

WP8 user requirements and system specifications

Grant Agreement N°: FP7 - SCP0 – GA – 2011 - 265647

Project Acronym: **ON-TIME**

Project Title: **Optimal Networks for Train Integration Management across Europe**

Funding scheme: Collaborative Project

Project start: 1 November 2011

Project duration: 3 Years

Work package no.: WP8

Deliverable no.: ONT-WP08-D-ANS-005-02; Rev 0

Status/date of document: Final, 31/10/2013

Due date of document: 31/10/2013

Actual submission date: 31/10/2013

Lead contractor for this document: Ansaldo
Genova, Italy

Project website: www.ontime-project.eu

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	
PP	Restricted to other program participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	X
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision control / involved partners

Following table gives an overview on elaboration and processed changes of the document:

Revision	Date	Name / Company short name	Changes
1	31/10/2013	Antonio Miano / ASTS	First issue

Following project partners have been involved in the elaboration of this document:

Partner No.	Company short name	Involved experts
1	NTT Data	Matteo Anelli
2	NTT Data	Daniele Carcasole
3	NTT Data	Bruno Ambrogio

Executive Summary

This document describes the user requirements of the OnTime Demonstrator.

The demonstrator can be considered the OnTime graphical user interface. It is made up of a Train Graph (TG), a Train Describer (TD), and a general purpose HMI.

TG and TD layouts depend on RailML data flow that comes from simulator. The first part of this document (see section 0 and section 2) describes how to convert the previous data in a TG/TD compatible format.

In section 3.1 and 0, communication protocol and user interactions of TG and TD are described.

HMI is described in section 0. The first two paragraphs are related to the communication protocol; the third shows a proposal of the HMI forms and their usage.

Table of contents

EXECUTIVE SUMMARY	3
TABLE OF CONTENTS	4
TABLE OF FIGURES	5
TABLE OF ABBREVIATIONS	6
1 INTRODUCTION.....	7
2 SYSTEM SPECIFICATION.....	8
2.1 Software requirements	8
2.2 Hardware requirements	8
2.3 Performance requirements	8
2.4 Development requirements	8
2.5 Environmental requirements.....	8
3 RAILML TO TD CONVERTER.....	9
3.1 Graphic formatter tool	9
3.2 RailML	9
3.3 SVG	9
3.4 Converter schema railML to SVG.....	10
4 RAILML TO ASTS ENVIRONMENT CONVERTER	14
4.1 Wayside elements	14
4.2 Logical elements	14
4.3 Mimic panels data	14
4.4 Controls and indications.....	14
4.5 ARS data	14
5 GRAPHICAL INTERFACES	15
5.1 Train Graph.....	15
5.1.1 Protocols	15
5.1.2 Interactions	17
5.2 Train Describer	18
5.2.1 Protocols	18
5.2.2 Interactions	20
5.3 HMI.....	21
5.3.1 Protocols - Events.....	21
5.3.2 Protocols - Cost Functions Parameters.....	27
5.3.3 Protocols - Data Provider.....	30
5.3.4 HMI forms description and interactions.....	33
6 REFERENCES.....	45

Table of figures

Figure 1 - Demonstrator general schema	7
Figure 2 - RailML to svg conversion process	10
Figure 3 - RailML to svg conversion process - Step 1	10
Figure 4 - RailML to svg conversion process - Step 2	11
Figure 5 - RailML to svg conversion process - Step 3	12
Figure 6 - Conversion process	13
Figure 7 - Example of animation logic for TD	13
Figure 8 - Cost Function Parameters Set and Retrieve	27
Figure 9 - Events Scenario	28
Figure 10 - System request - Type YN	34
Figure 11 - System Request - Type Q	34
Figure 12 - System Request - Type L	35
Figure 13 - Main demonstrator window	37
Figure 14 - System boot form	38
Figure 15 - Cost function parameters form	40
Figure 16 - Displaying a text	41
Figure 17 - Displaying a number	42
Figure 18 - Displaying a list	43

Table of abbreviations

ONT	ONTIME project code on the repository
TG	Train Graph
TD	Train Descriptor
RTTP	Real Time Traffic Plan
GUI	Graphical User Interface
RTC	Rational Team Concert
SVG	Scalable Vector Graphics
HMI	Human-Machine Interface

1 INTRODUCTION

Demonstrator is the module of the OnTime project that will be developed to give the project itself a Graphical User Interface (GUI).

The GUI will look like the one depicted at the top of the following picture.

