



FInest – **F**uture **I**nternet enabled optimisation of **t**ransport and logistics networks



D4.3

Interim specification for transport and logistics experimentation environment

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	30/9/2012	
Actual Delivery Date	30/9/2012	
Version	V1.0	

Abstract

Work package 4 in the FInest project deals with the identification and design of an experimentation environment for testing, demonstrating, and evaluating the technologies developed during the project based on FInest use cases.

This document is a straightforward continuation of D4.2 "Requirements and design of transport and logistics experimentation environment" and describes the technical specification for the conceptual design presented in D4.2. The main goal of the proposed architecture is to satisfy the conceptual design and requirements of the experimentation environment envisioned in FInest, allowing running the specified use cases scenarios in phase 2 of the project and large trials in phase 3.

Document History

Version	Date	Comments
V0.1	15/8/12	First draft
V0.2	23/8/12	Updates after discussions with WP2
V0.3	1/9/12	Updates after IBM internal review
V0.4	6/9/12	Updates after project plenary meeting
V0.5	21/9/12	Updates after external review
V1.0	30/9/12	Submitted version

Table of Contents

Abstract	2
Document History	3
Table of Contents	4
List of Tables.....	5
List of Figures	5
Acronyms.....	6
1. Introduction	7
2. FInest experimentation environment overview	7
2.1. What is the main goal of the experimentation environment?.....	7
2.2. Users of FInest experimentation environment	8
2.3. Terms	8
2.4. Experiment execution in FInest EE	9
2.4.1. Create test scenario.....	10
2.4.2. Configure experiment.....	11
2.4.3. Execute experiment.....	12
2.4.4. Report.....	13
2.4.5. Re-using an existing test.....	13
2.4.6. Ending an execution	14
2.5. Tightly connection with work package 2 "Use case specification"	14
3. Technical specification of FInest EE	15
3.1. FInest experimentation environment components.....	18
3.2. Data types definitions	19
3.3. Interfaces definitions	24

3.3.1. Non-component interfaces	24
3.3.2. Component interfaces	26
3.4. Association of a TCP ID to a test	32
4. Summary and next steps	33

List of Tables

Table 1: Example snippet of the "real-time event handling" use case test scenario.....	10
Table 2: Example for an experiment for the "real-time event handling" use case test scenario11	
Table 3: Example of an experiment test after execution	13

List of Figures

Figure 1: Experimentation process in FInest EE	10
Figure 2: Example of system notifications	12
Figure 3: Intersections points between WP2 and WP4	15
Figure 4: FInest experimentation environment architecture	17
Figure 5: Association of a TCP ID to a test.....	33

Acronyms

Acronym	Explanation
BCM	Business Collaboration Module
CRUD	Create, Read, Update, Delete
EE	Experimentation Environment
ECM	E-Contracting Module
EPM	Event processing Module
Finest	Future Internet Enabled Optimisation of transport and Logistics Business Networks
KPI	Key Performance Indicator
RDBMS	Relational Database Management System
T&L	Transport and Logistics
TCP	Transport Chain Plan
TPM	Transport Planning Module
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 FInest in phase 2 and for large trials in phase 3. The proposed architecture supports inclusion of physical sites and real data, as well as simulated data for cases in which real-time data can not be obtained.

Deliverable 4.2 "Requirements and design of transport and logistics experimentation environment"¹ provides a found list of requirements for FInest experimentation environment followed by a high level architecture of such an environment that satisfies the list of requirements. This report follows the conceptual design and provides a first technical specification of the different modules and interfaces of the architecture.

2. FInest experimentation environment overview

2.1. What is the main goal of the experimentation environment?

As stated in D4.2 "Requirements and design of transport and logistics experimentation environment", the main goal of FInest EE is to provide a means via which business people and different stakeholders in the Transport and Logistics (T&L) supply chain can explore, interact with, understand how the FInest technology will work in practice, and ultimately satisfy themselves that the system will meet their business requirements (*Does the system deliver the expected business functions?*)

FInest EE will provide a "safe environment" in which people can "try and play" with, using FInest to test new collaborative scenarios before these processes are implemented in practice, rather than "assuming" how the new business collaborations will look like. To some degree, FInest EE can also serve as a public relations exercise as end users are involved so they can gain some confidence in understanding how to use the proposed platform, and very importantly, can start to see the benefits of using it.

