors that are independent from the considered scenario.

Given the definitions in chapter Roles, below they are provided the actors descriptions that are envisaged in FI-CoDE and specifically tailored to the identified DCT use cases diagrams:

- **Project Manager (PM):** the PM is the main responsible for the realization of the project. It takes care of organizational and financial aspects.
- **Technical Manager (TM):** the TM is responsible for creating the CDE with all the contained services to support the project management and development activities. The TM also has to take the most appropriate technical and architectural decisions, such as which GEI has to be used.
- **Developer (DEV):** the DEV is responsible for the development of the project (FIApp or GEI) using the IDE that is configured accordingly to the CDE services;
- **Tester (TST):** the TST is responsible for the development and execution of the tests for validating the status of the project (FIApp or GEI).

Contexts

The presented high level scenarios (FIApp and GEI development) are elaborated, in the remainder of this chapter, as use cases detailing specific features together with the roles they refer to. In this respect, and for a matter of readability, the main functionalities provided by the FI-CoDE are grouped depending on which topic they mostly refer to:

- Users;
- Collaboration Development Environment (CDE);
- Project (FIApp or GEI);
- Catalogue.

It is worth to note that having a running FI-WARE Instance is not a prerequisite to all the defined use cases, in fact the development of a FI-WARE Compliant Platform Product may rely only on GE Open Specifications. In any case when the prerequisite is to have a running FI-WARE Instance, the use case involves the FI-WARE Instance Provider role.

Use Cases - Users

In order to get access to the services provided by the FI-CoDE platform it's mandatory to be registered. For example, the same user can access the CDE management services and the training courses with one single account and without the need to log in to every service. The User Management service supports the multi-tenancy feature.

User Registration

Name	User Registration

Initiator	Project Manager, Technical Manager, Developer, Tester
Goal	The end user registers an account in FI-CoDE platform in order to have access to the services as one of the predefined roles that have been identified.

User Login

Name	User Login
Initiator	Project Manager, Technical Manager, Developer, Tester
Goal	The registered user can log in to the FI-CoDE platform and access the services without the need to log in again (Single Sign On feature).

Manage Users

Name	Manage Users
Initiator	Project Manager, Technical Manager, Developer, Tester
Goal	Every platform user can manage its own account thanks to a dedicated interface. Here it's possible to add/update an account or remove the user from the platform.

Manage Tenants

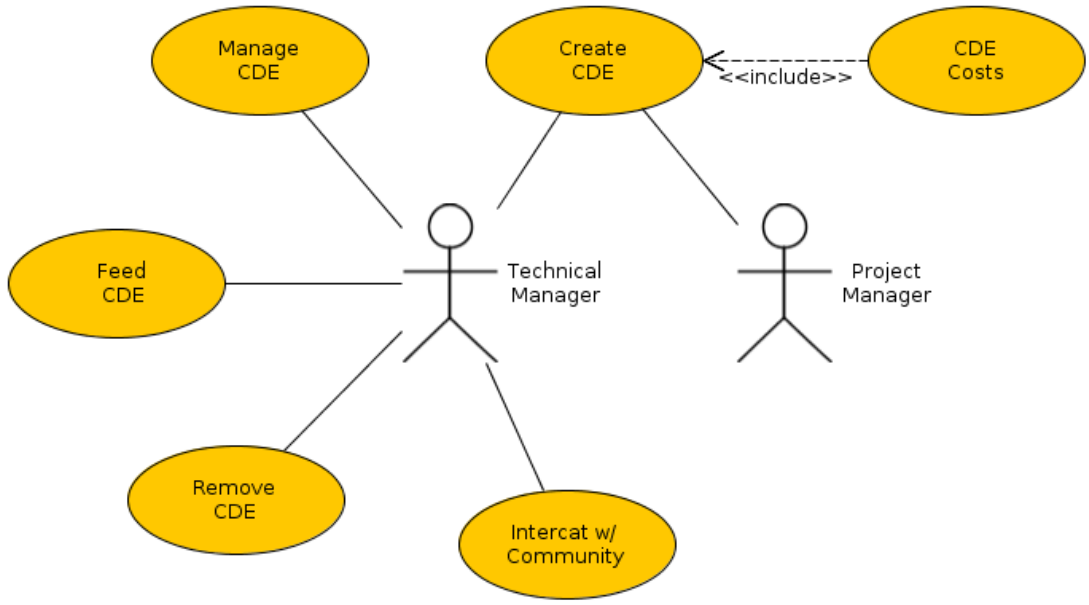
Name	Manage Tenants
Initiator	Project Manager, Technical Manager
Goal	Manage Tenants defined in FI-CoDE platform. This use case includes the actions of Define, Modify, Remove and View a Tenant configuration within the CDE Manager.

