

Deliverable D300.8

Experimentation Environment architecture, development and scenarios execution plan

WP 300

| | |
|---|--|
| Project Acronym & Number: | Flspace – 604 123 |
| Project Title: | Flspace: Future Internet Business Collaboration Networks in Agri-Food, Transport and Logistics |
| Funding Scheme: | Collaborative Project - Large-scale Integrated Project (IP) |
| Date of latest version of Annex 1: | 03.10.2013 |
| Start date of the project: | 01.04.2013 |
| Duration: | 24 |
| Status: | Final |
| Authors: | IBM: Dany Moshkovich UDE: Clarissa Cassales Marquezan NKUA: Sokratis Bampounakis |
| Contributors: | |
| Document Identifier: | Flspace-D300.8-V005.docx |
| Date: | 30.10.2013 |
| Revision: | 005 |
| Project website address: | http://www.Flspace.eu |

The Flspace Project

As a use case project in Phase 2 of the FI PPP, Flspace aims at developing and validating novel Future-Internet-enabled solutions to address the pressing challenges arising in collaborative business networks, focussing on use cases from the Agri-Food, Transport and Logistics industries. Flspace will focus on exploiting, incorporating and validating the Generic Enablers provided by the FI PPP FI-Ware project with the aim of realising an extensible collaboration service for business networks together with a set of innovative test applications that allow for radical improvements in how networked businesses work in the future. These solutions will be demonstrated and tested through early trials on experimentation sites across Europe. The project results will be open to the FI PPP program and the general public, and the pro-active engagement of larger user communities and external solution providers will foster innovation and industrial uptake planned for Phase 3 of the FI PPP.

The project will lay the foundation for realizing the vision and prepare for large-scale expansion, complying with the objectives and expected results of the Phase 2 use case projects. To achieve these outcomes the project will focus on the following four primary work areas, for which the main concepts and approach are outlined below:

1. **Implement the Flspace as an open and extensible Software-as-a-Service solution** along with an **initial set of cross-domain applications** for future B2B collaboration, **utilizing the Generic Enablers** provided by the FI-Ware
2. **Establish Experimentation Sites across Europe** where **pilot applications are tested in early trials** from the **Agri-Food and the Transport and Logistics** domains
3. **Provide a working Experimentation Environment** for conducting **early and large-scale trials** for Future Internet enabled B2B collaboration in several domains, and

Prepare for industrial uptake and innovation enablement by pro-active engagement of stakeholders and associations from relevant industry sectors and the IT industry.

Project Consortium

- | | |
|--------------------------------------|--|
| – DLO; Netherlands | – Kühne + Nagel; Switzerland |
| – ATB Bremen; Germany | – University Duisburg Essen; Germany |
| – IBM; Israel | – ATOS; Spain |
| – KocSistem; Turkey | – The Open Group; United Kingdom |
| – Aston University; United Kingdom | – CentMa; Germany |
| – ENoLL; Belgium | – iMinds; Belgium |
| – KTBL; Germany | – Marintek; Norway |
| – NKUA; Greece | – University Politecnica Madrid; Spain |
| – Wageningen University; Netherlands | – Arcelik; Turkey |
| – PlusFresc; Spain | – EuroPoolSystem; Germany |
| – FloriCode; Netherlands | – GS1 Germany; Germany |
| – Kverneland; Netherlands | – Mieloo & Alexander; Netherlands |
| – North Sea Container Line; Norway | – OPEKEPE; Greece |
| – LimeTri; Netherlands | – Innovators; Greece |

More Information

Dr. Sjaak Wolfert (coordinator)
LEI Wageningen UR
P.O. Box 35
6700 AA Wageningen

e-mail: sjaak.wolfert@wur.nl
phone: +31 317 485 939
mobile: +31 624 135 790
www.Flspace.eu

Dissemination Level

| | | |
|-----------|---|----------|
| PU | Public | X |
| PP | Restricted to other programme participants (including the Commission Services) | |
| RE | Restricted to a group specified by the consortium (including the Commission Services) | |
| CO | Confidential, only for members of the consortium (including the Commission Services) | |

Change History

| Version | Notes | Date |
|---------|--|------------|
| 001 | Creation of document & initial structure | 01.08.2013 |
| 002 | Ready for internal review | 28.10.2013 |
| 003 | WP300 Internal review | 29.10.2013 |
| 004 | Updated after review by Rod Franklin | 30.10.2013 |
| 005 | Final formatting of the document | 30.10.2013 |
| | | |

Document Summary

This report describes the experimentation environment architecture, development, and scenario execution plans. For clarity sake, this document is self-contained—it includes the summary and refinement parts of the relevant WP4 deliverables (<http://Flnest-ppp.eu/project-results/deliverables>) of the Flnest project. This document is composed of five parts. Section 2 gives an overview of the EE, Section 3 provides a development plan, Section 4 provides the scenario execution plan and Section 5 describes in detail the architecture of the EE. We conclude the report with a summary.

Abbreviations

| | | | |
|---------|---|-----|---------------------------|
| CKPI | Composite Key Performance Indicator | KPI | Key Performance Indicator |
| EE | Experimentation Environment | MVC | Model-View-Controller |
| Flspace | Future Internet Enabled Optimization of transport and Logistics Business Networks | TCP | Transport Chain Plan |
| GE | Generic Enabler | UI | User Interface |
| IoT | Internet of Things | WP | Work Package |

Table of Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 6 |
| 2 | Experimentation environment overview | 7 |
| 2.1 | Execution association | 9 |
| 3 | Experimentation environment development plan | 11 |
| 4 | Experimentation environment scenarios execution plan | 14 |
| 4.1 | Scenario execution template | 14 |
| 4.2 | Advice Request scenario execution plan | 16 |
| 4.3 | Advice Request Scenario setup and execution process | 18 |
| 4.3.1 | Flspace Cloud Hosting Infrastructure | 20 |
| 4.3.2 | Experimental Set-up | 22 |
| 4.3.3 | Experiment Execution | 23 |
| 5 | Execution Environment architecture | 26 |
| 5.1 | Main terms | 26 |
| 5.2 | Experimentation environment components | 27 |
| 5.3 | Data types definitions | 27 |
| 5.3.1 | Experiment (DataType) | 27 |
| 5.3.2 | VariableSpecification (DataType) | 28 |
| 5.3.3 | Step (DataType) | 29 |
| 5.3.4 | Execution (DataType) | 30 |
| 5.3.5 | Resource (DataType) | 30 |
| 5.3.6 | Link (DataType) | 31 |
| 5.3.7 | ExecutionLogEntry (DataType) | 31 |
| 5.3.8 | Report (DataType) | 32 |
| 5.4 | Interfaces definitions | 33 |
| 5.4.1 | Non-component interfaces | 33 |
| 5.4.2 | Component interfaces | 36 |
| 6 | Summary | 43 |

List of Figures

| | |
|---|----|
| Figure 1: Flspace experimentation environment architecture. | 8 |
| Figure 2: Example of setting a new value for a new shipment | 9 |
| Figure 3: WP 300 Flspace Hosting and Experimentation..... | 12 |
| Figure 4: Experimentation process | 13 |
| Figure 5: Simplified Advice Request Scenario..... | 19 |

List of Tables

| | |
|--|----|
| Table 1: Description of Flspace Cloud Hosting for Advice Request Scenario..... | 20 |
| Table 2: Description of Experimentation Set-up for Advice Request Scenario | 22 |
| Table 3: Description of Execution for Advice Request Scenario with Simulated data..... | 23 |
| Table 4: Description of Execution for Advice Request Scenario with Real Stream of Data | 25 |

1 Introduction

The Flspace project aims to address fundamental changes in how collaborative business networks will work in the future. It is the direct continuation of the work done in the Phase 1 use case projects (Flnest and SmartAgriFood). The Flspace project entails a plan to develop a multi-domain business collaboration space (Flspace) that employs FI technologies for enabling seamless collaboration in open, cross-organizational business networks, establishing eight working experimentation sites in Europe, where pilot applications are tested in early trials for Agri-Food and Transport and Logistics domains in preparation for industrial uptake by engaging with players and associations from relevant industry sectors and from the IT industry.

Flspace Work Package 300 (WP300) deals with the identification and design of an Experimentation Environment (EE) for testing, demonstrating, and evaluating the envisioned technologies. WP300 aims to provide a suitable environment for conducting the experiments for the use case scenarios for Agri-Food and Transport and Logistics. WP300 will use real life data feeds and simulation to test and validate the Flspace service and its supporting GEs. Simulation is a powerful tool to test, observe, and gain understanding of new concepts, processes, and technologies under current and future scenarios. Simulation of an end-to-end scenario can enable a deeper understanding of the Flspace platform and the evaluation of future processes on top of this new technology.

The work done by WP4 of Flnest project addressing a preliminary EE design has been re-checked and has been found to be relevant and appropriate for the changes introduced over the first six months of the Flspace project and therefore has been used as a basis for the current definitions in WP300.

This report describes the Experimentation Environment architecture, development and scenarios execution plans. For clarity sake, this document is self-contained—it includes the summary and refinement parts of the relevant WP4 deliverables (<http://Flnest-ppp.eu/project-results/deliverables>) of the Flnest project. This document is composed of five parts. Section 2 gives an overview of the EE, Section 3 provides a development plan, Section 4 provides the scenario execution plan and Section 0 describes in detail the architecture of the EE. We conclude the report with a summary.

2 Experimentation environment overview

The Flspace experimentation environment will operate by activating the Flspace platform and will invoke it at each test execution utilizing Flspace technologies and databases. It consists of three interconnected primary components (see Figure 1):

- *Flspace test*. This is a replica of the *Flspace* platform used for testing purposes to avoid "playing" in the production environment. It is anticipated that test executions will be enabled with real-data as well as with simulated data. The UI will be extended to support both modes.
- *Flspace experimentation environment*. This includes all components required to run and analyze test executions, as well as databases for the storage of executions, execution logs, reports, KPI(s), test data, resources, and roles and access rights.
- *Flspace experimentation environment front-end*. This is the UI that enables users to use the experimentation environment to create, update, execute, and report on tests.

The EE architecture follows the Model-View-Controller (MVC) paradigm characterized by:

- *Model*. This is the knowledge of the system, including the entities, statuses, and states, and the necessary logic for creating and conducting experiments.
- *View*. This refers to the presentation and representations of the model. In this case, the displayed information includes experiment steps and reports.
- *Controller*. The controller is the link between the user (the view) and the system (the model). It receives the user's input and updates the model state accordingly.
- *The EE (model)*. This contains the components required to realize all functionality, including the storage of experiments, execution states, execution logs, reports, and data to be used during execution.
- *The Experimentation Front End*. This refers to the UI for the users to be able to use the experimentation environment to create, update, execute and report on tests. It includes the following high-level views (see Figure 1):
 - *Access Handling*: Controls the access to the experimentation environment and EE artifacts (experiments, execution logs, reports, etc.)
 - *Experiment Management*: The management of experiments, including finding, creating and updating experiments
 - *Execution Management*: Creating and managing the execution of experiments
 - *Resource Management*: Provides basic information on available resources and allows managing the resources in the system
 - *Reports*: Finding, creating, editing, and viewing reports over executions

The system (EE) interacts with the *Flspace* Test system through a *Backend Simulator* component. This includes injecting data into *Flspace* test and recording events and other data processed by *Flspace* test so as to enable the calculation of KPIs.