Specifically, the proposed design of the EE provides an environment that can make use of real data and physical sites as well as simulation environment in which data is injected into the system. The ultimate goal is that the specified EE will enable the execution of the use cases specified in FInest in phase 2, and allow for large trials in phase 3.

¹ Available at <http://www.fi-ppp.eu/>

2.2. Users of Finest experimentation environment

We distinguish two group categories of users:

- *Experimenters/testers*: the actual user of the system, the person who designs and runs experiments/tests. Experimenters possess knowledge about the T&L domain (domain experts) and about simulation technologies. They are also experts in the functioning and operation of the simulation environment.
- *Participants*: people/organizations (business users) that form part of the test or experiment. These can be providers or consumers of services employed in the tests or anyone in the T&L supply chain. Participants or business users specify the scenarios to be tested as well as analyze the execution results.

2.3. 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 (see Table 1).

Test scenario – The ordered set of steps that compose a single test (see Table 1)

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 (see Section 2.4.3).

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) ID (see Section 3.4). 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 can begin (see Section 2.4.3).

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 Finest 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 (see Section 2.4.4).

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 (see Section 2.5).

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 (see Figure 4). 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 FInest frontend during an execution of a test. Notifications are recorded in the execution log of the test.

2.4. Experiment execution in FInest EE

As aforementioned, FInest EE enables the entire process, from test/experiment definition, through execution, to analysis of the test execution. Figure 1 illustrates this process flow.

Figure 1 describes the process in three layers performed in parallel.

- *Actor*: who performs this task/who interacts with FInest EE?
- *Task*: what is performed?
- *Output*: what is the result of this task?

Although the tasks in the process are sequentially in nature, each phase can iterate with itself and with the previous phase. For example, the configuration of a test might require several iterations between a tester and the relevant business user, as well as many iterations of the tester in order to achieve the set of the atomic steps to be executed. This is illustrated by the feedback/circular arrows between the tasks.

The outputs resulting from the test configuration, executions, and analysis, are stored in the EE along with references to each other to enable tracing.

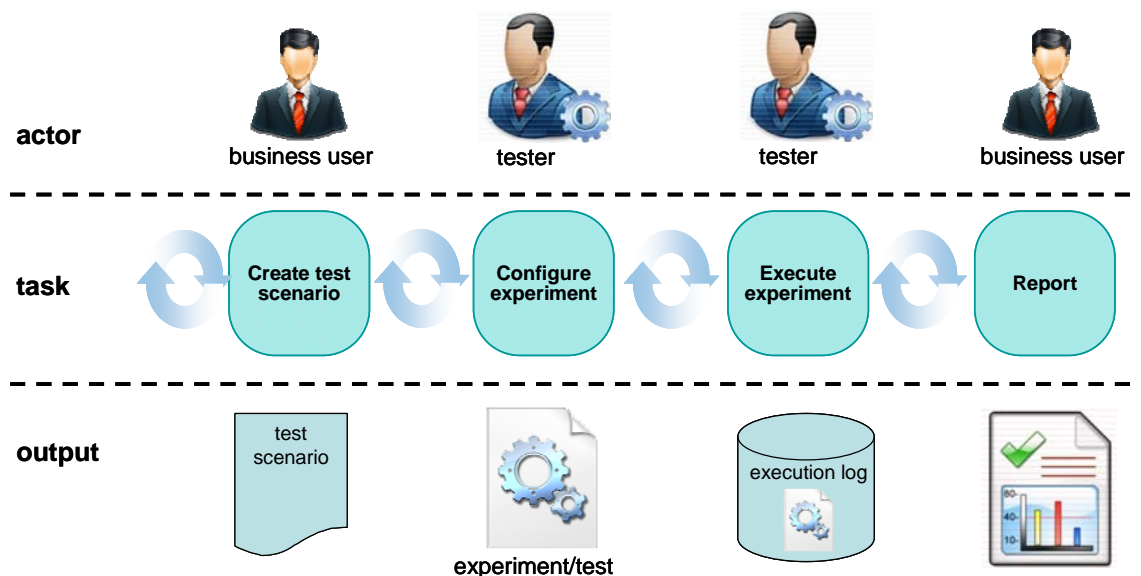


Figure 1: Experimentation process in Finest EE

2.4.1. Create test scenario

In general, time and effort should be invested in selecting appropriate scenarios for inclusion as it will be impossible to test all possible scenarios. The creation of the five use case scenarios selected for Finest is part of the work of WP2 and one of the outcomes of D2.4 is to provide the test scenarios specification for these selected use cases (refer to the tight coupling between WP2 and WP4 as described in Section **Error! Reference source not found.2.5**).