Manage Tenant Users

Name	Manage Tenant Users
Initiator	Project Manager, Technical Manager
Goal	Add/Remove platform Users to/from Tenants defined in FI-CoDE platform. This use case includes the actions of listing all the users associated to a Tenant and listing of all platform users.

Use Cases - CDE

This Use Case diagram (Figure: CDE ~~adoption~~ Use Cases) will be enriched by a textual description for each case in the next sections. These textual descriptions will also be used to inspire the User Stories necessary for the development of the various system functionalities.



CDE Use Case

Create CDE

Name	Create CDE
Initiator	Technical Manager, (Project Manager)
Goal	The TM defines the new CDE configuration in terms of resources and services provide by FI-CoDE platform, Testbed, Catalogue and Marketplace, estimating the costs (if any) of the drawn environment. The new CDE is created within the FI-CoDE platform.

Define CDE Properties

Name	Define CDE Properties
Initiator	Technical Manager
Goal	Define/Update all the fundamental properties of a Virtual Platform. This functionality allows a Technical Manager to define

	all the characteristics of a Virtual Platform avoiding, at the same, the complexities of such operation. The Technical Manager is guided by a Wizard to identify the fundamental requirements of the Virtual Platform. The main properties that has to be defined are related to the elements of *compute*, *storage* and *network* for the CDE to be provided by the cloud environment.

Advanced Define CDE Properties

Name	Advanced Define CDE Properties
Initiator	Technical Manager
Goal	Define/Update the properties of a Virtual Platform with higher flexibility compared to the default interface.

Define Elasticity Rules

Name	Define Elasticity Rules
Initiator	Technical Manager
Goal	Define/Update the elasticity ruleset for a Virtual Platform. This functionality allows a Technical Manager to define the elasticity rules for the evolution of a specific Virtual Platform. Elasticity Rules allows a Virtual Machine to scale up / scale down in order to maintain the requested level of service.

Define Platform Services

Name	Define Platform Services
Initiator	Technical Manager
Goal	Define/Update the platform services of a CDE. This functionality allows a Technical Manager to define platform services (e.g. Forge, VCS, HTTP Server, RDBMS) needed to manage and run a FIApp in the context of a CDE.

CDE Costs

Name	CDE Costs
Initiator	Project Manager
Goal	The PM validates the costs expenses required to create and

	maintain the CDE.

Manage CDE

Name	Manage CDE
Initiator	Technical Manager
Goal	The TM: • installs into the CDE the additional services not provided by the FI-CoDE platform; ○ makes these new services available to extend FI-CoDE platform; • is responsible for the CDE maintenance activities along the whole duration of the project; • defines the roles/permissions and groups (or updates predefined ones) for the CDE services; • configures the User Management System that serves the CDE; • adds users, as team members of the project, to the proper groups.

Retrieve a CDE

Name	Retrieve a CDE
Initiator	Project Manager, Technical Manager
Goal	Retrieve a specific CDE. This functionality allows to search and retrieve an existing CDE based on the user permissions.

Export CDE

Name	Export CDE
Initiator	FIA Development Project Deployer
Goal	Create an OVF package as export of a CDE in order to obtain the formal definition of a Virtual Appliance. This export is intended to be applied for the environment containing the running application and all its local dependencies.

Feed CDE

Name	Feed CDE
Initiator	Technical Manager, (Developer, Tester)
Goal	The TM inputs the information to drive the execution and monitoring of the project. This activity depends on the CDE services, some examples are: ● document the requirements into the wiki ● define and update the tickets/tasks into the ticketing/tracking system; ● organize the structure of the version control system ● configure the Continuous Integration (CI) service. The DEV and TST: ● contribute maintaining tickets and tasks updated; ● insert and update issues into the bug-tracker; ● commit source code updates. All to keep updated the shared information into the wiki.

