FInest – **F**uture **In**ternet **e**nabled optimi**s**ation of **t**ransport and logistics networks

D4.4

Experimentation environment specification and phase 2 implementation plan

Project Acronym	FInest	
Project Title	Future Internet enabled optimisation of transport and logistics networks	
Project Number	285598	
Workpackage	#WP4 Experimentation Environment	
Lead Beneficiary	IBM	
Editor	Fabiana Fournier	IBM
Contributors	Moti Nisenson	IBM
	Guy Sharon	IBM
Reviewers	Cyril Alias	UDE
	Agathe Rialland	MRTK
Dissemination Level	Public	
Contractual Delivery Date	30/3/2013	
Actual Delivery Date	30/3/2013	
Version	V1.0	

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 285598

Abstract

The ultimate goal of the experimentation environment of FInest is to allow the running of the specified use cases scenarios in phase 2 of the project and large trials in phase 3. Towards this end, a specification of the required environment, an examination of possible experimentation sites, and a detailed implementation plan have been elaborated. This report summarizes the work done so far in FInest in work package 4 and paves the way to the work to be done in phase 2 to achieve the desired goal.

Document History

Version	Date	Comments
V0.1	20/11/12	First draft
V0.2	20/1/13	Updates after IBM internal discussions
V0.3	15/2/13	Updates after the technical meeting in Essen
V0.4	1/3/13	Updates after project plenary meeting in Hamburg
V0.5	15/3/13	Updates after review
V1.0	30/3/13	Submitted version

Table of Contents

List of Tables

List of Figures

Acronyms

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

1. Introduction

FInest (Future Internet Enabled Optimisation of transport and Logistics Business Networks) Work Package 4 (WP4) deals with the identification and design of an Experimentation Environment (EE) for testing, demonstrating, and evaluating the envisioned technologies devised in FInest for the transport and logistics domain. The aim is to provide a suitable environment for conducting the experiments for the use case scenarios specified in phase 2 and for large trials in phase 3. FInest project will continue into phase 2 as the *cSpace* project (Future Internet Business Collaboration Networks in Agri-Food, Transport and Logistics) and FInest EE will become cSpace EE. More specifically, WP300 "cSpace Hosting and Experimentation" is the direct continuation of FInest WP4 Experimentation Environment.

Deliverable 4.2 "Requirements and design of transport and logistics experimentation environment" [1] (submitted at M12) provides a final list of requirements for FInest experimentation environment along with a first assessment of reusable technologies and experimentation sites. Deliverable 4.3 "Interim specification for transport and logistics experimentation environment"[1] (submitted at M18) provides a sound architecture of FInest experimentation environment. This architecture has been validated by both technical and domain partners during the phase 2 writing process of the cSpace proposal.

This report summarizes WP4 tasks so far and provides, in addition, a plan for the EE implementation during phase 2 of the project. This document is composed of three interconnected parts: Section 2 addressed the refinement of the architecture presented in D4.3; section 3 provides a final assessment of the reusable technologies and experimentation sites; and section 4 provides a detailed phase 2 implementation plan. We conclude the report with a summary.

2. FInest experimentation environment technical specification refinement

A very detailed specification of FInest EE including interfaces and data types has been presented in D4.3[1]. This specification has been re-checked during the cSpace writing proposal of WP300 description and has been found most relevant and appropriate also for the requirements of FInest successor project. In fact, the implementation plan introduced in Section 4 is fully aligned with the deployment of such an environment.

Still, cSpace platform should be more generic than FInest. First it should support the actual deployment of eight use cases from both the domains of agri-food and transport and logistics, and second, it should pave the way to large trials in phase 3. One of the targets of the EE is to support inclusion of physical sites and real data, as well as simulated data for cases in which real-time data can not be obtained. For the latter case, we foresee that as part of the configuration of an application to be developed on-top of cSpace, there will be some parameters that will have to be associated to a test before actual test execution takes place. These bindings are necessary in order to uniquely "package" the run of a specific scenario to a specific test. To this end we refined the specification proposed in D4.3 to be more generic in order to meet

[1] Available at http://www.finest-ppp.eu/

cSpace requirements. More specifically, the previous binding to a TCP ID as the initialization of a new execution will be replaced by a more sophisticated mechanism that extends the FInest EE architecture to include also the requirements from an application to be tested in the EE. We iterate that the specification presented in D4.3 fully meets FInest requirements, the updates presented in this document are targeted to meet cSpace requirements. For the sake of completeness we include in this report the full specification as submitted in D4.3 along with additions that reflect the modifications aforementioned. These will be shown in blue to differentiate new text from the one existing in D4.3 (or as ~~redundant~~). As the refined specification relates to cSpace (a more general platform than FInest), "cSpace" replaces "FInest" in line the text.