The business user prepares a test scenario that is a document consisting of the sequence of steps to be manually performed by the tester. For each step a description of the test, the actor(s) involved (role that performs the test, e.g., shipper, forwarder...), relevant data (including variables), and expected result are given. See Table 1 for an example extracted from the "Real-time event handling" use case (refer to deliverable D2.4)

Table 1: Example snippet of the "real-time event handling" use case test scenario

Step	Actor	Description	Data (including variables)	Expected result
1	shipper	Logs in Finest platform	Login data	Basic booking window is displayed
2	shipper	Selects the booking to be changed	Content of the information that is created by the system (e.g. list of offers received)	Booking window is displayed
3	shipper	Changes booking volume from 3-5	Information related to the notification	A change request is set up

			<i>(e.g. detail of a booking request)</i>	
4	shipper	Submits booking		A change request is being sent towards the forwarder and carrier
...

In addition to the test scenario steps, the business user might specify a KPI or set of KPIs (CKPI) to be calculated for later analysis of the results. The aim is to assess the degree of improvement (in terms of efficiency and effectiveness of business operations) achieved by the new collaborations enabled by FInest. In case an assessment of test performance is required, a set of KPI(s) is selected for the specific scenario (see 2.4.4).

2.4.2. Configure experiment

The tester/experimenter of FInest EE receives as input the test scenario and configures it in order to make it executable, i.e. goes over each of the steps and refines it if necessary so it can be executed as an *atomic step* in the EE. This step is required since business users might describe steps in different levels of granularity (sometimes too coarse grained for the scenario to be executable).

The definition of the execution parameters is done at the beginning of the test using a dedicated view in FInest EE frontend (see Figure 4). Note that the designed architecture also allows for assigning default values to data.

The output is an executable test that can be run by the tester using FInest EE, i.e. each row in it is an atomic step. This executable test can be conceptually seen as a "test template" which can serve as basis for new versions of the specific test or new tests (see Section 2.4.5).

The configuration phase includes for each step whether it is:

- Manual – The step is executed by the tester. It includes the table fields as described in Table 2.
- Injected data – Simulation of a step (s) using historical data. This is done by the selection of "injected data" in input view (see Figure 4).
- Vuser script – A pointer to the script to be run.

Table 2 shows the experiment snippet that corresponds to rows 1-2 in the test scenario of Table 1.

Table 2: Example for an experiment for the "real-time event handling" use case test scenario

Step	Actor	Description	Data (including variables)	Expected result
1	shipper	Logs in FInest platform into "Booking change application"		Login window is displayed
2	shipper	Type in the username field	Var1	
3	shipper	Type in the password field	Var2	

4	shipper	Select the user role filed	"shipper"	
5	shipper	Click the login button		Shipper is logged into the application
6	shipper	Click the search OrderID button		Search Order window is displayed
7	shipper	Selects the OrderID to be changed	In the OrderID field select Var3	Details of OrderID==Var3 are displayed
...

2.4.3. Execute experiment

The experiment defined in the *configure experiment* stage enables its execution by binding the parameters specified in the experiment to specific values. For example: *var1* in our example can be bounded to "John Brown".

The tester executes the experiment by sequentially performing each of the steps specified in the experiment, including Vuser scripts and injection of data into the execution. For each of the steps two additional fields are incorporated: *actual results and notifications*.

Actual results reflect the observed output of the execution of the step.