Remove CDE

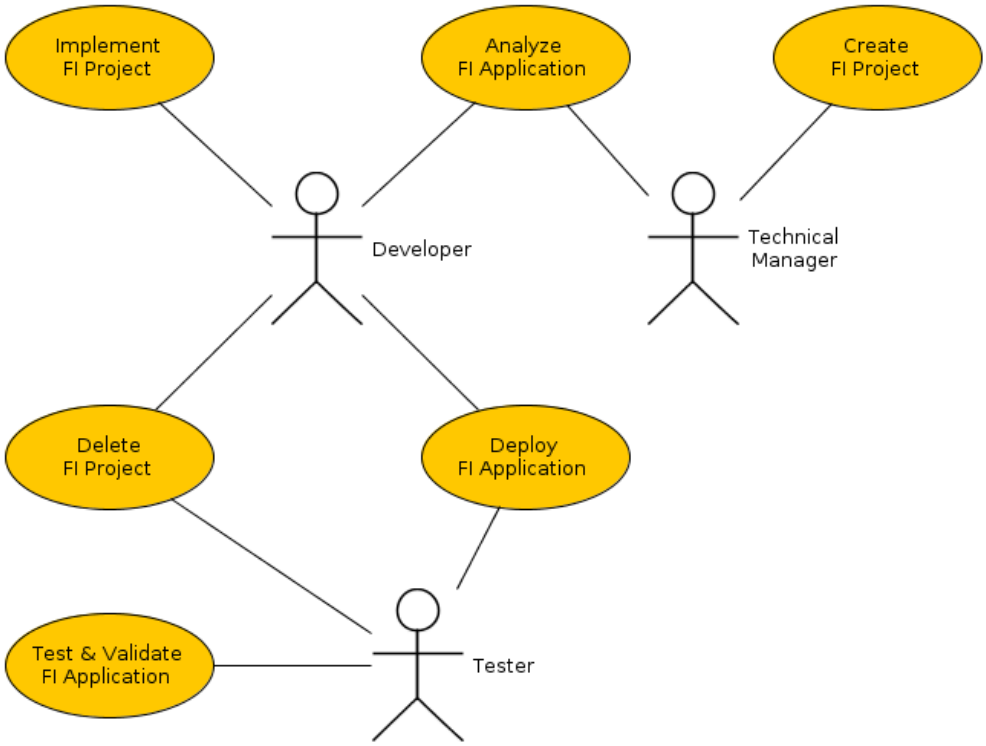
Name	Remove CDE
Initiator	Technical Manager, (Developer, Tester)
Goal	The TM, once the project is over, can archive the project outcomes and free the virtual resources. The DEV and TST that are no more a team member, can locally remove the references to the project.

Interact with Community

Name	Interact with Community
Initiator	Technical Manager, (Developer, Tester)
Goal	Take advantage of the communication channels to: ● create interest around the project; ● increase the usage of the project assets; ● make sure developers and users keep coming back to the community to learn about new things happening; ● take care of feedback regarding the project assets.

Use Cases - FI Project/Application

These use cases, represented in the next diagram (FI Project/Application Use Cases) are enriched by a textual description in the remainder of this sections. These textual descriptions are used to inspire the User Stories necessary for the development of the various system functionalities. In these Use Cases the term FI project is used to represent both FI-WARE Compliant Platform Products and FI Applications.



FI Project/Application Use Cases

Create FI project

Name	Create FI project
Initiator	Technical Manager, (Developer, Tester)
Goal	Create a new project environment in the FI-WARE IDE. The set of Eclipse plugins (developed or selected) support the creation of the software project and it's binding to the management services available from the CDE.

Implement FI project

Name	Create FI project
Initiator	Developer
Goal	Implement a FI Application using the FI-CoDE IDE and the services provided by the CDE created for the project. Integrate in the FIApp the FI-WARE GEIs by generating the proper client libraries.

Test & Validate FI Application

The below use cases cover testing and monitoring using the different tools and approaches provided by FI-CoDE.

Perform unit and functional test on FI Application

Name	Perform unit and functional test on FI Application
Initiator	Tester
Goal	Implement and execute unit and functional tests to validate the behaviour of a FI Application (or GEI). The FI-CoDE IDE makes available a framework to support these testing activities.

Create a Performance Test Suite for a GE Specification

Name	Create a Performance Test Suite for a GE Specification
Initiator	~~Developer~~ Tester
Goal	~~GE specification writers~~ GE Provider should be able to develop a Performance Test Suite for any GE Specification. The performance suite defines GE-specific performance criteria and will serve as a basis for analysis and comparison of GE implementation performance results.