Table 1 shows the relationships between D4.3 and D4.4 sections related to the EE specification to assist the reader familiar with the wording and specification in D4.3. For overview on the proposed EE refer to D4.3.

Table 1: Relation between D4.3 and D4.4 EPM specification sections

Section in D4.3	Section in D4.4
2.3 Terms	Section 2.1 - Copied
3. Technical specification of FInest EE	Section 2.2 – copied with slight modifications (shown in blue)
3.1 FInest EE components	Section 2.2.1 – copied with slight modifications (shown in blue)
3.2 Data types definitions	Section 2.2.2 – copied with slight modifications (shown in blue)
3.3 Interfaces definitions	Section 2.2.3 – copied with slight modifications (shown in blue)
3.4 Association of a TCP ID to a test	Section 2.3 – modified
N/A	Section 2.4 - Summary of important architecture refinements

2.1. 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 single action/task defined in a test scenario.

Test scenario – 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 – Replacement of variables values with the test data. This is done by the experimenter during test execution.

Experiment/test – The ordered set of steps to be carried out by an experimenter during execution. Each experiment is identified by a unique ID and version. ~~It is also associated to a single Transport Chain Plan (TCP).~~ An experiment may have variables to enable multiple executions of the same experiment with different data.

Execution – The actual running of an Experiment. All variables should be bound to Data Providers before execution of steps can begin.

Vusers – Virtual users that play human users in a specific experiment.

Vusers scripts – 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 – A 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 cSpace test.

Execution log - 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 – the anticipated outcome of a step in a test.

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

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

Report – 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. 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 (real-time) historical data to simulate the processes.

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

2.2. Architecture refinement

In general, the envisioned experimentation environment will operate by activating the cSpace platform and will invoke it at each test execution, utilizing cSpace technologies and databases. It consists of three interconnected major components (see Figure 1)

- *cSpace test* – a replica of cSpace platform for testing purposes in order to avoid "playing" in production environment. It is anticipated that in order to enable test executions with real-data as well as with simulated data, the UI will be extended to support both modes.

- *cSpace experimentation environment* – includes all components required to run and analyze tests executions, as well as databases for the storage of executions, execution logs, reports, KPI(s), test data, resources, and roles and access rights.
- *cSpace experimentation environment front-end* – The UI for the users to be able 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*: the knowledge of the system, including the entities, statuses, and states; and the necessary logic for creating and conducting experiments.
- *View*: the presentation and representations of the model. In this case the displayed information includes experiment steps and reports.
- *Controller*: the link between the user (the view) and the system (the model). The controller receives the user's input and updates the model state accordingly.

The EE (model) 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.

Experimentation Front End: the UI for the users to be able to use the experimentation environment to create, update, execute and report on tests. This includes the following high-level views (see Figure 1):

- *Access Handling*: control 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 cSpace Test system through a *Backend Simulator* component. This includes injecting data into cSpace test and recording events and other data processed by cSpace 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 and be incorporated into the EE for specific purposes. Specifically, we believe that the *reporting* component and the *script engine* component (for executing Vusers scripts), can be off-the-shelf and do not require self-development by the cSpace 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 cSpace.

Figure 1 presents cSpace EE architecture followed by a description of the different modules and definition of the data types and interfaces. Note that the technical architecture depicted in Figure 1**Error! Reference source not found.** is defined at the model-level, using TAM (the Technical Architecture Modeling language)[2], a UML derivate, following the convention used in the other technical work packages.