Notifications are messages sent to FInest front end during the execution of a test. We identify two types of notifications:

1. Messages or alerts - These are notifications sent by the BCM (Business Collaboration Module) or EPM (Event Processing Module) modules regarding actual or expected outcomes of the process being carried out. An example of a notification can be notifications sent to the forwarder and carrier once a booking modification has been requested by a shipper (refer to demonstrator 3 in FInest and D2.4 for details on the use case), as illustrated in the snippet screenshot from demonstrator 3 in Figure 2. Messages can sometimes being expected and therefore referred to in the expected results of the use case, but they can sometimes be result of proactive notifications of the EPM (refer to D6.3 regarding proactive event-driven computing).
2. Error messages displayed during the execution of a test.










Figure 2: Example of system notifications

An automated log regarding the actual execution of the test is generated by FInest EE *execution log manager* (see Figure 4).

Table 3 shows an example for the execution of the experiment in Table 2.

Table 3: Example of an experiment test after execution

Step	Actor	Description	Data	Expected result	Actual results		Notifications
1	Same as in Table 2			Login window is displayed		Login window is displayed	
2					John Brown is typed		
3					JBrown password is typed		
4					Shipper is selected		
5					John Brown is logged into the application		
6					Search order window is displayed		
7					Order 12345 details are not displayed	Error message "Order 1234 cannot be displayed"	
...			

2.4.4. Report

Analysis of the performance execution of the test can be done by the business user through the *reporting component* (see Figure 4Figure 3). The EE enables selecting which KPIs (either simple or composite) can be calculated out of the *execution logs*. In this phase, reports based on the KPIs are generated to enable the business user to analyze the performance of the test execution (note that this is an optional task, as described in D4.2). This phase includes two actions:

- Set-up/select the relevant KPI(s) (as appear in the test scenario)
- Generate report(s)

Note that the business user should explicitly state whether the KPIs should also be calculated on aborted or cancelled executions (see Section 2.4.6).

2.4.5. Re-using an existing test

A common situation is business users who want to test similar situations (collaborations) or even the same business collaboration with different variables values. This is possible in Finest EE by locating and uploading the relevant experiment (via the *Search* component, see Figure 4) and re-configuring it either to a new experiment or to a new version of the same experiment. This new test can then be executed.

2.4.6. Ending an execution

In general, each execution log can have one out of three possible statuses (recorded as part of the execution log):

- *Completed*: the test has been carried out and has been completed
- *Aborted*: At some point the test has been stopped by the EE (as a result of an "error" in FInest test)
- *Cancelled*: At some point the test has been stopped by the tester

2.5. Tightly connection with work package 2 "Use case specification"

There is a tightly connection between WP2 "Use case specification" and WP4 "Experimentation environment", since the designed experimentation environment needs to support the execution of the test scenarios specified in WP2. More specifically, D2.4 "Initial experimentation specification and evaluation methodologies for selected use case scenarios" is tightly related to D4.3 "Interim specification for transport and logistics experimentation environment" and there are two main intersection points that relate to the two main outcomes of D2.4:

- *Experimentation specification of use case scenarios* – WP2 (Task 2.3), and specifically the domain partners in this work package are accountable for the definition of the test scenarios (five selected use cases in FInest) to be executed in phase 2.
- *Evaluation methodologies for selected use case scenarios* – WP2 (Task 2.4). The evaluation framework devised in D2.4 will pave the way to the definition and specification of the KPIs to be stored in FInest EE. Moreover, this framework will facilitate the selection and analysis of the test performance based on the selected KPI(s).

These two intersection points are demonstrated in Figure 3. The business users in WP2 define the test scenarios to be executed in the experimentation environment, initializing the execution process (represented by the solid line rectangle). During the *report* task, business users will analyze the performance of the test based on the KPI(s) and assessment methodology defined in WP2 (represented by the dotted line rectangle).