~~Steps~~

~~*(The GE Specification is available and its API is accessible in SoPeCo.)*~~

~~1. The user creates a SoPeCo adapter descriptor defining the configuration parameters for the measurements.~~

1. ~~The user creates a configuration model including a measurement specification that defines the experiment series.~~
2. ~~The user creates a SoPeCo Software Adapter for the target GE Specification that can access the GE API, perform measurements, and finally capture the target metrics.~~
3. ~~The software adapter is added as a plugin to the SoPeCo framework.~~
4. ~~The user will create a new Satellite Starter plugin in the SoPeCo framework which can start an RMI server and publish a corresponding satellite controller for the target GE Specification in order to provide access the software adapter through RMI.~~
5. ~~The user will create a Cockpit Starter for the target GE Specification which can create an instance of SoPeCo and initiate the measurements.~~
6. ~~The user will create a new SoPeCo Test Suite for the target GE Specification with the previously created SoPeCo components.~~
7. ~~She then publishes the SoPeCo Test Suite in the repository for future access by GE developers.~~

~~*(A SoPeCo test suite is available for future performance analysis of GE instances that adhere to the GE Specification.)*~~

~~Extension points~~

~~None at the moment~~

~~Non-Functional requirements~~

~~tbd~~

Perform Performance Measurements on a GE Implementation

Name	Perform Performance Measurements on a GE Implementation
Initiator	~~Developer~~ Tester
Goal	Platform product tester should be able to perform performance tests on their platform products (implementation of GE Specifications) and publish the performance results along with the products.

~~Steps~~

~~*(A GE Implementation is running and accessible and a SoPeCo Test Suite package is available for the GE Specification of the target GE Implementation.)*~~

~~1. The GE developer starts the target GE Implementation, selects the corresponding SoPeCo Test Suite and binds the SoPeCo Test~~

Suite to the running GE Implementation.

8. The GE developer consequently starts the SoPeCo Satellite Starter and Cockpit Starter for the target GE Implementation to perform the measurements.
9. The GE developer analyzes the data by feeding the previously generated measurement data into the SoPeCo visualization tool.
10. The GE developer publishes the results of her/his performance analysis on the target GE Implementation in the repository accessible by App or GE developers.

(The results of performance analysis are accessible in the repository.)

Extension points

None at the moment

Non-Functional requirements

tbd

Retrieve the latest published performance analysis data for a GE

Name	Retrieve the latest published performance analysis data for a GE
Initiator	Tester
Goal	The application tester should be able to retrieve and validate the latest performance analysis results of a given GE implementation (platform product).

Steps

(The results of performance analysis for a given GE Implementation are available in the repository.)

1. The App developer searches for a specific GE Implementation, and chooses to view the results of performance analysis on that specific instance.

Extension points

None at the moment

Non-Functional requirements

tbd

Perform Performance Measurements on a FI Application

Name	Perform Performance Measurements on a FI Application
Initiator	Tester
Goal	FI Application tester should be able to define and perform performance tests on their applications and view the performance results.

~~Steps~~

1. ~~The user performs the Use Case Create a Performance Test Suite for a GE Specification for a FI Application instead of a GE.~~
11. ~~The user performs the Use Case Perform Performance Measurements on a GE Implementation for a FI Applications instead of a GE.~~

~~Extension points~~

~~None at the moment~~

~~Non-Functional requirements~~

~~tbd~~

Setup of a FI Application Monitoring with Trace Analyzer

Name	Setup of a FI Application Monitoring with Trace Analyzer
Initiator	Tester
Goal	FI Application tester should be able to select tools and monitoring options for their performance tests.

~~Steps~~

~~*(The FI Application is running in its target environment.)*~~

1. ~~The user opens the FI-WARE IDE, selects the specific FI Application project and chooses the environment to be set up.~~
12. ~~The user opens the Trace Analyzer configuration~~
13. ~~Out of the list of the available monitoring tools, the user selects a target tool and sets up the monitoring options for it~~
14. ~~The user repeats step 3 for all the target monitoring tools~~
15. ~~The user saves the changes. The FI-WARE IDE updates the changes in the runtime environment of the FI Application.~~
16. ~~The user closes the FI-WARE IDE.~~

~~*(The FI Application on the specified environment is being monitored by Trace Analyzer through the specified tools.)*~~

~~Extension points~~

~~None at the moment~~

~~Non-Functional requirements~~

~~tbd~~

Monitor Services in a Composition with PROSA

Name	Monitor Running Services
Initiator	Tester
Goal	Continuously Collect and visualize monitoring data (i.e., response times) of services running in a service composition engine.