[2] http://www.fmc-modeling.org/fmc-and-tam

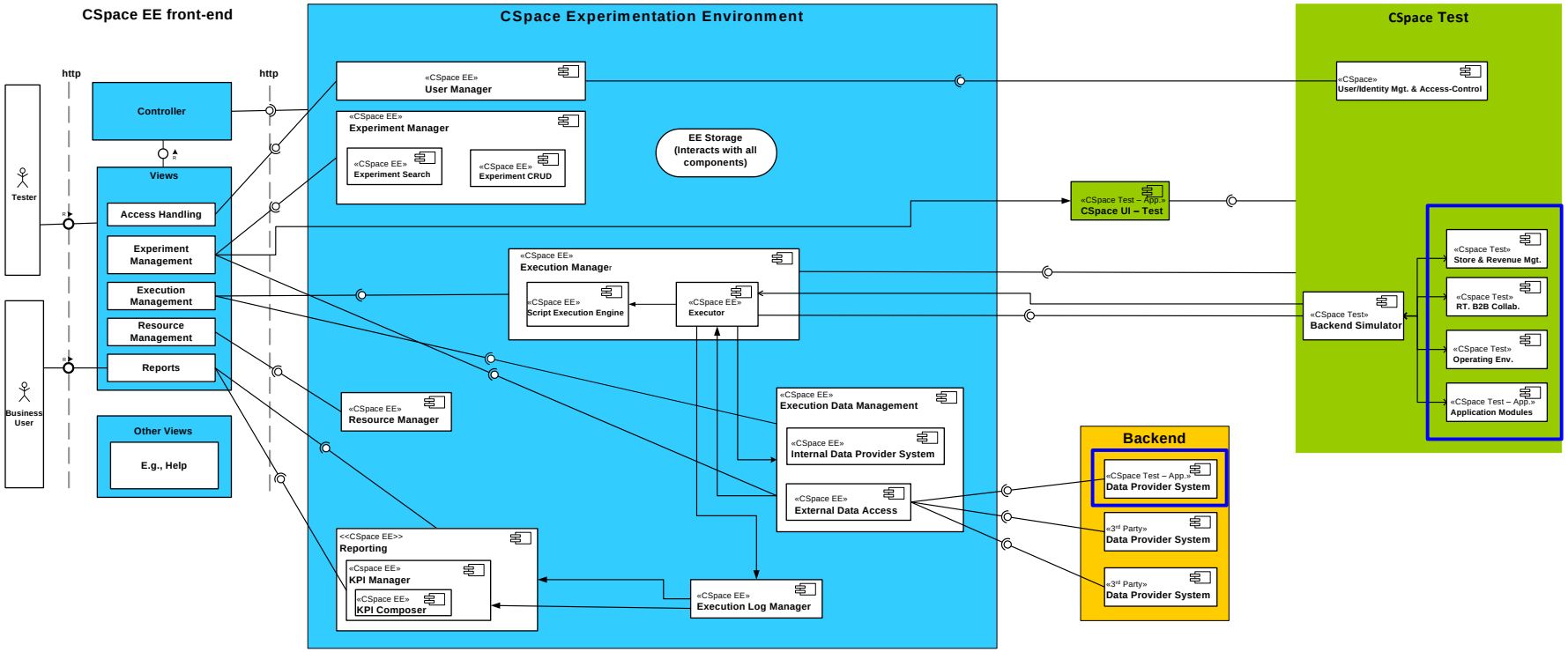


Figure 1: cSpace experimentation environment architecture

2.2.1. Experimentation environment components

Only two modifications (shown with the blue square around the relevant components in Figure 1) have been introduced to the architecture proposed in D4.3.

1. Data Provider System (app) – This is a new component which addresses the binding of the application to the execution data. See below for details.
2. Relevant cSpace components replace the four FInest modules (Business Collaboration Module, Event Processing Module, Transport Planning Module, and E-Contracting Module), to reflect the extended platform proposed in (see cSpace proposal for further details on cSpace components).

Follows a description of the cSpace EE components:

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: Handles experiment lifecycle and experiment querying. This includes creation, versioning, archiving, and search capabilities.

1) *Experiment CRUD* (Create, Read, Update, and Delete): provides services for experiment lifecycle. Archiving is used instead of deletion so that traceability is never lost.
2) *Experiment Search*: provides services for finding experiments according to various search criteria.

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.

1) *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.
2) *Script Execution Engine*: executes VUser scripts to automate user actions.

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

Reporting: Generates reports based on execution logs and KPIs. Note that in Phase 2 of the project, reporting (together with KPIs) are expected to become off-the-shelf modules and will not require self-development by the cSpace team.

1) *KPI Manager*: Manages the calculation and composition of KPIs
    a. *KPI Composer*: Used to create and manage composite KPIs

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.

1) *Internal Data Provider System*: Used for storing and retrieving manually configured data providers.
2) *External Data Access*: Used to retrieve data from 3rd party external systems. For example, this could be used to "replay" events from a real-world shipment. The access to these systems is configured by the tester. Configuration could be UI or file driven.

Backend Simulator: used to simulate input data from backend systems to cSpace; provides APIs to inject data to the cSpace Test system's modules. Also reports back on events and other processed data.

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

2.2.2. Data types definitions

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

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()	Gets the experiment id; this is common to all the different versions of an	

GUID Public	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.	