We foresee that a few components of the envisioned EE may be off-the-shelf components, that is, can be bought as specific-purpose components to be incorporated into the EE for specific purposes. Specifically, we believe that the *reporting* component and the *script engine* component (for executing user scripts), can be off-the-shelf and do not require self-development by the Flspace team. Furthermore, we expect reporting (together with KPIs) capabilities to become a separate application from the experimentation environment and be part of the services provided by Flspace.

Figure 1 presents the Flspace EE architecture. A detailed description of the different modules and definition of the data types and interfaces are given in Section 0. Note that the technical architecture depicted in Figure 1 is defined at the model-level, using TAM (the Technical Architecture Modeling language)¹, a UML derivate, following the convention used in the other technical work packages.

¹ <http://www.fmc-modeling.org/fmc-and-tam>

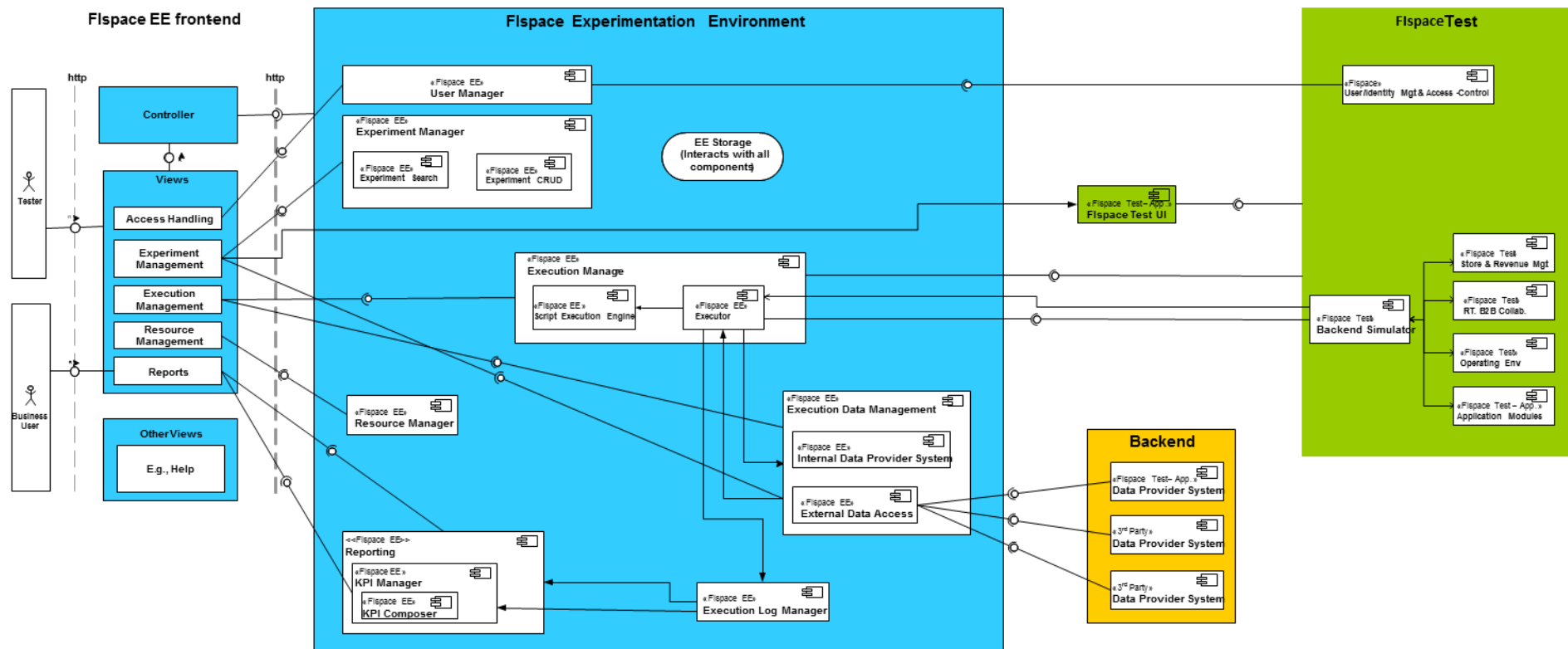


Figure 1: Flspace experimentation environment architecture.

2.1 Execution association

The first step of an execution includes the set-up required. One possible example is illustrated in Figure 2, using the Application transport module. In the experiment, an ExposedDataProvider is defined for storing the shipment ID. Another data provider is used to provide the link for creating a new shipment, which passes through the URI for accessing the ShipmentIdDataProvider.

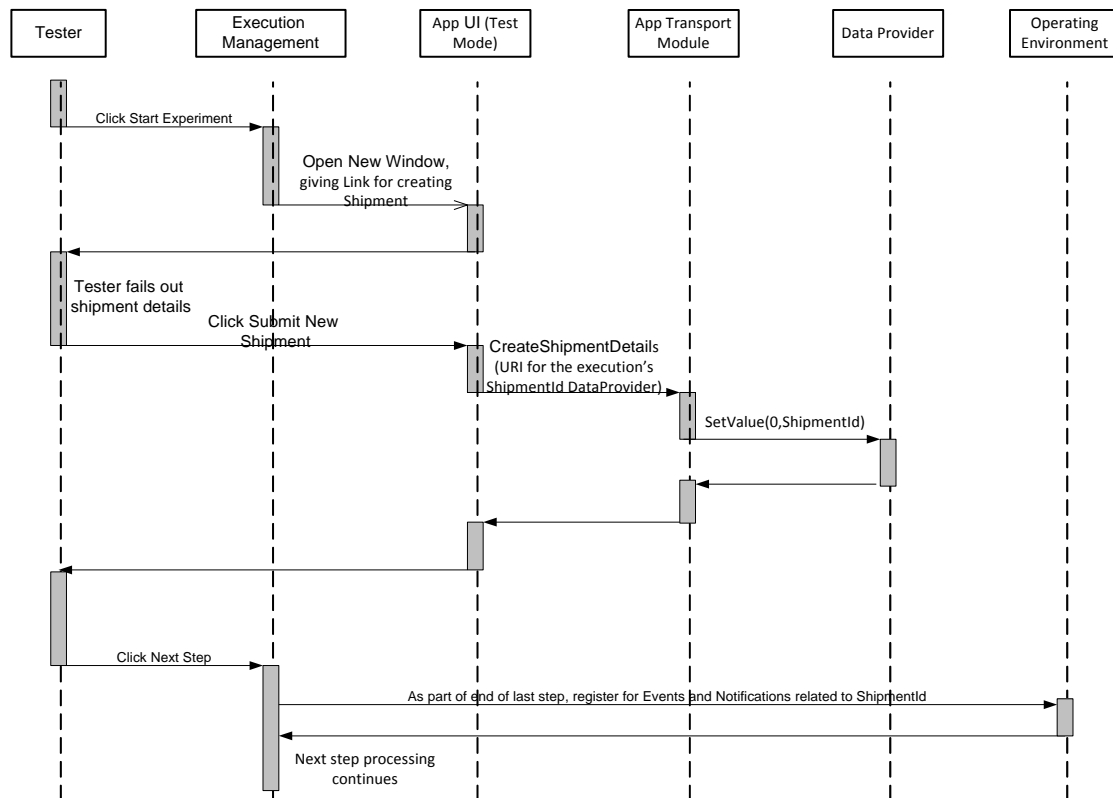


Figure 2: Example of setting a new value for a new shipment

Extensions for supporting applications

- Dynamic data binding:
 - Data Provider Factories: These allow for the dynamic creation of data providers. These data providers may need to communicate with the application; if so, the factories should be provided by the application.
 - Dynamically Bound Data Providers: These entail the lazy creation of a data provider through a factory. These data providers should be defined using variable names that are referenced during execution. These providers should most likely be defined by the experiment author and configured to be saved in the internal data provider system.
- Exposed Data Providers: These applications can access and potentially update data for data providers that are declared as being exposed to applications. If needed for updating data, then these should be DynamicallyBoundDataProviders as well (i.e., they should be created by a factory so that different executions have different instances).
- Links: Links are first-class citizens. They are created to give access to the application and pass through information needed by the application, such as access details for exposed data providers for accessing and storing data.
- Explicit registrations for notifications and events: Since the notifications and events are application specific; the registration must be explicit as well.

Application-specific components are labeled with “Flspace Test – App” in the architecture diagram to differentiate them from the Flspace environment, which are labeled “Flspace Test”.

Application Requirements

Flspace applications are required to supply certain capabilities so that they can be supported in the Flspace Experimentation Environment. These capabilities primarily center on the lifecycle of the application state. For example, consider a transport scenario in which a shipment consists of several legs. The shipment itself would need to be created. Each leg would need to be created. Events may need to be injected (i.e., simulated) for one of the legs. The shipment ID and leg ID would likely need to be included in the event. Thus, the execution of the experiment would need to be able to access this data, and indeed the application should provide that data as part of the shipment and leg creation. To support such scenarios, the application UI needs to provide additional capabilities within the test or experimentation context. Specifically, the UI should be able to receive the access details for exposed data providers, and the application logic should receive or set data as needed. The links to the application UI are provided through data providers. Additionally, for any application data that is managed by the application (for example, data that changes dynamically within Flspace as a result of events) and that is necessary during experiment execution, the application will need to provide a data access system for connecting to the application.

3 Experimentation environment development plan

WP300 is accountable for the "Flspace Hosting and Experimentation" of the project. WP300 in Flspace is a straightforward continuation of the work accomplished in Flnest WP4 "Experimentation Environment". In addition, Flspace WP300 also includes the deployment of the platform components and FI-WARE² Generic Enablers (GEs) in a cloud infrastructure. Flspace WP300 incorporates Flnest WP4 results and takes Flnest EE specification as the starting point upon which to build.

As part of the WP300 work in Flspace we will build an internal cloud infrastructure in which the Flspace components as well as the use cases will be deployed, and the EE will operate. The eight use cases of the Flspace project will be specified in a way that enables their physical testing or simulation in the experimentation environment of the project. In the latter case, historical real data will be used for simulation, thus enabling an environment as close as possible to the real environment.