Online Services in a Composition with PROSA

Name	Test Running Services
Initiator	Tester
Goal	Collect additional monitoring data (i.e., response times; when necessary) for the services running in a service composition engine by testing them.

Deploy FI Application

Name	Deploy FI Application
Initiator	Developer, Tester
Goal	Deploy FI project into the CDE part dedicated to run the application. The deployment activity is supported by dedicated services that are made available by the FI-CoDE platform.

Analyse FI Application

Name	Analyse FI Application
Initiator	Technical Manager, Developer
Goal	Gain information about the QoS of a running FI project

Analysis of service response times

Name	Analysis of service response times
Initiator	Developer
Goal	FI Application developers should be able to view the monitoring results and apply filtering and aggregation to reduce the amount of data. The FI Application on the specified environment is monitored to collect service responsiveness data (service name, request time, response parameters, optional: request parameters, response status). The Trace Analyzer presents the user with a sequence of charts showing number of concurrent requests handled and their durations. The charts show overall state as well as state by request type.

~~Steps~~

~~*(The FI Application on the specified environment was monitored to collect service responsiveness data (service name, request time, response parameters, optional: request parameters, response status).)*~~

~~1. The Trace Analyzer parses the monitoring data.~~

17. ~~The Trace Analyzer aggregates and associates related events (e.g., corresponding request and response events)~~
18. ~~The Trace Analyzer presents the user with a sequence of charts showing number of concurrent requests handled and their durations. The charts show overall state as well as state by request type.~~
19. ~~The user applies optional filters to visualize only requests with certain parameters or requests with duration above the user-set threshold.~~
20. ~~The user goes to the statistics view to view metrics summary for the trace.~~
21. ~~The user identifies additional data to be collected for more in-depth analysis (e.g., more detailed monitoring for certain service types, or system monitoring for cross-cutting problems.~~

~~*(The monitoring data available for the target run is presented for the user analysis.)*~~

~~Extension points~~

~~None at the moment~~

~~Non-Functional requirements~~

~~tbd~~

Analysis of causes for long service response times

Name	Analysis of causes for long service response times
Initiator	Developer
Goal	FI Application developers should be able to easily observe important characteristics such as response times. The Trace Analyzer presents the user with charts combining request duration data (optionally filtered by the type of request and its parameters) together with the additional monitoring data (visualization depending on the type of additional data; for example: threads in blocking states, cpus and their activity breakdown, etc.)

~~Steps~~

~~(The FI Application on the specified environment was monitored to collect service responsiveness data (service name, request time, response parameters, optional: request parameters, response status) as well as additional system or application data (queue states, synchronization data, IO activity, etc.))~~

1. ~~The Trace Analyzer parses the monitoring data.~~
2. ~~The Trace Analyzer aggregates and associates related events (e.g., corresponding request and response events)~~
3. ~~The Trace Analyzer presents the user with charts combining request duration data (optionally filtered by the type of request and its parameters) together with the additional monitoring data (visualization depending on the type of additional data; for example: threads in blocking states, cpus and their activity breakdown, etc.)~~
4. ~~The user turns on the options to highlight the time intervals where the service times were beyond the user-defined threshold.~~
5. ~~The user consults the Performance Anti-Patterns view identifying possible issues within the collected data (e.g., hot locks, blocking IO within locks, etc.)~~
6. ~~The user identifies whether the metrics collected are related to the cause of the unsatisfactory response times.~~

~~(The monitoring data available for the target run is presented for the user analysis.)~~

~~Extension points~~

~~None at the moment~~

~~Non-Functional requirements~~

~~tbd~~

Delete FI project

Name	Delete FI project
Initiator	Developer, Tester

Goal	Delete a project environment in the FI-WARE IDE.

Annex

Here are reported the two questionnaires that have been used to collect requirements that are used to support the decision taken at Tools Chapter level and to define the road-map for the features to make available.

The FI-WARE WP9 Questionnaire is focused on gathering of information from the FI-WARE project partners. It has been produced in two subsequent versions (second version available here below) with a simplified second one that had the aim of collecting more contributions compared to the first call. The partners that haven't sent their contributions have been contacted in person during the Madrid plenary meeting in January 2012.