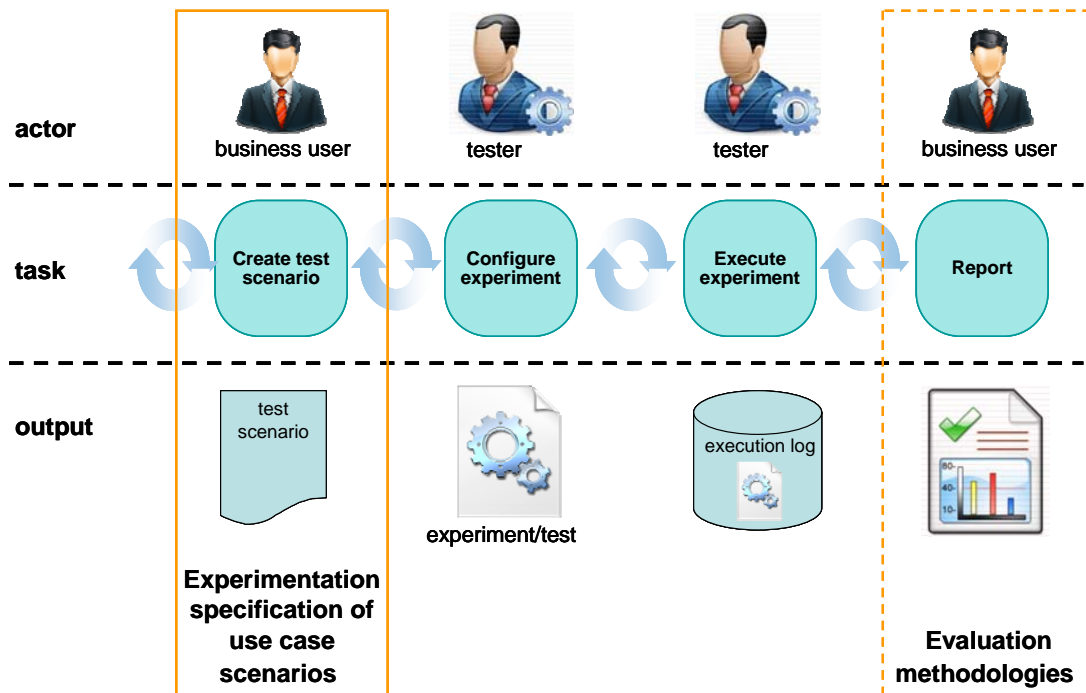


Figure 3: Intersections points between WP2 and WP4

3. Technical specification of Finest EE

In general, the envisioned experimentation environment will operate by activating the Finest platform and will invoke it at each test execution, utilizing Finest technologies and databases. It consists of three interconnected major components (see Figure 4 for an initial detailed architecture of the EE):

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

FInest EE front end includes the following high-level views (see Figure 4):

- *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 FInest Test system through a *backend simulator* component. This includes injecting data into FInest test and recording events and other data processed by FInest 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 FInest 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 FInest. The examination of build-versus-buy components will be part of the implementation plan to be submitted as D4.4 at M24.

Figure 4 presents FInest EE initial architecture followed by a description of the different modules and definition of the data types and interfaces. Members of WP4 closely worked with members of the technical team so that the architecture proposed is in-line with the technical specification defined for each of the technical deliverables of the project at M18. Please also note that the technical architecture depicted in Figure 4 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>