In total, eight use case trials have been specified for Flspace, organized along three themes:

(A) **Farming in the Cloud** addresses food production issues at the farm level and covers two use case trials:

- Crop Protection Information Sharing – Using field sensors and satellite data to intelligently manage the application of pesticides for maximum crop protection
- Greenhouse Management and Control – Using sensors to monitor key growth factors (UV radiation, moisture and humidity, soil conditions, etc.) and to feedback data to control systems to modify the growth environment for maximum yield and optimal quality

(B) **Intelligent Perishable Goods Logistics** addresses monitoring and environmental management issues of perishable goods as they flow through their supply chains so that waste is minimized and shelf life maximized covering three use-case trials:

- Fish Distribution and (Re-)Planning – Focuses on the planning of logistics and transport activities, including transport order creation, transport demand (re)planning, and distribution (re)scheduling
- Fresh Fruit and Vegetables Quality Assurance – Looks at the management of deviations (transports, products) that affect the distribution process for fresh fruit and vegetables (transport plan, food quality issues), either deviation from the plan or other external events requiring re-planning
- Flowers and Plants Supply Chain Monitoring – The monitoring and communication of transport and logistics activities focusing on tracking and tracing of shipments, assets and cargo, including quality conditions and simulated shelf life, focusing on cargo and asset quality tracking ("intelligent cargo"), shipment tracking ("intelligent shipment"), and lifecycle information tracking of cargo characteristics/cargo integration along the chain

(C) **Smart Distribution and Consumption** is about helping consumers to obtain better information on the goods they purchase and helping producers to better control the flow of their goods to the consumer, covering three use-case trials:

- Meat Information Provenance – Ensuring that consumers, regulators, and meat supply chain participants all have accurate information concerning where a meat product originated (production farm) and how it was affected by its distribution (quality assurance)
- Import and Export of Consumer Goods – The intelligent management of inbound materials to a production site and the smart distribution of finished goods to consumers
- Tailored Information for Consumers – The provisioning of accurate information to individual consumer's needs and feedback of this information to the producers

Figure 3 depicts the different Tasks to be accomplished by WP300 in Flspace.

² <http://www.fi-ware.eu/>

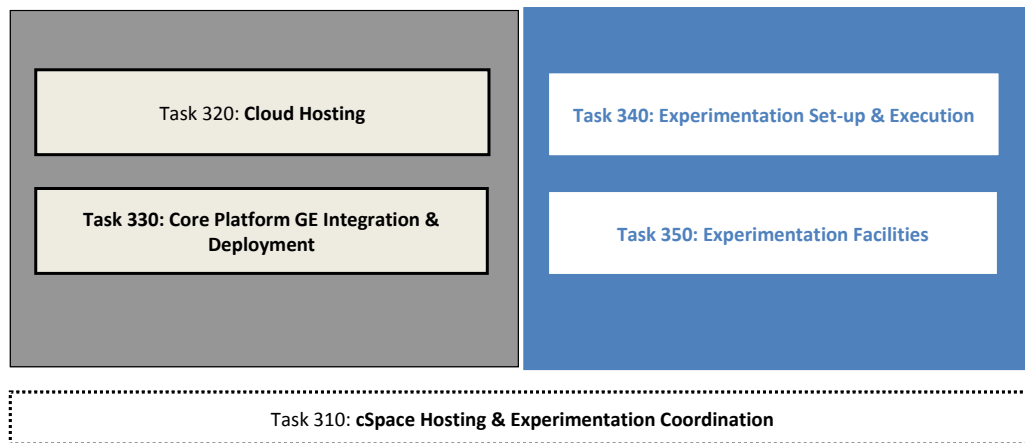


Figure 3: WP 300 Flspace Hosting and Experimentation

More specifically, the following Flspace WP300 tasks ([Task 340](#) and [Task 350](#), respectively) will deal with the experimentation environment follow up of Flspace EE:

Experimentation set-up and execution – This task objective is to provide support for the actual execution of the use cases scenarios in the experimentation environment.

Experimentation facilities – The objectives of this task are twofold: to provide an EE to test the provided new services using real data and physical sites, as well as simulation environment for the testing execution and to provide means to facilitate the analysis and assessment of Flspace new collaborations performance as reflected in the use-case scenarios.

Experimentation set-up and execution task

This task is concerned with the actual execution set-up and support of the use-case trials (test scenarios) specified in the Flspace project. In essence, this task supports the experimentation process as shown in Figure 4.

This task is further divided into two subtasks:

- **Experiment design and configuration** (M1- M15) – This subtask focuses on making the use-case trials defined throughout Flspace executable, meaning that they can be run either manually or automatically (using scripts) in the experimentation environment. This subtask corresponds to the *experimentation design and configuration* phase in the figure (and denoted as a **solid line**).
- **Experiment execution and analysis** (M9 - M24) – This subtask deals with the actual execution of the different steps of the use cases trials and the analysis of the outcomes. This subtask corresponds to the *experiment execution and analysis* phase in the figure (and denoted as a **dotted line**).

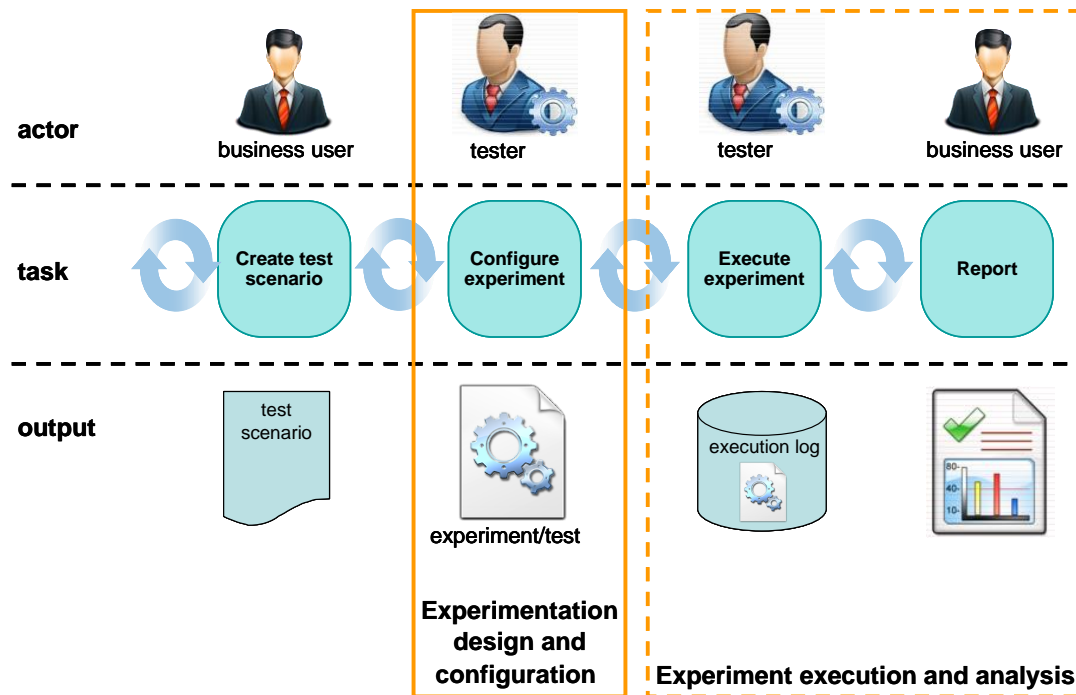


Figure 4: Experimentation process

Experimentation facilities task

This task is concerned with the scaffolding and interfaces that are required to have an environment that is as-close-as-possible to the actual real-life environment. The starting point of this task is the experimentation environment specification detailed in Section 2.

This task is further divided into three subtasks:

- **EE test (M1-M9)** – This subtask is concerned with putting a test bed in place for Flspace in which the use case trials will be carried out, including simulation capabilities, namely, the test components in Figure 1.
- **EE core (M6-M21)** – This subtask is concerned with the development and support of all components required to run and analyze experiments executions, namely, the EE components in Figure 1.
- **EE front-end (M6-M21)** – This subtask is concerned with the development of the user interface to enable the use of the EE to create, update, execute, and report of tests, namely, the EE front-end in Figure 1.

The aim is to have three releases of the EE resulting in three incremental versions of the EE. Furthermore, these releases will be in full synchronization with the milestone releases of the project and with the work package deliverables as described in the DoW.

In the future, we plan to run our scenarios in environments provided by XiFi. Our emphasis is on physical sites equipped with IoT (Internet of Things) sensors and realtime data that can be obtained from those sensors, especially in the domains of transport and logistics and agri-food. Ideally, we would like to have environments that support point-to-point scenarios (e.g., flight routes or ship itineraries).

4 Experimentation environment scenarios execution plan

This section provides the guidelines for Flspace scenarios execution and demonstrates the process on the example of “Greenhouse Management and Control” scenario. The goal is to show, based on a concrete example, how the hosting environment and the experimental environment of Flspace will work together with the components of Flspace platform (WP200) and the trials (WP400).

To simplify and standardize our work, we recommend a partial adoption of the IEEE 829 standard for testing Flspace scenarios as described in 4.1. Section 4.2 illustrates the use of the template on example of “Advice Request” (Trial 422), and Section 4.3 gives the overview of the entire scenario execution process using the same trail example.

4.1 Scenario execution template

Testing of software systems is a well-established field. Therefore, we have investigated exiting standards to adopt them for testing of the Flspace scenarios. The IEEE 829-2008 standard for Software and System Test Documentation specifies the form of a set of documents for defined stages of software testing. The following is an overview of the relevant parts of the standard (we will use the short name appearing in **parenthesis** after the item in the rest of the discussion):

- **Test Plan (LTP):** This is a management planning document that shows how the testing will be done, who will do it, what will be tested, how long it will take, and what the test coverage will be, such as what quality level is required.
- **Test Design Specification (LTD):** These specifications detail the test conditions and the expected results as well as test pass criteria.
- **Test Case Specification (LTC):** This specifies the test data for use in running the test conditions identified in the Test Design Specification
- **Test Procedure Specification (LTPr):** This details how to run each test, including any set-up preconditions and the steps that need to be followed.
- **Test Item Transmittal Report (LTISR):** This reports on when tested software components have progressed from one stage of testing to the next.
- **Test Log (LTL):** The test log records which tests cases were run, who ran them, in what order, and whether each test passed or failed.
- **Test Incident Report (LTIR):** This report describes, for any test that failed, the actual versus expected result, and other information intended to throw light on why a test has failed, and it may include an assessment of the impact of an incident upon testing.
- **Test Summary Report (LTSR):** This is a management report providing any important information uncovered by the tests accomplished, assessments of the quality of the testing effort, the quality of the software system under test, and statistics derived from Incident Reports. This final document's purpose is to indicate whether the software system under test is fit for purpose, according to whether or not it has met acceptance criteria defined by project stakeholders.

Based on the standard, we derived a template that we suggest to use for the planning and execution of Flspace scenarios.