The FI-WARE IDE Survey is a specific questionnaire focused on collecting the priorities, based on personal viewpoint, regarding a set of proposed features to insert into the road-map for the development of the FI-WARE IDE. This survey has been useful also to collect additional features required by people directly involved in development aspects, also outside FI-WARE project partners and especially within partners corporate colleagues.

FI-WARE WP9 Questionnaire

Version 2.0

General

1. Is the document sufficiently clear in describing "the what" and "the how"? *(List the sections to improve specifying if it has to be improved the content or the readability)*
27. Do you have any specific standard or technological constraint?
28. Are the Use Cases descriptions sufficiently detailed and clear to understand what the functionality is expected to do?
29. Do you think additional functional and non-functional requirements are needed in the proposed environment? If yes, which

ones?

30. How do you imagine your GE will be used?

31. How are GE Open Specifications defined? Do they define the interface of the GEs in a standard/commonly-accepted language?

32. If there is no standard/commonly-accepted language for defining GE specifications, HOW will YOU define your GE Open Specification?

IDE - CDE Interaction

1. Is it clear the purpose of the integration between the IDE and the CDE?

33. What are the most valuable features you would expect from the CDE facilities directly accessible from the IDE? (e.g. CRUD on tickets, ...)

34. If you were using the FI-WARE IDE what of the following functionalities you would expect to use directly from it? Please, prioritise (HA - highly appreciable, MA - Medium Appreciable, LA - Low Appreciable)

a. Tickets *(manage the tickets of the forge from the Eclipse IDE, based on Mylyn)*

b. Bugs *(the same as for Tickets)*

c. Tasks *(the same as for Tickets)*

d. Cross contents searches *(an IDE based search interface that lets to query different sources and displays aggregated results)*

e. Contextual Menu: *(starting from a text/code selection)*

- add a new Forum post
- add a new Bug/Task/Ticket
- start a collaborative session *(see Collaborative editing)*

f. Contextual Chat *(start a chat directly addressed to a specific topic - e.g. method, lib, etc.)*

g. Mailing list access and Mail notification

h. User creation/authorization *(forge users management)*

i. Policy settings

j. Project creation

- new project into the forge
- selecting a specific type *(GE dev. - Application dev.)*

k. Search for GE clients within the FI-WARE Catalogues *(a wizard that lets to browse the specific Catalogue and select the GE clients to use)*

l. Enabling FI-WARE instance *(IDE configuration to work with a specific FI-WARE instance - supported by wizard)*

m. Social networking interaction *(send and receive updates to/from the major social networks; e.g. twitter)*
n. Relationship Service *(an IDE interface that enables to relate various items among each other - source code, bugs, tickets, forum, chat, etc.)*
o. Collaborative Editing *(edit simultaneously a source file with other members of the team)*

35. In your opinion are there functionalities currently not considered? If yes, which ones?

Generic Enablers

1. Do your GEs depend on other GEs in order to be used? If yes, synthetically explain these dependencies.

Catalogue

1. Do you see something important that's missing in the high level features of the Catalogue?
36. What are the most valuable features you would expect from the Catalogue?

Deployment / Test / Validation

Questions to all GE Providers

Performance testing

1. For what purposes do you need performance data? Please choose all the options that apply.
   - SLA monitoring
   - Monitoring performance at interfaces with other services
   - Monitoring performance of GE and its components
   - Bottleneck prediction
   - Other, please specify: _____

37. What information are you interested in? Please specify the

combinations you are interested in ("I want to see CPU and IO data together") and acceptable aggregation levels. Please choose all the options that apply.

- CPU
- IO
- Network
- Hypervisor activity (which?)
- Guest OS activity (which?)
- Performance of interfaces to other services
- Inter-thread/process communication
- Performance of user-defined internal interfaces
- GE performance characteristics
- Other, please specify: _____

38. What are the most important aspects of your GEs that has to be monitored at run time? (response time, bandwidth, server resources, ...)
39. What do you perceive to be the most important challenges when testing software and/or services that rely on the capabilities offered by your chapter?

Deployment testing

1. Can the platform used to run your software be installed on common Linux machine? E.g. if your GE run on a mobile platform, There exist any simulation tool that make it possible to deploy the GE on a Linux machine and test from there?
40. How many distributed nodes do you expect to use in a typical deployment test for your software?