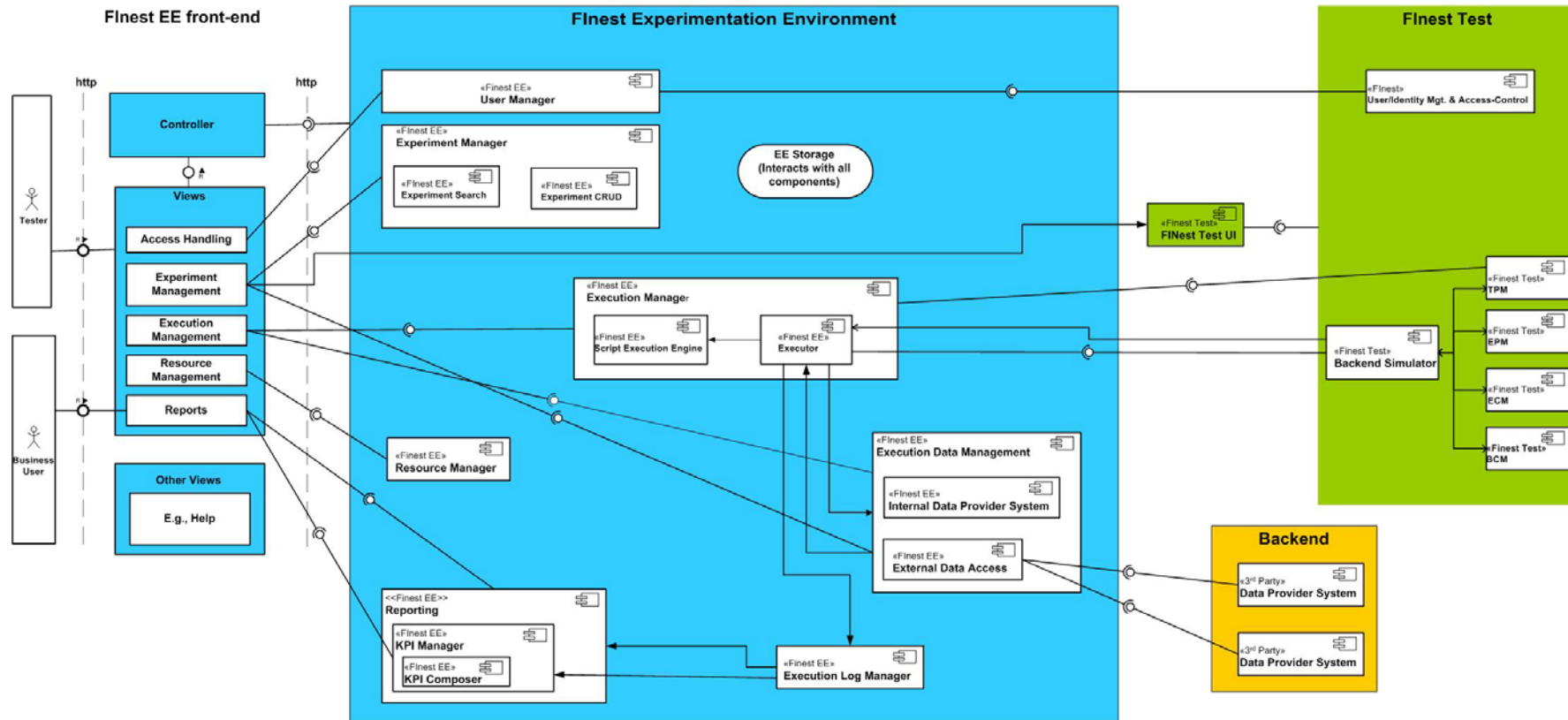


Figure 4: Finest experimentation environment architecture

3.1. FInest experimentation environment 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 FInest 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 FInest front-end and error messages.
- 2) *Script Execution Engine:* executes VUser scripts.

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.

- 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 FInest; provides APIs to inject data to the FInest Test system’s modules. Also reports back on events and other processed data.

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

3.2. 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() 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.	

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, 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..*	
GetDefaultDataPro vider 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
NewTCPRequired	Indicates whether a new TCP is required. The execution of such a step involves	

	the user clicking a link to access the FINest Test UI where the new shipment details are entered (see the sequence diagram). TCPs created in this manner are associated with the Execution instance.	
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()	Associates a TransportChainPlan Id with this execution.	TransportChainPlanId – the identifier of a TCP created as part of this experiment execution
GetAssociatedTCPI ds() void Public	Gets the TransportChainPlan identifiers associated with the execution	
BindVariable() void Public	Binds a variable name with a data provider	String – variable name DataProviderId
GetVariableBindings Map<String,DataProv iderId> 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. 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 for associated TCPs which were received while executing this step. Notifications conform to Finest formats	Notification[0..*]
DateTypes	Returns the data types of data received from the back-end simulator during execution. The indexes here must	DataType[0..*]

	match up with those of DataReceived	
DataReceived	Returns the data received from the back-end simulator during execution, which refer to an associated TCP. Indexes must match those for DataTypes. The data is in the appropriate standard Finest format (e.g. XML).	String[0..*]

7. 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	

3.3. Interfaces definitions

3.3.1. 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 Finest 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).	

2. 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.

3. 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	

3.3.2. Component interfaces

1. Executor

Method	Notes	Parameters
ExecuteStep() void Public	<p>Executes the current step, and logs output. Note for manual steps this doesn't 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 Finest 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.	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 given timeout will forcibly stop the script's execution	ScriptExecutionId 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

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(systemId).GetProvider(providerId)	DataProviderId – this is a pair of GUIDs, one for systemId and one for providerId

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 Public	Injects data to the appropriate Finest module.	DataType – the type of data. 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 Finest

		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	Associates the given TCP id to an execution.	Execution TransportChainPlan Id

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	<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

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()	Unarchives a composite KPI	String – name

void Public		
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 Finest 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.