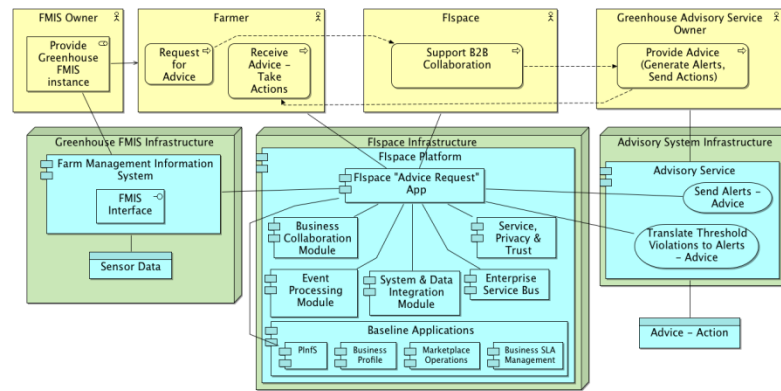
| | | | |
|----------------------------|--|--------------------------|--------|
| | | | |
| Test Started: (LTP) | (Date) | Test Ended: (LTP) | (Date) |
| Partners: (LTP) | NKUA, Innovators | Work Package: | WP 400 |
| Status: | Defining requirements/ started/ finished (1st test) etc... | | |

| | |
|--|--|
| | |
| KPIs / Pass-Fail Criteria (LTD) | Provide criteria that will determine the final outcome of the test |
| Final outcome: | Passed / Failed |
| Test Input / Output Data (LTC) (Figure 14 in FInest D2.5 doc) | Input: Output: |
| Flspace Modules Involved | Provide Flspace components involved in the particular scenario |
| Preconditions (LTPr) | Provide conditions that should be met, before the start date of the experiment (e.g., Mediator GE must be configured, deployed and running) |
| Requirements (LTPr) | According to the standard, "shall" corresponds to obligatory requirements, while "should" refers to requirements of secondary importance The FMIS shall be already instantiated and connected to the Flspace platform The B2B collaboration module shall have been set-up accordingly to support the communication between the PinFS Baseline App and the FMIS service instance |
| Actions/steps to be performed (LTPr) | The user opens the corresponding Flspace app... When the "New notification" icon is active, he clicks on it. The user gets the input from ... By the time he clicks on the message... |
| Expected Results corresponding to above action numbers (LTD) | To be filled before the start date of the experiment |
| Chronological record of details of actual events (LTD) | To be filled during and after the experiment has completed Provide the chronological record of relevant details about the actual execution of tests. |
| Results (LITSR/LTR) | Actual outcomes of the experiment. To be compared with the expected results form LTD above |
| Anomaly Report | Provide for tests that failed the actual result, the reason why the test has failed, and if possible, the impact of the Anomaly Report upon testing |

4.2 Advice Request scenario execution plan

This section illustrates the use of the template described in the previous section with the Advice Request scenario from the Greenhouse Management & Control (Trail 422) as an example. The example of the execution plan that is given in this section is not complete (as the work on this scenario is still in progress) and is given just for the demonstration purposes.

| | | | |
|-----------------------------|---|------------------------|--|
| Test Case Name: | Greenhouse Management & Control Trial | Test Case Type: | Use Case Trial |
| Created: | 1/4/2013 | Last Change: | 16/9/2013 |
| Partners: | NKUA, Innovators | Work Package: | WP400 |
| Specific Use Case: | "Advice Request" scenario | Status: | <ul style="list-style-type: none"> • Business Process created • Functional requirements defined • GSM modeling of the Advice business entity created • Identifying required interfaces/data models with legacy systems |
| Description/Purpose: | <p>Business Layer</p> <p>The sensors' values of the Greenhouse are forwarded via the Greenhouse Farm Management System (FMIS) to the Advisory (Expert system) via Flspace. The farmer is using the Advice Request App to view the advices and actions to take, provided by the expert system, to maximize the efficiency of the Greenhouse's production.</p> <p>Technology Layer</p> <p>The Farmer uses the Flspace Advice Request App using a specific GUI to handle the monitoring of the greenhouse sensors. Apart from the constant monitoring choice and the advice/actions based on these values, the farmer has the option to create a manual advice request, by inputting the appropriate text. The Agrosense Service, which is handling the sensor data, forwards it to Flspace. In particular, the core module of Flspace, which receives and monitors the sensor values, is the Event Processing Module (EPM). Whenever a threshold violation is detected, according to pre-specified rules, the EPM forwards the event to the Expert System. All generated values (including the normal ones that do not exceed any threshold) are stored inside the FMIS for analysis, graphical representation to the farmer, etc. The Expert System based on the particular events received (generated from the threshold violations from EPM), translates them to the appropriate alerts, advice and actions to be taken and sends them back to Flspace, which forwards this information for the farmer back to the FMIS service, as well as to the Flspace Advice Request app's GUI.</p> | | |



Prerequisites

- Re-

- The business process has already been created inside the platform (BCM) and has been linked to the Advice Request App
- Flspace Advice Request App is already set up and configured to the specific external systems
- The External System adapter (T250) is configured and supports the incoming / outgoing traffic for the Greenhouse FMIS and the Expert System
- The Greenhouse FMIS is configured and the sensor values are being forwarded to Flspace's EPM module through the first adapter
- The expert system is connected to Flspace's EPM and is able to receive the generated events as well as forward back the generated advice through the second adapter
- EPM is configured with the event rules (sensor values' thresholds) according to the GSM modeling of the "Advice" business entity
- Business Collaboration Module is configured for the specific Business Process
- ...
- ...

Actions:

Test Scenario 1:

1. Manually create a threshold violation event (either using the Backend Simulator component of the Flspace Experimentation Environment or manually "mislead" a particular sensor inside the Greenhouse, e.g., temperature to generate exception)
2. The user logs in Flspace and opens the Advice Request app (pre-installed for the particular user)
3. The user clicks on the notification
4. The user views the alert related to the temperature raise, and views the advice-actions that the expert system generated
5. The user gives feedback on the

Expected Results:

Test Scenario 1:

1. The EPM detects the temperature threshold violation and generates an event, which is sent to the expert system; the expert system generates advice/actions based on the input from the EPM and then forwards it back to the Advice Request App
2. A notification shows up when the user logs into Flspace and opens the Advice Request App
3. The alert as well as the actions (advice) from the expert system are presented to the farmer
4. -
5. The expert system receives via Flspace the user's feed-

| | | | |
|---------------|---|--|---|
| | expert system's advice | | back and updates the advice algorithm for the particular user |
| Report | <p>Log example for Test Scenario 1: (by timestamp - TS)</p> <p>TS 1: [Greenhouse FMIS]: <i>sending sensor values...</i></p> <p>TS 2: [Flspace EPM]: <i>Greenhouse sensor values received – “All values within accepted boundaries”</i></p> <p>TS 3: [Greenhouse FMIS]: <i>sending sensor values...</i></p> <p>TS 4: [Flspace EPM]: <i>Greenhouse sensor values received – “All values within accepted boundaries”</i></p> <p>TS 5: [Greenhouse FMIS]: <i>sending sensor values...</i></p> <p>TS 6: [Flspace EPM]: <i>Greenhouse sensor values received – “Temperature HIGH!”</i></p> <p>TS 7: [Flspace EPM]: Sending event to the expert system</p> <p>TS 8: [Flspace BCM]: Forwarding event to the expert system</p> <p>TS 9: [Expert System]: Event received. Generating advice...</p> <p>TS 10: [Expert System]: Sending advice back to Flspace...</p> <p>TS 11: [Expert System]: Advice sent.</p> <p>TS 12: [Flspace BCM]: Advice received from Expert System</p> <p>TS 13: [Flspace BCM]: Forwarding advice to Greenhouse FMIS</p> <p>TS 14: [Greenhouse FMIS]: New advice received</p> <p>TS 15: [Flspace Advice Request App]: New advice received</p> <p>...</p> <p>...</p> <p>TS t: [Flspace Advice Request App]: Notification consumed by user</p> <p>...</p> <p>...</p> | | |

4.3 Advice Request Scenario setup and execution process

In this section, we describe the execution process for Advice Request scenario associated with the Trial 422 Greenhouse Management and Control. We show how the hosting environment and the experimental environment of Flspace can work together with the components of Flspace platform (WP200) and the trials (WP400). As illustrated in Figure 5, three roles (farmer, Greenhouse Management, Expert) and three systems (Flspace Platform, Farm Management System (FIMS), and Expert System) are involved in the Advice request scenario.

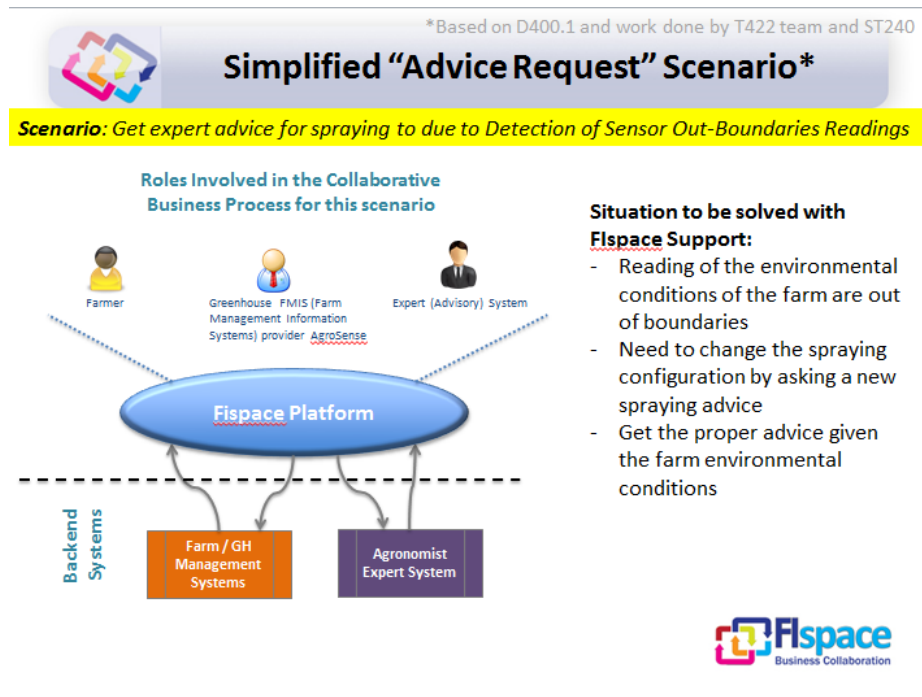


Figure 5: Simplified Advice Request Scenario

The situation to be experimented in this scenario involves:

- Checking if the reading of experimental conditions of the farm are out of boundaries
- If an out of boundary situation is identified, steps necessary to request new advice about the spraying parameters must be activated
- Receive the new advice and send it to the farmer system