- one (e.g. your sw is installed on the node and a testing script check everything is like expected)
- two (e.g. client-server: the server is installed on a node and the client make some calls from a second node. This would allow to check also the network configuration of the deployment)
- 3 to 5
- more than 5. Please quantify: _____

41. Can deployment tests be executed in batch mode (non-interactive mode, without human intervention)?

Questions to Service Composition Engine Provides (WP3)

1. Which information are being monitored by your service composition engines (business process engine, server-side mashup

engine, ...) that are available from the outside?
- Execution status of a service composition
- Actual variable binding of a service composition
- Performance information (expired runtime, call durations of external services, ...)
- No monitoring available.

42. Do your service composition engines (business process engine, server-side mashup engine, ...) provide means for adaptation (e.g. replacement of a called service implementation at runtime)?
- Yes, even for RUNNING instances!
- Yes, but only on MODEL level (not for specific instances).
- No adaptation possible at all.

Questions to Cloud Service Providers (WP4)

1. For what purposes do you need performance data? Please choose all the options that apply.
- SLA monitoring
- Infrastructure monitoring
- Bottleneck prediction
- Co-location optimization
- Other, please specify

43. What information are you interested in? Please specify the combinations you are interested in ("I want to see CPU and IO data together") and acceptable aggregation levels. Please choose all the options that apply.
- CPU
- IO
- Network
- Hypervisor activity (which?)
- Guest OS activity (which?)
- Application performance characteristics
- Other, please specify: _____

44. What information can you expose to the GEs for application performance analysis? Please choose all the options that apply.
- CPU
- IO
- Network

- Hypervisor activity (which?)
- Guest OS activity (which?)
- Other, please specify: _____

45. Which client-related performance information is of interest to identify possible bottlenecks
46. Which monitoring data about communication with clients can be provided for this purpose (identification of possible bottlenecks)?
47. What SLA information is available to Cloud Service Providers?
48. What kind of performance data does the monitoring GE provide for Cloud services? Please give some examples.

FI-WARE IDE Survey

The survey, to collect priorities information on proposed FI-WARE IDE features is reported here for convenience as it has been integrated into the FI-WARE WP9 Questionnaire.

Together with the proposed features here are reported also the results, in square brackets, as a number between 1 and 3. This value is computed as average of the votes provided (HA=3, MA=2, LA=1). The design and the planning of the Tools Chapter activities has been based also on the highest evaluated features obtained from the survey result.

If you were using the FI-WARE IDE what of the following functionalities you would expect to use directly from it? Please, prioritise (HA - highly appreciable, MA - Medium Appreciable, LA - Low Appreciable)

1. Tickets *(manage the tickets of the forge from the Eclipse IDE, based on Mylyn)* **[2,71]**
49. Bugs *(the same as for Tickets)* **[2,71]**
50. Tasks *(the same as for Tickets)* **[2,71]**
51. Cross contents searches *(an IDE based search interface that lets to query different sources and displays aggregated results)* **[2,18]**
52. Contextual Menu: *(starting from a text/code selection)*
    - add a new Forum post **[1,83]**
    - add a new Bug/Task/Ticket **[2,18]**
    - start a collaborative session *(see Collaborative editing)* **[2,27]**

53. Contextual Chat *(start a chat directly addressed to a specific topic - e.g. method, lib, etc.)* **[1,16]**
54. Mailing list access and Mail notification **[1,28]**
55. User creation/authorization *(forge users management)* **[1,83]**
56. Policy settings **[1,33]**

57. Project creation
    - new project into the forge **[2,06]**
    - selecting a specific type *(GE dev. - Application dev.)* **[2,22]**

58. Search for GE clients within the FI-WARE Catalogues *(a wizard that lets to browse the specific Catalogue and select the GE clients to use)* **[2,25]**
59. Enabling FI-WARE instance *(IDE configuration to work with a specific FI-WARE instance - supported by wizard)* **[2,07]**
60. Social networking interaction *(send and receive updates to/from the major social networks; e.g. twitter)* **[1,47]**
61. Relationship Service *(an IDE interface that enables to relate various items among each other - source code, bugs, tickets, forum, chat, etc.)* **[2,17]**
62. Collaborative Editing *(edit simultaneously a source file with other members of the team)* **[2,42]**

In addition to the features proposed, other requests has been pointed out and here are reported the most interesting ones:

1. Test management tool integration
63. Database design and reverse engineering
64. UML and ER modelling tools
65. License compatibility check

Some of these are already taken into account for the design and planning of the next activities (e.g. Test management tool integration).