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..*	
GetDefaultDataProvider 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.	

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 FINest (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 CSpace 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 CSpace 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..*]
~~NewTCPRequired~~	* Not required *	
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

4. Execution (DataType)

Method	Notes	Parameters
GetExperiment() Experiment Public	Returns the experiment instance this execution is associated with.	

~~AssociateTCP()~~	* Not required *	
~~GetAssociatedTCPIds() void Public~~	* Not required *	
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. 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

6. Link (DataType)

Member	Notes	Type
URI	Gets the URI. The URI embeds	String

	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). $$ is used to encode a single $. If additional data is not null, then the uri also embeds an additionalData parameter containing a URI (which included the execution id) for retrieving the additional data	
AdditionalData	Application specific additional data to be retrieved. Embeds variable access by using $varName (which is substituted upon retrieval with the variable's current value). $$ 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

7. ExecutionLogEntry (DataType)

Member	Notes	Type
Actor	Returns the actor (user/ source / system) which performed the action	String
Execution	Returns the execution which was logged	Execution
StepNumber	Returns the step number which 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 which 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 which matched registrations. Indexes must match those for DataTypes. The data is in the appropriate standard format (e.g. XML).	String[0..*]

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	

2.2.3. Interfaces definitions

2.2.3.1. Non-component interfaces

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 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() 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 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 which 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 CSpace applications. If true, then the ExposedDataProvider interface must be implemented.	

2. ExposedDataProvider

Methods for accessing (and potentially storing) data from CSpace 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.	

3. DynamicallyBoundDataProvider

Provides a binding point for variables to factory created data providers during execution. This interface extends 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 which 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 which 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…*]

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 which 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. 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	Initialize the calculation	

Public		
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.

6. KPICalculatorFactory

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	

2.2.3.2. Component interfaces

1. Executor

Method	Notes	Parameters
ExecuteStep() void	Executes the current step, and logs	Execution

Public	output. Note for manual steps this doesn't do anything. 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. 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.	
LogResult() void Public	Writes a new entry to the log for the current step.	Execution String – the text to be logged
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

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	ScriptExecutionId

	given timeout will forcibly stop the script's execution	int – timeout in seconds

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..*] Public	Gets the data provider factories matching the desired DataType and DataCardinality.	DataType DataCardinality
GetFactory() DataProviderFactory Public	Gets a data provider factory by id	GUID

4. **ExternalDataAccess**

As part of system setup, a configuration stage is necessary where-in 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() DataProviderSystem Public	Gets a data provider system by id	GUID
GetSystems() [0..*] Public	Returns the data provider systems	

GetArchivedSystems() [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 Public	Equivalent to GetSystem(sytemId).GetProvider(providerId)	DataProviderId – this is a pair of GUIDs, one for systemId and one for providerId
GetFactory() DataProviderFactory Public	Equivalent to GetSystem(sytemId).GetFactory(factoryId)	DataProviderFactoryId – this is a pair of GUIDs, one for systemId and one for factoryId

5. **InternalDataProviderSystem**

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

Method	**Notes**	**Parameters**
CreateProvider() DataProvider Public	Creates a new data provider which 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	Unarchives the data provider	GUID
GetArchivedProviders() DataProvider[0..n] Public	Gets the archived data providers.	

6. BackEndSimulatorService

Method	**Notes**	**Parameters**
InjectData() void	Injects data to the appropriate module.	DataType – the type of data.

Public		The modules which need this data should be uniquely determinable from this String[1..*] – a serialized representation of each data entity in an appropriate format for consumption by the modules (e.g XML)

7. ExecutionManagerService

Method	Notes	Parameters
GetActiveExecutions() Execution[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
StartNewExecution() 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
~~AssociateTCP()~~ ~~void Public~~	*not required*	

Since the system logs all notifications dealing with TransportChainPlans (TCPs) and associates them to a specific execution, the creation of TCPs is a special action in the experimentation environment. When creating a TCP, the user must be directed through a UI which enables the experimentation environment to capture the TCP id and associate it with the execution.

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	Unarchives the experiment	Experiment

9. ExperimentSearchService

Method	Notes	Parameters
FindExperiments() Experiment[0..*] Public	Finds experiments according to a query. The following information should be searchable in the query language: • Description • Creator • Full text (including steps and variables) • Variable Descriptions • Archived Status Only Experiments the user has access to will be returned.	String

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() void Public	Updates a resource in persistent storage	Resource

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

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

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

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	Unarchives a composite KPI	String – name
NewKPICalculator() KPICalculator Public	Creates a new KPICalculator instance for the given name	String

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

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.