The main actions performed by each role are:

Farmer:

- Visualize in Flspace Frontend the content of the computed advice

AgroSense (Farm Management System - MIS)

- Send events to Flspace with sensor consolidated information
- Receive computed spraying advice when out-boundary sensor readings was detected

Expert (Advisory) System

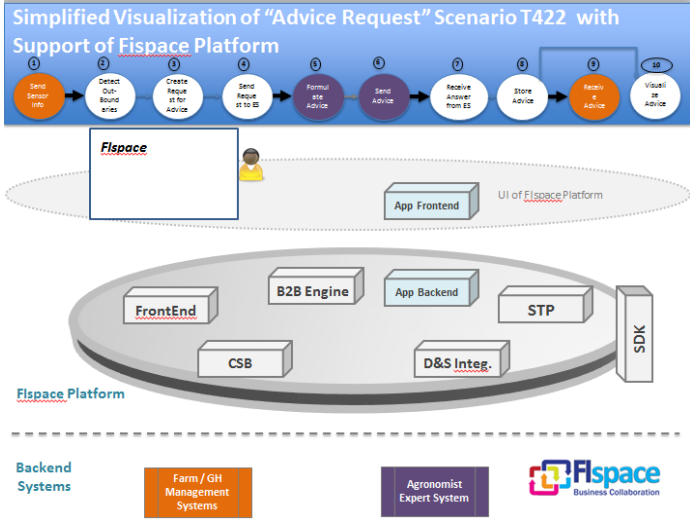
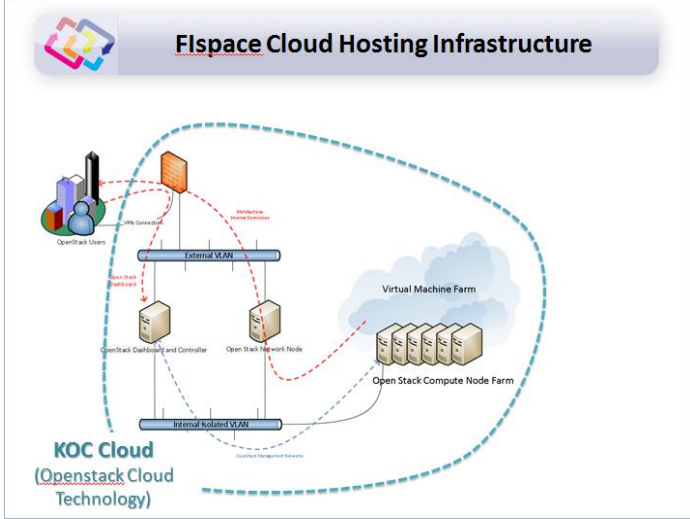
- Receive request for creating an spraying advice
- Compute the advice
- Answer request with computed advice

The next sections describe how the functionalities offered in WP300 will support the deployment and execution of experiments associated with this scenario.

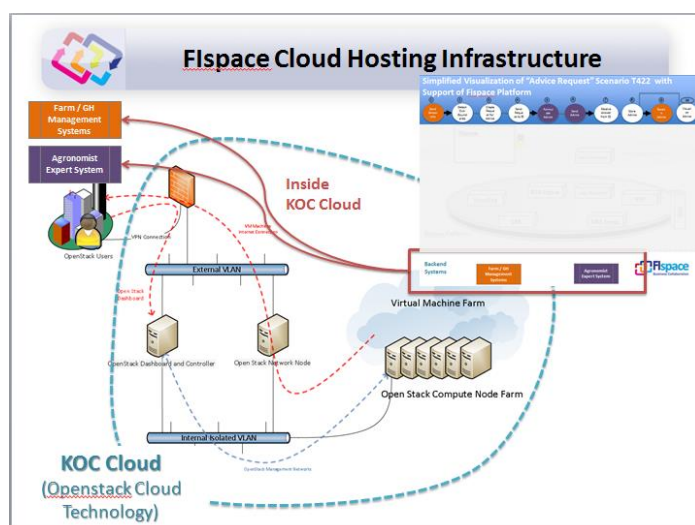
4.3.1 Flspace Cloud Hosting Infrastructure

The Flspace cloud hosting facility has been defined in D300.2. In this section, we describe how this hosting facility supports the deployment of the software components and the roles involved in the Advice Request scenario. The details of this support are illustrated and described in Table 1.

Table 1: Description of Flspace Cloud Hosting for Advice Request Scenario

| Screenshot | Description of Situation |
|---|---|
|  | <p>A) Software support from Flspace</p> <p>The execution of the advice request scenario demands the software components illustrated in the figure on the left. There are backend systems that are not part of Flspace Platform and there are the components that are part of Flspace Platform. In addition, there is also the tester, which in this case is the farmer interested in receiving the new advice.</p> |
|  | <p>B) Illustration of the hosting environment</p> <p>The illustration on the left show the architecture of the cloud hosting facilities (Details in are available in D300.2). In the Advice Request scenario there systems and software components that should be deployed inside the Flspace cloud hosting facilities and there will be others deployed and used by users/tester that are outside the hosting environment.</p> |

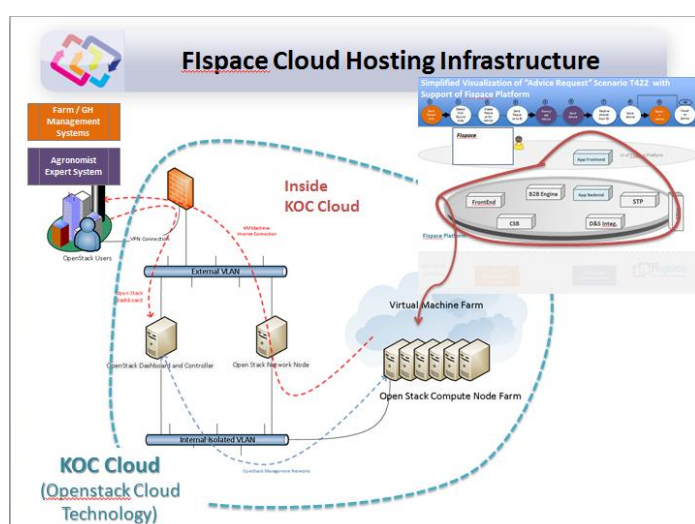
Screenshot



Description of Situation

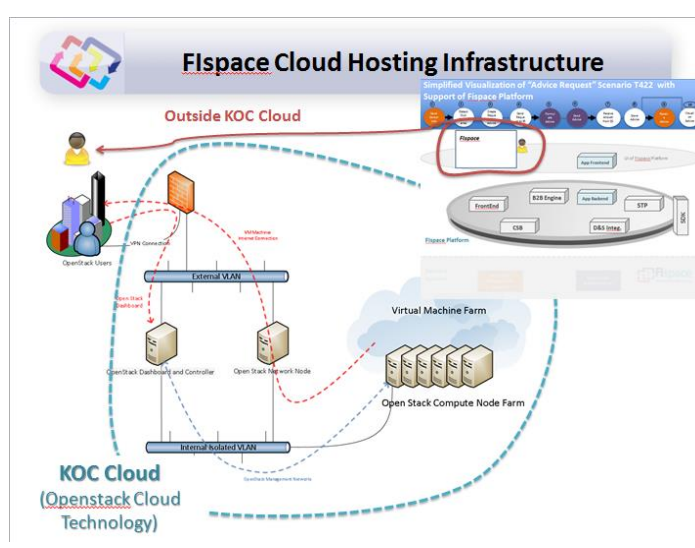
C) Backend Systems placement

All backend systems required by any scenario will not be deployed inside the hosting facilities of WP300. These are external systems and are kept outside the KOC Cloud. In the Advice Request scenario, the Farm Management Information System (FMIS) and the Expert System are examples of system that will not be deployed inside the cloud hosting, but they will be accessed remotely.



D) Flspace software components hosting

All software components developed in the core Flspace Platform (WP200) and for supporting the trials will be hosted inside the KOC cloud, i.e., inside the Flspace hosting facilities.



E) Tester of the scenario

All users or testers of a scenario will access the Flspace platform through the Internet. In the Advice Request scenario, the user (in this case the farmer) will login inside Flspace from his/her notebook or smartphone (both outside the cloud hosting environment).

4.3.2 Experimental Set-up

To support the proper execution of the trial experimentation, we created a template based on IEEE 829 standard, as indicated in 4.1. This template contains the set of documents that need to be generated by the trials in order to help the set-up of the experiments in WP300. In this section, we present a summary in Table 2 of how these some of these documents are used in each one of the phases defined for the experimentation set up of the Advice Request scenario.

Table 2: Description of Experimentation Set-up for Advice Request Scenario

| Screenshot | Description of Situation |
|------------|--|
| | <p>F) Definition of Test Scenario</p> <p>The definition of the scenario to be tested is a task conducted by each Trail in WP400. The result of this work is documented according to the template detailed in 4.1. The first set of information from the template (e.g., description/purpose) is important to give the overall scope of the trial. They help to define which kinds of roles are expected to be used in the scenario and the goal of the test in this scenario. The figure on the left summarizes and illustrates the details presented in 4.2. For the Advice Request scenario 3 roles are addressed: farmer, FMIS System and Expert System</p> |
| | <p>G) Experimental Protocol</p> <p>The precise definition of what needs to be configured and designed for the experimentation of the scenario is described in the Experimentation Protocol. The experimentation protocol comprises the following fields of the template: Prerequisites-Requirements, Actions, Expected Results and Report. This information is essential for configuring the scripts that will run step by step the scenario.</p> |

Screenshot

Flspace Experimentation Set-up & Execution (Task 340)

Information needed to run the experiment

- Sensor information from Meteo service's sensors
- Advice from Expert System (NKUAEs)

| Type of information | Measurement | Valid measurements range |
|---------------------|-------------|----------------------------------|
| Air temperature | °C | -40 C up to 460 C |
| Air humidity | % | 0% up to 100% |
| Wind velocity | m/s | 0m/s up to 360m/sec |
| Water radiation | w/m2 | 0 w/m2 up to 2000 w/m2 |
| Rain | mm | 0mm up to 100mm for 10 min step |
| Barometric pressure | mmHg | 300mmHg - 1150mmHg |
| Ground humidity | % | 5% - 60% |
| Period of sunshine | min | 0min up to 10min for 10 min step |

Experiment execution and analysis

tester

business user

Execute experiment

Report

execution log

Report

Description of Situation

H) Information needed to run the experiment

In addition to the definition of the steps and expected results. It is also necessary to indicate which kind of information, formats and values each scenario requires. For example, for the Advice Request Scenario the definition of the sensor information propagated by the FMIS system is essential. Based on this information Flspace platform will be able to detect out boundaries sensor information and then start the collaborative process between farmer, FMIS and Expert System to get a new spraying advice according to the new conditions of the farm environment reported in the sensor information.