3.4. Association of a TCP ID to a test

Finest aims at enabling new transport and logistic business collaborations based on future internet technologies. Finest platform is composed of four core modules to facilitate these new business collaborations (Refer to D3.2 "Conceptual Design of Domain-Specific FI Platform for Transport and Logistics"³): Business Collaboration Module (BCM), the Event Processing Module (EPM), the E-Contracting Module (ECM), and the Transport Planning Module (TPM).

The overall logistic process is described by the TCP which is the output of the TPM and serves as input for the BCM and EPM for the actual execution of the process. For cases in which we simulate the process, i.e. inject data and events into Finest EE, we need a mechanism to set-up a new TCP and associate it to a new test. This set-up scenario is detailed in Figure 5. The tester requests the initialization of a new TCP through Finest Test UI. The TPM, which simulates the set-up of a new transport plan, returns its ID back to the tester in order to carry out the test execution.

³ Available at <http://www.fi-ppp.eu/>

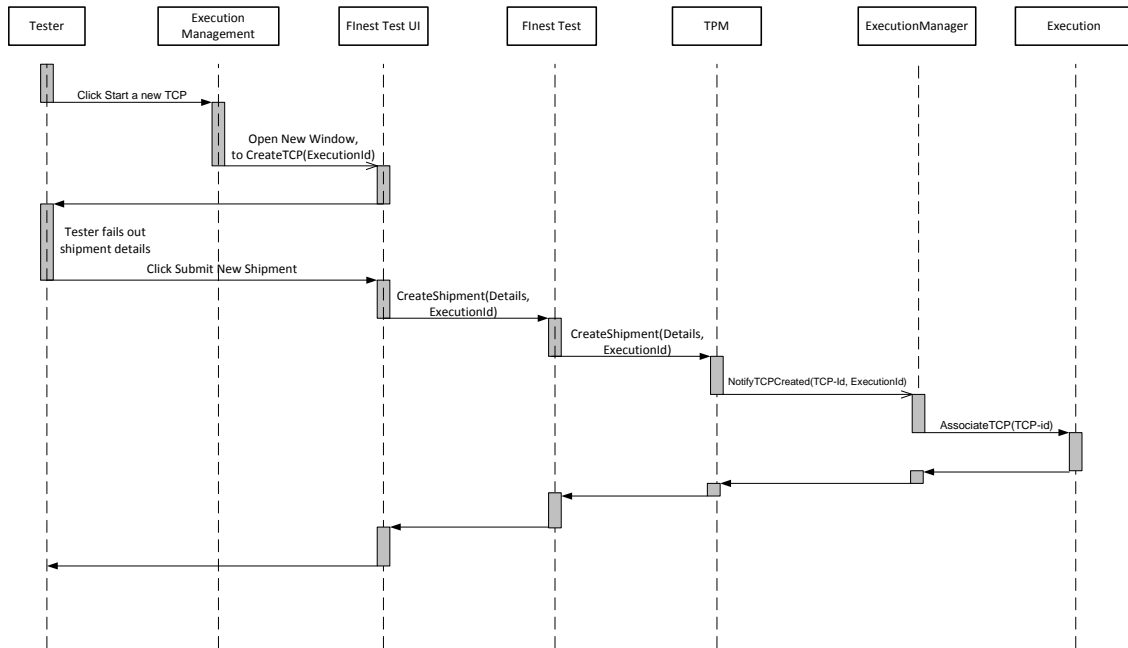


Figure 5: Association of a TCP ID to a test

4. Summary and next steps

Deliverable D4.3 "Interim specification for transport and logistics experimentation environment" is a straightforward continuation of D4.2 "Requirements and design of transport and logistics experimentation environment" and provides the first technical specification for the conceptual design given in D4.2. The proposed architecture meets all the requirements specified in D4.2 and enables the planning, execution, and analysis of Finest use cases in phase 2 and allows for large trials in phase 3.

The proposed EE enables execution of tests using physical sites as well as real data, and the injection of events and data into the system in cases in which real-time data cannot be accessed.

Next steps include a refinement of the proposed architecture based on developments in the project along with an implementation plan for phase 2 of the project.