2.3. Execution association

As previously noted, we extend the EE specification to support generic scenarios beyond transport and logistic. 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.

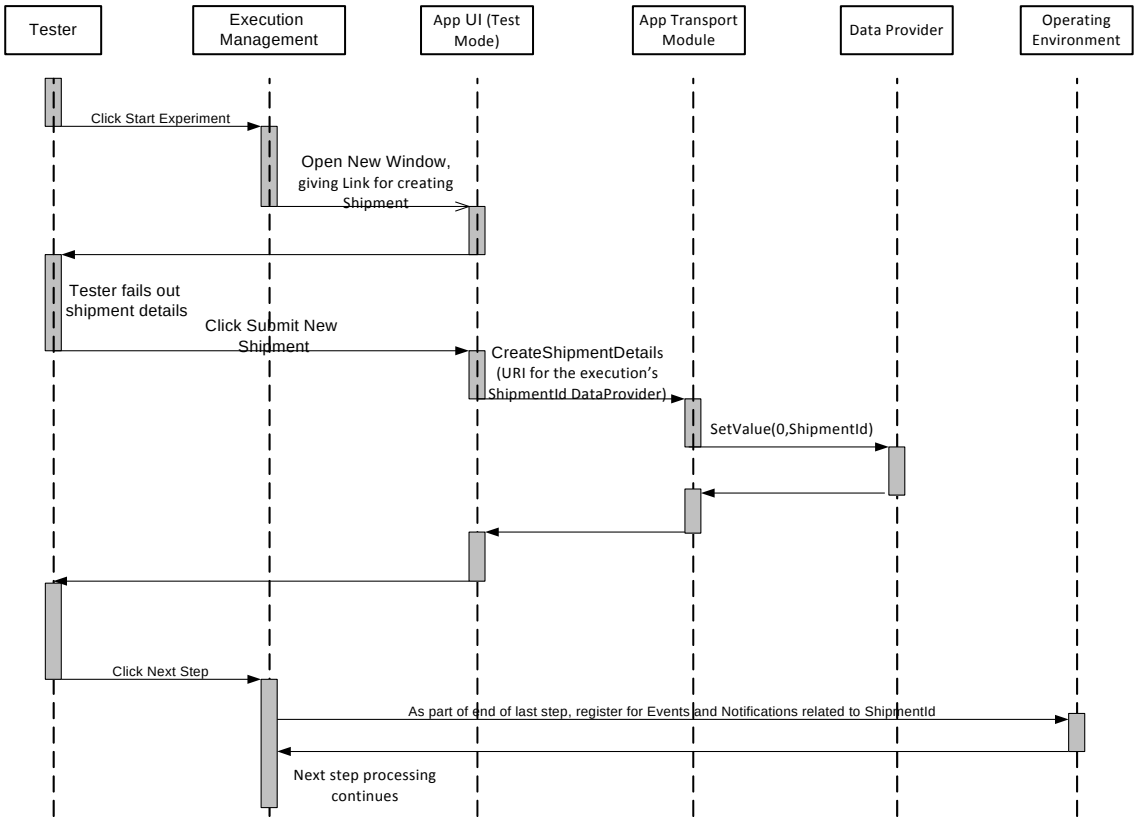


Figure 2: Example of setting new value for a new Shipment

2.4. Summary of important architecture refinements

In general, cSpace EE needs to extend FInest EE to cope with a higher level of abstraction – the application level. We summarize below the additions and modifications we proposed so far to achieve this target.

Extensions for supporting applications

* Dynamic data binding:
    - Data Provider Factories: 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: lazy creation of a data provider through a factory. These data providers should be defined using variable names which 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: applications can access and potentially update data for data providers which 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. These links 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 labelled with "CSpace Test – App" in the architecture diagram to differentiate them from the CSpace environment, which are labelled "CSpace Test".

Application Requirements

CSpace applications are required to supply certain capabilities so that they can be supported in the CSpace Experimentation Environment. These capabilities primarily center around the lifecycle of application state. For example, consider a transport scenario where there is a shipment which consists of several legs. The shipment itself would need to be created. Each leg would need to be created. There may be events which need to be injected (i.e., simulated) for one of the legs. It is likely that the shipment id and leg id would 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 shipment and leg creation. In order 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 receive or set data as needed. The links to the application UI are provided through data providers. Additionally, for any application data which is managed by the application (for example, data which changes dynamically within CSpace as a result of events) and which is necessary during experiment execution, the application will need to provide a data access system for connecting to the application.