4.3.3 Experiment Execution

Flispace Experimental Environment provides basically two ways for running the experiments—simulated data streams and real-time data streams.

In the first case, the real data associated with the situations to be tested in the trial are collected and stored in a database previously to the execution of the experimentation. Then, at the moment of conducting the experiment, the “Backend Simulator” component from the FIspace Experimentation Environment framework reads this database and pushes the simulated the data stream into the trial experiment execution as if the data was coming from external systems. Table 3 illustrates the use of simulated data in the Advice Request scenario.

Table 3: Description of Execution for Advice Request Scenario with Simulated data

Using Experimentation Facilities to Execute Experiments on “Advice Request” Scenario T422

A) Overview of Experimentation facilities

For each scenario in the trial a technical sequence of steps will be associated with such scenario. The definition of these technical steps is not part of WP300 or WP400 but they are implicitly created once the software components associated to the scenario are used together. In the figure on the left, the sequence of steps of the Advice Request scenario is illustrated. The orange steps represent the interaction with the FMIS system, the white ones are interaction inside Fspace Platform, and the violet steps indicate interaction with the Expert System. As it will be described in the item B of

| Screenshot | Description of Situation |
|---|--|
| <p>Using Experimentation Facilities to Execute Experiments on "Advice Request" Scenario T422</p> <p>Simulated Streams of Data</p> <p>The diagram illustrates the data flow and system architecture for the 'Advice Request' scenario. The top flowchart shows the sequence of events: 1. Send Sensor Info, 2. Collect Data, 3. Create Request for Advice, 4. Send Request to ES, 5. Receive Advice, 6. Send Advice, 7. Process Advice from ES, 8. Store Advice, 9. Produce Advice, 10. Visualize Advice. The architecture diagram below shows the Fspace EE frontend (Controller, Access Control, etc.) interacting with the Fspace Experimentation Environment (Data Manager, etc.), which in turn interacts with the Backend (Data Storage, etc.). A red circle highlights the 'Fspace components automatic execution' step in the Backend.</p> | <p>this table, the actual interaction with the orange and violet steps will be simulated.</p> <p>B) Backend Simulator Component pushing information into Flspace Platform</p> <p>In the Advice Request scenario, the FMIS system sends events with sensor information to Flspace Platform. These sensor values are checked by the Flspace Platform. If they are out of expected boundaries the Flspace start the process to request an advice to the Expert System.</p> <p>In the simulated execution of this scenario experimentation, the events with sensor information are stored inside the "backend Simulator" and are pushed into the Flspace test environment. Thus, a simulator component will be available in the experimentation facility of Flspace to enact this process and there will be no connection to the FMIS outside Flspace hosting facilities (as illustrated in Table 1 – C)</p> |
| <p>Using Experimentation Facilities to Execute Experiments on "Advice Request" Scenario T422</p> <p>Simulated Streams of Data</p> <p>The diagram illustrates the data flow and system architecture for the 'Advice Request' scenario. The top flowchart shows the sequence of events: 1. Send Sensor Info, 2. Collect Data, 3. Create Request for Advice, 4. Send Request to ES, 5. Receive Advice, 6. Send Advice, 7. Process Advice from ES, 8. Store Advice, 9. Produce Advice, 10. Visualize Advice. The architecture diagram below shows the Fspace EE frontend (Controller, Access Control, etc.) interacting with the Fspace Experimentation Environment (Data Manager, etc.), which in turn interacts with the Backend (Data Storage, etc.). A red circle highlights the 'Fspace components automatic execution' step in the Backend.</p> | <p>C) Flspace Platform sending information to Backend Simulator Component</p> <p>In the Advice Request scenario, the Flspace Platform has to send a message to the Expert System (external system as illustrated in Table 1 – C) requesting a new advice. This process is also simulated in the experimentation facilities as depicted in the figure on the left. The request is send indeed to the Backend Simulator Component that answers back to the Flspace Platform as if it was the Expert System. To enable this behaviour it is necessary also to store which answers the Expert System would answer given a set of information coming from the Flspace Platform. This information should be defined and collected previously by the trial owners, so that it can be simulated inside the experimentation facilities.</p> |

In the situation of supporting real-time data streams during the experimentation, the Flspace test environment is directly connected to the external systems generating the stream of data to be consumed in the experiment. In this case, there is no need for previously storing the data to be used in the experiment. This data will be generated in real time by the environment of the trial. Table 4 illustrates how it would look like to use realtime data stream experimentation in the Advice Request scenario.

Table 4: Description of Execution for Advice Request Scenario with Real Stream of Data

| Screenshot | Description of Situation |
|------------|--|
| | <p>A) Receiving Information from Backend Systems</p> <p>When the experimentation of the Advice Request scenario is executed with real-time data stream, the FMIS system deployed outside Flspace hosting environment is actually connected to the Flspace Platform. This allows the system itself to send the events with sensor information, thus removing the role of the Backend Simulator component of the Experimental Environment.</p> |
| | <p>B) Sending Information to Backend Systems</p> <p>The Backend Simulator component is also not used in the case that the Flspace Platform needs to send messages to the Expert System (i.e., request the advice). Again, this system is actually used and connected to reach the Flspace Platform.</p> |

5 Execution Environment architecture

This section gives a detailed overview of the Execution Environment architecture as illustrated in Figure 1.

5.1 Main terms

The proposed technical specification of the EE enables the entire process, from test/experiment planning and configuration, through execution, to analysis of the test execution. We introduce below terms to be used throughout this report.

Step – A step refers to a single action/task defined in a test scenario.

Test scenario – This is the ordered set of steps that compose a single test.

Variables – In the context of a test, these are field names that stand for specific values during execution. Variables enable flexibility in test execution, as they enable running the same test with different field values.

Variables binding – This refers to the replacement of variables values with the test data. This is done by the experimenter during test execution.

Experiment/test – This is the ordered set of steps to be carried out by an experimenter during execution. Each experiment is identified by a unique ID and version. An experiment may have variables to enable multiple executions of the same experiment with different data.

Execution – This is the actual running of an experiment. All variables should be bound to Data Providers before the execution of steps can begin.

Vusers – These are virtual users that play human users in a specific experiment.

Vusers scripts – These scripts are the ordered set of actions a Vuser performs during the execution of an experiment. In other words, the set of instructions carried out during execution without user intervention.

Atomic step – This is the smallest (inseparable) single instruction that is carried out during the execution of a test. An atomic step may contain (a) an instruction to be manually performed by a tester; (b) a reference to run a Vuser script; or (c) an instruction to inject data provided through a variable into Flspace test.

Execution log – This log is a file that lists actions as occurred during execution, including all process and system notifications. The entries in an execution log can provide insight into what happened during execution of the test and provide an audit trail of information related to the execution. In fact, the execution log is the input to the *Reporting* module in the EE which analyzes the log and provides performance assessment of the execution.

Expected results – This is the anticipated outcome of a step in a test.

Actual results – These are the real outcome of a step as result of execution.

Key Performance Indicator (KPI) – The KPI is a set of performance measurements related to T&L stored in the EE, for the sake of performance assessment and analysis. The evaluation framework specification is in the scope of Work Package 2, but the KPI(s) related to the performance assessment are stored in the EE, and can be used to assess the performance of the test executed.

Composite Key Performance Indicator (CKPI) – This is a KPI composed of one or more KPIs jointly analyzed.

Report – The report is a summary of what occurred over one or more test executions. A report may include performance assessment of the execution based on given KPI(s).

Injected data – Data fed into the test by the *backend simulator* module in EE is injected data. Injected data is used whenever real data in real time cannot be obtained during the execution of a test. In these cases, the intention is to use (realtime) historical data to simulate the processes.

Notifications – These are messages given to a user via Flspace frontend during an execution of a test. Notifications are recorded in the execution log of the test.

5.2 Experimentation environment components

In this section, we describe the Flspace EE components:

User Manager. The User Manager handles user accounts, passwords, and access. This includes features such as user groups and access control to data, as well as users being able to assign other users permissions.

Experiment Manager. The Experiment Manager handles experiment lifecycle and experiment querying. This includes creation, versioning, archiving, and search capabilities.

- *Experiment CRUD* (Create, Read, Update, and Delete): Provides services for experiment lifecycle, using archiving instead of deletion so that traceability is never lost
- *Experiment Search*: Provides services for finding experiments according to various search criteria

Execution Manager. The Execution Manager handles the concrete executions of an experiment. This includes the creation of new executions (including the configuration of variables), executing (or tracking the execution of) the steps in the experiment, and logging the results.

- *Executor*: The Executor tracks the execution of the individual steps in an evaluation. This includes the automated execution of certain steps, such as injecting data/events into the test instance and running VUser scripts through the *Script Execution Engine*. This component also creates and updates entries in the execution log, including notifications received from the actual process execution and error messages.
- *Script Execution Engine*: This executes VUser scripts to automate user actions.

Resource Manager. The Resource Manager provides an inventory of available resources. Services include the ability to locate resources according to various search criteria.

Reporting: Reports are generated based on execution logs and KPIs.

- *KPI Manager*: Manages the calculation and composition of KPIs
- *KPI Composer*: Used to create and manage composite KPIs

Execution Log Manager. The Execution Log Manager provides logging services for an execution. This includes the logging of the results for each step of an execution, including any received notifications during the execution of each step. Also provides access to these logs.

Execution Data Manager. This is used to manage the access to data that is used during execution.

- *Internal Data Provider System*: Used for storing and retrieving manually configured data providers
- *External Data Access*: Used to retrieve data from 3rd party external systems, could be used to “replay” events from a real-world shipment, for example

Access to these systems is configured by the tester. Configuration could be UI or file driven.

Backend Simulator. This simulates input data from backend systems to Flspace and provides APIs to inject data to the Flspace Test system’s modules. The simulator reports back on events and other processed data.

EE Storage: This provides internal storage services for the experimentation environment.

Data Provider System (app) – This component addresses the binding of the application to the execution data.

5.3 Data types definitions

The data types are described below. Note that additional methods are included for convenience. While not mentioned for brevity, getters have associated setter methods as well.

5.3.1 Experiment (DataType)

Note that once an experiment-version has associated executions it cannot be modified, although new versions can still be created for the experiment.

| Method | Notes | Parameters |
|--|--|--|
| DefineVariable() void Public | Used to define a new variable which must be bound for use during execution; The variable's name must be unique within the experiment. | VariableSpecification |
| AddStep() void Public | Insert a new step to the experiment | Step int – where to insert |
| RemoveStep() void Public | Removes a step from the experiment | int – where to remove |
| ReplaceStep() void Public | Replaces a step in the experiment | Step int – where is the step to be replaced |
| GetSteps() Step[0..*] Public | Gets the steps for this experiment | |
| GetVariables() VariableSpecification [0..*] Public | Gets the variables defined for the experiment | |
| GetVersion() int Public | Gets the version number for the experiment | |
| GetExperimentId() GUID Public | Gets the experiment ID; this is common to all the different versions of an experiment | |
| GetId() GUID Public | Gets the global unique ID for this experiment and version | |
| GenerateCopy() Experiment Public | Creates a copy of this experiment; This is a deep copy—changes to this experiment should not affect the copy and vice-versa; The copy is not in persistent storage | |

5.3.2 VariableSpecification (DataType)

A VariableSpecification instance gives a type of data that needs to be provided when creating an Execution instance for an Experiment.

| Method | Notes | Parameters |
|---|--|------------|
| GetType() DataType Public | Returns the data type. May be int, String, long, HTML, TCP, Event, ContractStatus, Link, etc. | |
| GetCardinality() DataCardinality Public | Returns the necessary cardinality. Cardinalities are characterized by a minimum value (which is at least 0), | |

| | | |
|---|---|--|
| | and a maximum value (which is at least 1), which may be unbounded. Examples are: 1..1, 0..1, 2..*, 0..5, 1..* | |
| GetDefault- DataProvider DataProvider[0..1] Public | Returns the default data provider for binding | |
| GetDescription() String Public | Gets the human-readable description of the variable and what it is used for in the experiment | |
| GetName() String public | Gets the human-readable name of the variable. | |

5.3.3 Step (DataType)

| Member | Notes | Type |
|-----------------------|--|------------------------------------|
| Actor | Sets the actor to perform the action; This can be a user, a role, or a system | String |
| VUserScript | A VUser Script to be used when executing | String – the script to be executed |
| DataInjectionVariable | Stores a variable name, whose DataType should be injectable into Flspace (such as Transport Execution Data, Event, Booking); When executing, the bound data provider will provide the data to be injected; Data is injected at the beginning of execution | String |
| Link | Sets a link to be resolved during execution, which the user should click to access the Flspace UI or application specific UI; Such links will often contain variables to pass through to the application which can be used by the application to update data providers; If necessary, the application should provide an appropriate Test UI component for handling this pass-through data (or the standard UI could be configured for “test mode”) | Link |
| Registrations | Registrations for events and notifications to be made after the execution of this step; The strings should conform to Flspace formats; Variables can be embedded using \$varName notation (\$\$ is used to represent a single \$); At runtime, the variables’ values will be substituted (similar to how link URIs are resolved) | String [0..*] |

| | | |
|-----------------|---|--------|
| DataDescription | Sets the description of the data to be used during the step | String |
| Description | Sets the description of the action to be taken during the step | String |
| ExpectedResult | Sets the expected result of the execution (a human-readable string) | String |

5.3.4 Execution (DataType)

| Method | Notes | Parameters |
|--|--|--|
| GetExperiment() Experiment Public | Returns the experiment instance this execution is associated with. | |
| BindVariable() void Public | Binds a variable name with a data provider | String – variable name DataProviderId |
| GetVariableBindings Map<String,DataProviderId> Public | Returns a mapping from variable names to their bound data providers | |
| GetCursor() int Public | Returns the index of the next step not completely executed (from 0 to total number of steps) | |
| IncrementCursor() void Public | Increases the cursor by 1 | |
| GetId() GUID Public | Gets the global unique ID for the execution | |
| GetCreator() UserId Public | Returns the user id of the creator of this execution | |
| GetStatus() ExecutionStatus Public | Returns the status of the execution, one of: Initializing, In Progress, Complete, Aborted, Cancelled | |

5.3.5 Resource (DataType)

| Member | Notes | Type |
|-------------|--|--------|
| ID | Gets the resource's ID | GUID |
| Description | Gets a human-readable description for the resource | String |
| Name | Gets the human-readable name for the resource | String |

5.3.6 Link (DataType)

| Member | Notes | Type |
|----------------|---|--------|
| URI | Gets the URI; The URI embeds variable access by using \$varName (during execution \$varName will be replaced with the variable's current value (for variables with cardinality > 1 this is a list); \$\$ is used to encode a single \$; If additional data is not null, then the uri also embeds an additionalData parameter containing a URI (which included the execution id) for retrieving the additional data | String |
| AdditionalData | Application specific additional data to be retrieved; Embeds variable access by using \$varName (which is substituted upon retrieval with the variable's current value); \$\$ is used to encode a single \$; To pass the URI of an exposed data provider use \${varName} (instead of passing its value); The use of the additional data field is intended to prevent the URI member from exceeding possible size limitations; For example, this field is used to pass the URIs for accessing exposed data providers | String |
| Description | Gets a human-readable description for the link | String |
| Name | Gets the human-readable name for the link | String |

5.3.7 ExecutionLogEntry (DataType)

| Member | Notes | Type |
|--------------|--|-----------|
| Actor | Returns the actor (user/ source / system) that performed the action | String |
| Execution | Returns the execution that was logged | Execution |
| StepNumber | Returns the step number that was executed for this entry | int |
| Timestamp | Returns a time-stamp (time and date) of when this entry was created | Timestamp |
| ActualResult | Gets the actual result; for skipped steps the text will read Skipped | String |

| | | |
|---------------|---|--------------------|
| Notifications | Returns notifications that were received while executing this step; Notifications conform to formats | Notification[0..*] |
| DateTypes | Returns the data types of data received from the back-end simulator during execution; The indexes here must match up with those of DataReceived | DataType[0..*] |
| DataReceived | Returns the data received from the back-end simulator during execution that matched registrations; Indexes must match those for DataTypes; The data is in the appropriate standard format (e.g., XML) | String[0..*] |

5.3.8 Report (DataType)

| Method | Notes | Parameters |
|--|---|---------------|
| GetName() String Public | Returns the name of the report | |
| GetExperiments() Experiment[1..*] Public | Returns the experiments this report covers.; This should be gathered from the Execution instances, there should be no matching setter | |
| GetExecutions() Execution[1..*] Public | Returns the executions over which this report was created | |
| AddKPICalculator() void Public | Adds another KPI | KPICalculator |
| RemoveKPICalculator() void Public | Removes the KPI with the given name | String |
| CalculateKPIValues() void Public | Calculates KPI value by iterating over the covered executions and passing them to the KPICalculators | |
| GetKPIValues() Map<String, Double> Public | Returns a mapping from KPI names to values calculated over the executions. There should be no matching setter. | |
| GetDescription() String Public | Returns the description of the report | |

5.4 Interfaces definitions

5.4.1 Non-component interfaces

5.4.1.1 DataProvider

Instances are retrieved by the Executor from the Execution Data Management subsystem and are used for variable binding purposes. They can be used to retrieve constants, dynamic values, and data for injection into Test. There should be implementations for each DataType for retrieving constant data. This allows the execution setup to use constant values. Implementations should also be available for common storage repositories, such as Relational Data Base Management Systems (RDBMS) systems.

| Method | Notes | Parameters |
|--|--|------------|
| GetDataType() DataType Public | Returns the type of data provided. | |
| GetCardinality() Data-Cardinality Public | Gets the cardinality of the data that can be provided; For static data the minimum and maximum should be equal to the exact number of data entities available | |
| GetId() GUID Public | Gets the identifier of the provider; The identifier should be unique within the providing system | |
| GetSystemId() GUID Public | Returns the unique identifier for the providing system | |
| GetDataIterator() Iterator Public | Returns an Iterator that gives access to the data; The iterator is only required to support moving forward through the data; It may optionally provide ability to jump to an index, move backwards, or return the amount of data | |
| GetName() String Public | Returns the human-readable name of a data provider; May return null (a data provider is not required to have a name) | |
| IsExposedToApplications boolean Public | Returns whether or not the data is accessible for retrieval from Flspace applications; If true, then the ExposedDataProvider interface must be implemented | |

5.4.1.2 ExposedDataProvider

The following are methods for accessing (and potentially storing) data from Flspace applications, this interface extends DataProvider interface. These data providers are then accessible through a restful interface, which uses their DataProviderIds (pairs of GUIDs to identify the data access system and provider).

| Method | Notes | Parameters |
|-----------------------------|--|---|
| HasMore() boolean Public | Returns whether there is more data (i.e., calling getValue(Index) is valid) | int - index |
| GetValue() Object Public | Gets the value at Index; The value will be serialized to a string representation according to this instance's data type when the result is returned to the calling application; This operation must enable random access (although optimizations may be possible for forward only access) | int - index |
| SetValue() void Public | Sets the value at Index to the given parameter; The value received will have been deserialized from a string representation into an object according to this instance's data type; This is an optional operation, and should only be used if the application doesn't manage its own state for a variable | int – index object – value to be set |
| GetSize() int Public | Returns the amount of data available in the provider, if known; If unknown, returns -1 | |

5.4.1.3 DynamicallyBoundDataProvider

DynamicallyBoundDataProvider provides a binding point for variables to factory-created data providers during execution. This interface extends the DataProvider interface, although all DataProvider methods will fail until the binding takes place. The creation of the underlying data provider takes place the first time that a variable that is bound to the instance is accessed (i.e., lazy-loading on read or write). These providers will usually be managed in the internal data provider system, although the factories they access will most often be in External Data Access as they may need to access the application. These data providers will often be exposed as well. Note that if this data provider is exposed and supports setting values, then the underlying data provider must also support setting values.

| Method | Notes | Parameters |
|------------------------------|---|-----------------------|
| SetFactory() void Public | Sets the factory that will be used to create the data provider instance to which calls will be delegated | DataProviderFactoryId |
| SetFactoryArgs() void Public | Sets the arguments to be passed to the factory create method; If an Object is a VariableReference, then the iterator for that variable as given by its bound data provider will be the argument | Object[0...*] |

5.4.1.4 DataProviderFactory

Instances are used to dynamically create data providers. They should be used when dealing with dynamic variables that arise as part of application controlled lifecycles. These factories will most often create data providers that access or store application state.

| Method | Notes | Parameters |
|--|---|--|
| GetDataType() DataType Public | Returns the type of data provided | |
| GetCardinality() DataCardinality Public | Gets the cardinality of the data that can be provided; For static data the minimum and maximum should be equal to the exact number of data entities available | |
| GetId() GUID Public | Gets the identifier of the factory; The identifier should be unique within the providing system | |
| GetSystemId() GUID Public | Returns the unique identifier for the providing system | |
| CreateDataProvider() DataProvider Public | Creates a new data provider instance, based on the received parameters; The method should be capable of resolving both iterators and value objects | Object[0..*] – parameters for creating the data provider |
| GetName() String Public | Returns the human-readable name of a data provider factory; May return null (a factory is not required to have a name) | |

5.4.1.5 KPICalculator

KPICalculator is used to calculate a KPI. Instances are created in the KPI Manager component and are used by the Reporting component.

| Method | Notes | Parameters |
|------------------------------------|--|------------|
| Initialize() void Public | Initialize the calculation | |
| Update() void Public | Updates the internal state with information related to the given execution; This would involve calculating over notifications in the relevant logs | Execution |
| CompleteCalculations() void Public | Performs any final calculations necessary | |
| GetValue() double Public | Returns the calculated KPI value | |

| | |
|-------------------------------|-----------------------------|
| GetName() String Public | Returns the name of the KPI |
|-------------------------------|-----------------------------|

Additional KPIs can be composed from provided KPI functions and the base set of KPIs. Functions provided would include Sum, Average, Difference, Standard Deviation, Minimum, and Maximum. Each function would receive additional KPIs as inputs.