3. Reusable technologies and experimentation sites final assessment

The main objective of the INFINITY (http://www.fi-infinity.eu) project is to capture and communicate information about available experimental infrastructures in Europe and beyond, in order to facilitate large scale experimentation and testing for Future Internet projects and applications and service developments. To this end, the INFINITY project has launched the portal for Future Internet infrastructures named XiPi (www.xipi.eu) that currently stores about 130 infrastructures details. The follow up project, XiFi will further develop these infrastructures to be exploited by the FI-PPP projects. The current infrastructures are basically clouds that can be used by the FI-PPP use cases projects. The XiFi project will leverage these infrastructures with the deployment of some of the generic enablers (GEs) provided by the FI-WARE (http://www.fi-ware.eu/) project.

Our goal is to run the phase 2 scenarios in environments provided by XiFi. Our emphasis is on physical sites equipped with IoT (Internet of Things) sensors and real-time 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 route or ship itinerary). These requests have been presented to the INFINITY project and we do hope that we will be able to use some of the environments provided by XiFi during the phase 2 project. However, after reviewing all the infrastructures provided so far, we cannot rely on the proposed infrastructures as reusable technologies for cSpace at the moment. Therefore, in the intermediate time, as part of the WP300 work in cSpace, we will build upon an internal cloud infrastructure in which the cSpace components as well as the use cases will be deployed, and the EE will operate. As part of WP400 "Use Case Trails" in cSpace, eight use cases will be

specified in such a way that they can either be physically tested or simulated 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, 8 use case trials have been specified for cSpace, organized along 3 themes:

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

1. Crop Protection Information Sharing – use of field sensor and satellite data to intelligently man-age the application of pesticides for maximum crop protection
2. Greenhouse Management & Control – use of 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:

3. 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
4. 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.
5. 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. Focus is with 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 producers to better control the flow of their goods to the consumer, covering three use case trials:

6. 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).
7. 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.
8. Tailored Information for Consumers – the provisioning of accurate information to individual consumer’s needs and feedback of this information to the producers

First step towards a full definition of the scenario to be tested in the EE will be carried out during a try-out meeting to be held during April 4-5. The aim is to build a procedure or method to define the specification of an end-to-end scenario, so the other use cases can follow. The experimental sites presented in D4.2 will serve as the basis for the use cases originating from FInest and as examples for the new use cases in cSpace.

4. Experimentation environment phase 2 implementation plan

As previously noted, a dedicated work package in cSpace (WP300) will be accountable for the "cSpace Hosting and Experimentation" of the project. WP300 in cSpace is a straightforward continuation of the work accomplished in FInest WP4 "Experimentation Environment". In addition, cSpace WP300 also includes the deployment of the platform components and FI-WARE[3] Generic Enablers (GEs) in a cloud infrastructure. cSpace WP300 uptakes FInest WP4 results and takes FInest EE specification as the starting point to build upon. Figure 3 depicts the different Tasks to be accomplished by WP300 in cSpace.

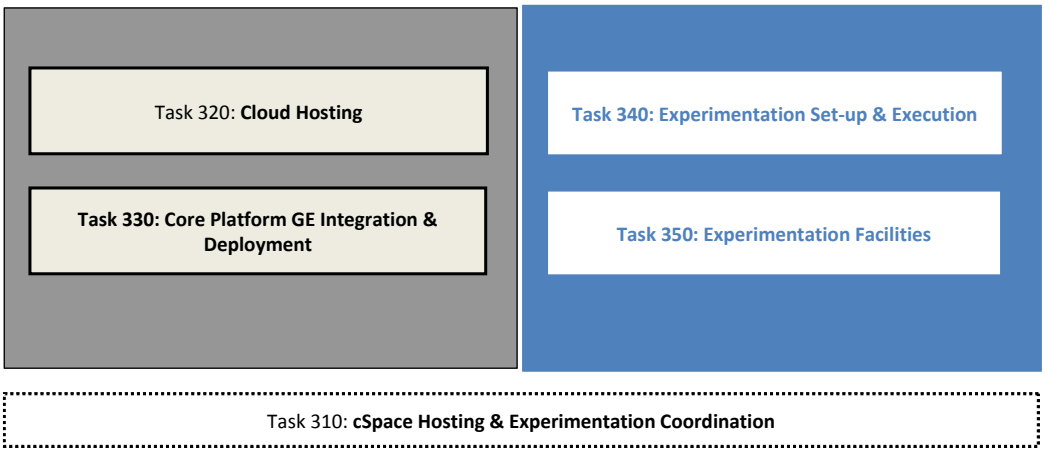


Figure 3: WP 300 cSpace Hosting & Experimentation (source: cSpace proposal)

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

Experimentation set-up and execution - This task objective is to 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 cSpace new collaborations performance as reflected in the use cases 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 cSpace project. In essence, this task supports the experimentation process as shown in Figure 4 (cf. Figure 1 in D4.3).