KPICalculator instances are created through a KPICalculatorFactory.

5.4.1.6 KPICalculatorFactory

KPICalculatorFactory is a named factory of KPICalculator instances. Base KPIs will have preinstalled KPICalculatorFactory implementations. The KPI Composer creates new instances by composing KPIs. Used by KPIManager to create new KPICalculator instances for the ReportManager.

| Method | Notes | Parameters |
|--|--|------------|
| GetName() String Public | The name of the calculation performed | |
| Create() KPICalculator Public | Creates a new KPICalculator instance | |
| ToStringRepresentation() String Public | Returns the string representation; this representation can be used as input to the KPIComposer | |

5.4.2 Component interfaces

5.4.2.1 Executor

| Method | Notes | Parameters |
|---------------------------------|--|--|
| ExecuteStep() void Public | <p>Executes the current step, and logs output; Note that for manual steps, this does not do anything</p> <p>For steps with data injectors, it will access the DataProvider instance (through the bound injection variable), retrieve each data object one at a time, injecting each through the Backend Simulator into the Test system, before proceeding to the next</p> <p>If the given execution is not properly initialized (e.g., it has unbound variables), an ExecutionNotReadyException will be thrown; If errors occur during automated steps, the execution is aborted</p> | Execution |
| LogResult() void Public | Writes a new entry to the log for the current step | <p>Execution</p> <p>String – the text to be logged</p> |

| | | |
|----------------------------------|---|-----------|
| CompleteStep() void Public | Completes the current step and progresses to the next | Execution |
| SkipStep() void Public | Skips the current step, logs a skip entry to the log | Execution |
| CancelExecution() void Public | Stops and cancels the given execution. If there is a script running for this Execution, then it will kill it. | Execution |

5.4.2.2 ScriptExecutionEngine

| Method | Notes | Parameters |
|--|---|--|
| ExecuteScript() ScriptExecutionId Public | Executes the given script; The scripting language will be dependent on the engine selected in the implementation phase; The engine should support data-binding to variables; Returns an identifier for the script execution | String – the VUser script Map<Name, DataProvider> - the variable bindings |
| WaitForCompletion() boolean Public | Waits for the script execution to complete up to a given timeout; Returns true if the execution has completed, else false | ScriptExecutionId int – timeout in seconds |
| KillScript() void Public | Will attempt the orderly stopping of the script; If not completed by the given timeout will forcibly stop the script's execution | ScriptExecutionId int – timeout in seconds |

5.4.2.3 DataProviderSystem

| Method | Notes | Parameters |
|---|--|-----------------------------|
| GetProviders() DataProvider[0..*] Public | Returns the data providers that are available in this system | |
| GetProviders() DataProvider[0..*] Public | Gets the providers matching the desired DataType and DataCardinality | DataType DataCardinality |
| GetProvider() DataProvider Public | Gets a data provider by ID | GUID |
| GetFactories() DataProviderFactory[0..*] Public | Returns the data provider factories that are available in this system | |
| GetFactories() DataProviderFactory[0..*] | Gets the data provider factories matching the desired DataType and DataCardinality | DataType DataCardinality |

| | | |
|---|------------------------------------|------|
| Public | | |
| GetFactory() DataPro- viderFactory Public | Gets a data provider factory by ID | GUID |

5.4.2.4 ExternalDataAccess

As part of system setup, a configuration stage is necessary in which DataProviderSystem instances would be configured. An implementation could for example be configured to connect to an RDBMS and retrieve data from specific tables.

| Method | Notes | Parameters |
|---|--|---|
| AddSystem() void Public | Adds a data provider system | DataProviderSystem |
| GetSystem() DataProviderSys- tem Public | Gets a data provider system by ID | GUID |
| GetSystems() [0..*] Public | Returns the data provider systems | |
| GetArchivedSys- tems() [0..*] Public | Returns the archived data provider systems | |
| ArchiveSystem() void Public | Archives the data provider system | GUID |
| UnarchiveSystem() void Public | Unarchives the data provider system | GUID |
| GetProvider() DataProvider Pub- lic | Equivalent to GetSys- tem(systemId).GetProvider(providerId) | DataProviderId – this is a pair of GUIDs, one for systemId and one for providerId |
| GetFactory() DataProviderFactory Public | Equivalent to Get- System(systemId).GetFactory(factoryId) | DataProviderFactoryId – this is a pair of GUIDs, one for systemId and one for factoryId |

5.4.2.5 InternalDataProviderSystem

This is also a DataProviderSystem but has functionality for static data configuration.

| Method | Notes | Parameters |
|--|--|--|
| CreateProvider() DataProvider Pub- lic | Creates a new data provider that provides the given data | DataType – the type of data provided by the new DataProvider Object [0..n] – the data entities to be returned by the new DataProvider |

| | | |
|---|----------------------------------|------|
| ArchiveProvider() void Public | Archives the data provider | GUID |
| UnarchiveProvider() void Public | Un-archives the data provider | GUID |
| GetAr- chivedProviders() DataProvider[0..n] Public | Gets the archived data providers | |

5.4.2.6 BackEndSimulatorService

| Method | Notes | Parameters |
|-----------------------------|--|---|
| InjectData() void Public | Injects data to the appropriate module | <p>DataType – the type of data; The modules that need this data should be uniquely determinable from this</p> <p>String[1..*] – a serialized representation of each data entity in an appropriate format for consumption by the modules (e.g., XML)</p> |

5.4.2.7 ExecutionManagerService

| Method | Notes | Parameters |
|--|--|------------|
| GetActiveExecu- tions() Execu- tion[0..*] Public | Returns the active (incomplete) executions for the given user ID | UserId |
| GetExecutions() Execution[0..*] Public | Returns the executions for a given experiment | Experiment |
| StartNewExecu- tion() Execution Public | Creates a new execution for a given experiment; The new execution has no variables bound | Experiment |
| CopyExecution() Execution Public | Creates a new execution from a given execution; The new execution will not have any records in the ExecutionLog. Variables are bound | Execution |

5.4.2.8 ExperimentCRUDService

| Method | Notes | Parameters |
|---------------------------------------|--|---|
| CreateExperiment() void Public | Creates a new experiment in persistent storage | Experiment |
| UpdateExperiment() void Public | Updates an experiment in persistent storage | Experiment |
| ReadExperiment() Experiment Public | Gets an experiment by ID | GUID – the ID corresponding to an instance (experiment ID together with version ID) |
| ArchiveExperiment() void Public | Archives the experiment | Experiment |
| UnarchiveExperiment() void Public | Un-archives the experiment | Experiment |

5.4.2.9 ExperimentSearchService

| Method | Notes | Parameters |
|---|--|------------|
| FindExperiments() Experiment[0..*] Public | <p>Finds experiments according to a query. The following information should be searchable in the query language:</p> <ul style="list-style-type: none"> • Description • Creator • Full text (including steps and variables) • Variable Descriptions • Archived Status <p>Only Experiments the user has access to will be returned</p> | String |

5.4.2.10 ResourceManager

| Method | Notes | Parameters |
|---|--|------------|
| CreateResource() void Public | Creates a new resource in persistent storage | Resource |
| GetResources() Resource[0..*] Public | Returns the resources available | |
| UpdateResource() | Updates a resource in persistent stor- | Resource |

| | | |
|--|---|----------|
| void Public | age | |
| ArchiveResource() void Public | Archives the resource | Resource |
| UnarchiveResource() void Public | Unarchives the resource | resource |
| FindResources() Resource[0..*] Public | Returns resources whose name and/or description match the query string given; Results are returned such that better matching results appear first | String |

5.4.2.11 ExecutionLogManager

| Method | Notes | Parameters |
|--|--------------------------------------|-------------------|
| LogEntry() void Public | Creates a new entry in the log. | ExecutionLogEntry |
| GetEntries() ExecutionLogEntry [0..*] Public | Returns the entries for an execution | Execution |

5.4.2.12 KPIComposer

| Method | Notes | Parameters |
|---|--|--|
| CreateCompositeKPI() KPICalculatorFactory Public | Creates a KPICalculatorFactory based on functions and base KPIs, given a string representation | String – KPI name String – KPI composition string |

5.4.2.13 KPIManager

| Method | Notes | Parameters |
|--|--|--|
| GetBaseKPINames() String[0..*] Public | Returns the base KPI names available in the system | |
| GetKPINames() String[0..*] Public | Returns the names of all KPIs | |
| RegisterNewKPI() void Public | Uses the KPIComposer to create a new KPICalculatorFactory and register it with the given (unique) KPI name | String – KPI name String – KPI composition string |

| | | |
|--|---|---------------|
| ArchiveKPI() void Public | Archives a composite KPI | String – name |
| UnarchiveKPI() void Public | Un-archives a composite KPI | String – name |
| NewKPICalculator() KPICalculator Public | Creates a new KPICalculator instance for the given name | String |

5.4.2.14 ReportManager

| Method | Notes | Parameters |
|--------------------------------------|---|----------------|
| CreateReport() void Public | Stores a new report | Report |
| GetReports() Report[0..*] Public | Retrieves reports covering the given experiment | Experiment |
| GetReports() Report[0..*] Public | Retrieves reports covering the given execution | Execution |
| FindReports() Report[0..*] public | Searches by name, description and note to find reports. Results should be returned with better match results first. | String – query |

5.4.2.15 UserManager

The User Manager will expose the standard user and authorization management methods for controlling access to Experiments, DataProviderSystems, Executions, and Reports. By default, access to the individual experiment is used to control who can access the resulting executions and related reports. Optionally, these may be overridden to provide more fine-grained control.

6 Summary

This document describes the architecture, development and scenario execution plan of Flspace EE. This report is the direct continuation of the work done by the WP4 of Flnest project. The EE architecture section provides the detailed description of the EE main components, data types and interfaces. The EE development section describes the development processes that is planned to be performed during the Flspace project which is at this point is fully aligned with the work plan described in the DoW. Finally, the scenario execution section provides the template for execution plan and demonstrates the entire process on the example of the Advice request scenario taken from the Greenhouse Management and Control Trial 422.