This task is further divided into two subtasks:

1) **Experiment design and configuration** (M1- M15) – This subtask will focus on making the use case trials defined throughout phase 2 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).

[3] http://www.fi-ware.eu/

2) **Experiment execution and analysis** (M9-M24) - This subtask will deal 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).

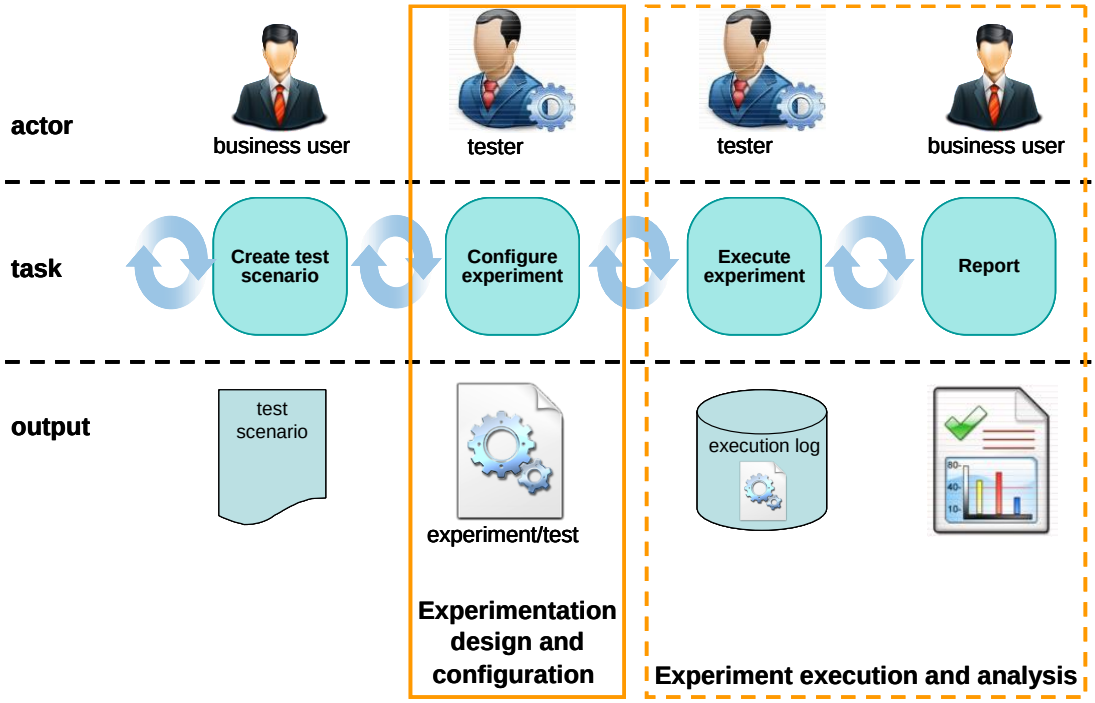


Figure 4: Experimentation process (cf. Figure 1 in D4.3)

Experimentation facilities task

This task is concerned with the scaffolding and interfaces that are required in order 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

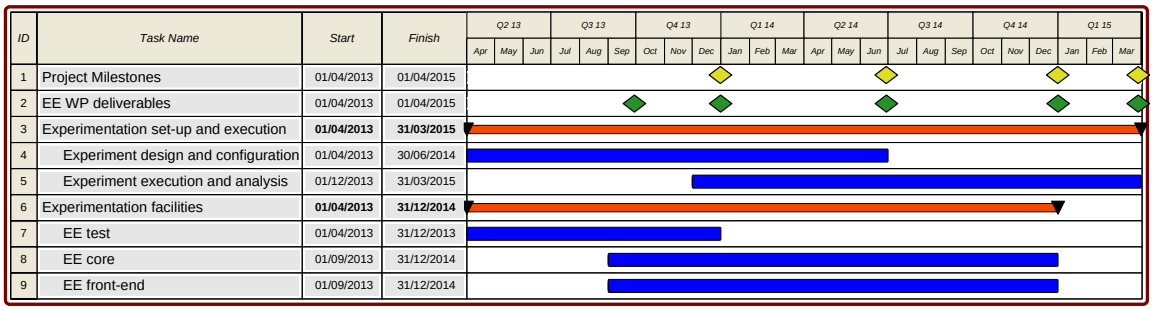
This task is further divided into three subtasks:

1) **EE test** (M1-M9) – This subtask is concerned with putting in place a testbed for cSpace in which the use case trials will be carried out, including simulation capabilities. Namely, the Test components in Figure 1.

2) **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.

3) **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 (see MS3, MS5, and MS7 descriptions below) and with the work package deliverables (see deliverables descriptions below). The above tasks are shown in the implementation plan Gantt chart in Figure 5 along with the project milestones and work package deliverables.



ID	Task Name	Start	Finish
1	Project Milestones	01/04/2013	01/04/2015
2	EE WP deliverables	01/04/2013	01/04/2015
3	Experimentation set-up and execution	**01/04/2013**	**31/03/2015**
4	Experiment design and configuration	01/04/2013	30/06/2014
5	Experiment execution and analysis	01/12/2013	31/03/2015
6	Experimentation facilities	**01/04/2013**	**31/12/2014**
7	EE test	01/04/2013	31/12/2013
8	EE core	01/09/2013	31/12/2014
9	EE front-end	01/09/2013	31/12/2014

Figure 5: EE phase 2 implementation plan Gantt chart

List of relevant project milestones

For a full description of the milestones refer to D3.4 "Technical specification of domain specific FI platform for transport and logistics and phase 2 implementation plan" (to be submitted at M24) and cSpace proposal.

MS3: Release V1 (M9) Initial release of experimentation environment; experimental support and guidelines in place

MS5: Release V2 (M15) Advanced release of experimentation environment

MS7: Release V3 (M21) Final release of experimentation environment

MS8: Trial-Round 3 (M24)

Planned deliverables

EE architecture and development plan + scenarios execution plan (M6, Type: R)

Regarding the EE architecture and development plan, this deliverable will provided a detailed plan for the development of the required scaffolding (modules and interfaces) required for the experimentation environment. This work also includes the examination of build-versus-buy components, such as reporting and script engine. This deliverable relates to the EE test, EE core, and EE front-end subtasks.

Regarding the scenarios execution plan, the deliverable will review the detailed experimentation and initial work plans of WP400 (use cases), gather test cases from WP400 use cases and their test scenarios designed together with business users, analyze required functionalities and desired outcomes from WP400 use cases, and report proposals to refine use cases or requirements. This relates to the Experiment design and configuration and Experiment execution and analysis subtasks.

Initial release of the EE + first scenario executions (M9, Type: R + P)

First release and integration of the development efforts into cSpace EE. This relates to the EE test, EE core, and EE front-end subtasks. The first release will include alpha test runs to validate test scenarios execution and report results and further requirements if needed to respective WP400 use cases. This deliverable relates to the Experiment design and configuration and Experiment execution and analysis subtasks.

Advanced release of the EE + more scenario executions (M15, Type: R + P)

Second release and integration of the development efforts into cSpace EE, including refinement of first release based on the test executions (EE test, EE core, and EE front-end subtasks). More scenarios executions will include connecting first business partners to the platform and running beta and performance tests to examine further hardware requirements (Experiment design and configuration and Experiment execution and analysis subtasks).

Experimentation environment development final release (M21, Type: P)

Final release and integration of the development efforts into cSpace EE. This will include refinement of the previous release based on the scenarios execution. This deliverable relates to EE test, EE core, and EE front-end subtasks**.**

Final scenario executions (M24, Type: R)

Report on the execution of the use cases as defined in WP400 in collaboration with business partners. Therefore, it relates to Experiment execution and analysis subtask.

5. Summary

Deliverable 4.4 "Experimentation environment specification and phase 2 implementation plan" in fact refines and summarizes the work accomplished in WP4 of FInest project and provides a plan to implement this work in the scope of cSpace, FInest phase 2 FI-PPP project. The results of the work done in FInest WP4 have been validated by both domain and technical partners not only in FInest consortium but also in cSpace consortium as part of the preparations of the WP300 "cSpace Hosting and Experimentation" during the proposal writing. We are confident that the work accomplished in the scope of WP4 in FInest provides a significant spring board to the tasks to be completed in cSpace WP300, FInest WP4 straightforward continuation.