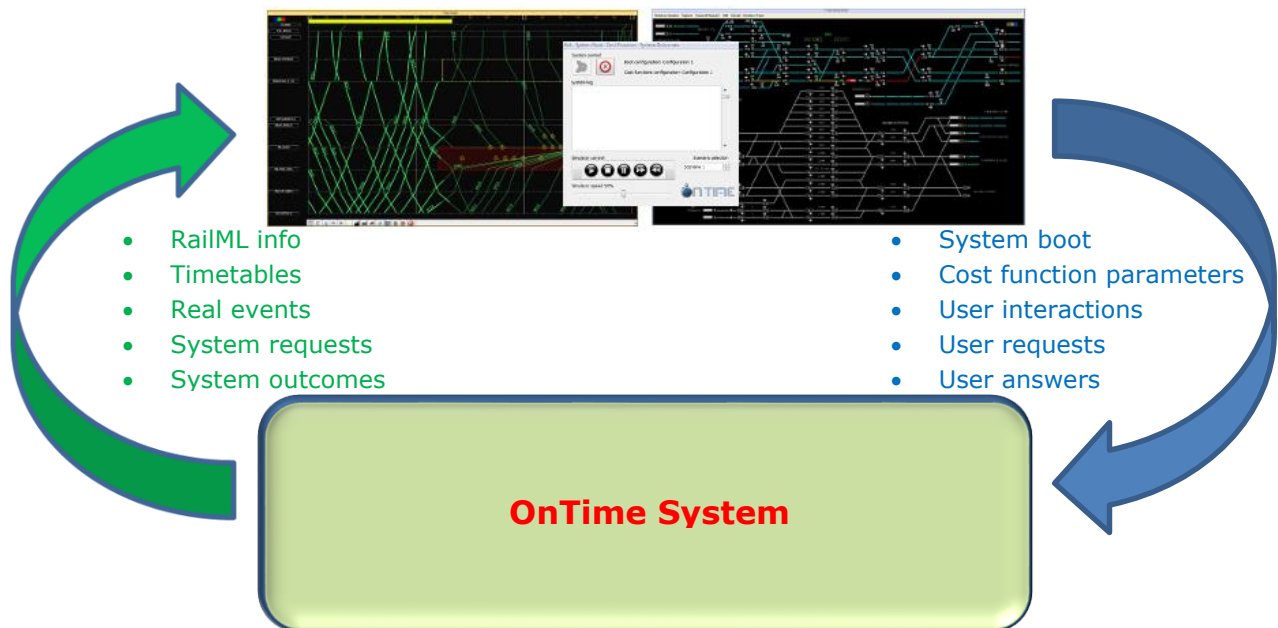


Figure 1 - Demonstrator general schema

The picture shows the Train Graph (TG), the Train Descriptor (TD) and the HMI main window as described in 0. For convenience, it has been represented as a floating window.

In green it is depicted the data flow from OnTime System to demonstrator, in blue the data flow from demonstrator to OnTime System.

SYSTEM SPECIFICATION

1.1 Software requirements

- The demonstrator shall be written in Java 1.7.
- The graphic library used shall be JavaFX.

1.2 Hardware requirements

- The workstation in which the system runs shall provide two monitors, one for TD and one for TG.
- The workstation in which the system runs shall have at least 4Gb of RAM.
- The workstation in which the system runs shall have at least a 3th generation I5 Intel processor.

1.3 Performance requirements

- The time between a user action and the demonstrator response shall be less than a second.
- The time between a system outcome and its representation on the demonstrator shall be less than a second.

1.4 Development requirements

- The tool used for the development shall be Netbeans 7.x.
- The tool used for versioning shall be RTC.

1.5 Environmental requirements

- Demonstrator shall be able to run on a Windows 8 workstation.

RAILML TO TD CONVERTER

1.6 Graphic formatter tool

The objective of this converter is to extract and transform geographic data of a rail network.

The conversion is based on reading a railML file and the production of one or more patterns in SVG format.

1.7 RailML

RailML is a generic language that can be used to describe railway-related data. The language specification contains subschemas for three main areas: infrastructure (with subsets for lines and stations), timetable (schedule), and rolling stock. These topics are themselves further divided into additional subschemas that address more specific areas.

It is based on XML. XML is not an application language, but rather a set of rules that can be used to define other “mark-up” language and therefore it acts as a meta language. The major advantage of XML based documents is that they describe both data as well as the data’s structure.

Therefore, XML is an ideal solution for transfer and storage of railroad data.

RailML is an open source data structure that has been developed to simplify the transfer of data between various railroad simulation and operations computer programs.

1.8 SVG

Scalable Vector Graphics (SVG) is a standard from the W3C which is built on top of XML. The SVG describes a language to draw graphics mainly on web pages.

The SVG recommendation comes from the W3C. The recommendation describes how to create vector graphics using a mark-up language. Because SVG is based on XML, tools that already know how to interpret XML will be able to interpret SVG. Having this well-established standard as a foundation saves a lot of the work in defining the language. It also means that there are numerous tools around which can check the structure of an SVG document or read it to pull out interesting information. Being built on XML also makes it simpler to produce SVG programmatically from other XML data sources using XSL and XSLT.

SVG graphics are vector graphics, so they can be resized without losing quality. A single SVG file can be scaled to any size or transformed to any resolution without compromising the clarity of the graphic. Bitmap images such as PNG and GIF lose quality any time they are resized.

Also, if you need to display the same graphic at multiple sizes or resolutions, you would need multiple bitmap images, but only one SVG file. SVG files display clearly at any size in any viewer or browser that supports SVG. Using browser controls, the user can zoom in to view details in a complicated SVG graphic.

An SVG file might also be smaller in size than the same graphic created by a bitmap (image) device such as GIF or PNG.

SVG files are ideal for producing any documents of any size to display on a computer monitor, PDA, or cell phone, or documents to be printed or delivered as PDFs.

1.9 Converter schema railML to SVG

The railway line described in railML is converted and, if necessary, processed to obtain many micro areas from a macro area.

Each area will be displayed by a Train Descriptor.

The conversion of the railML structure in the SVG format used by TDs is shown in Figure 2.



Figure 2 - RailML to svg conversion process

The process "Converter railML to SVG_{TD}" in the Figure 2 can be divided into three steps.

Step 1

In the first step, the railML elements (such as tracks, stations, signals, etc) are analysed and transformed into svg code (Figure 3).

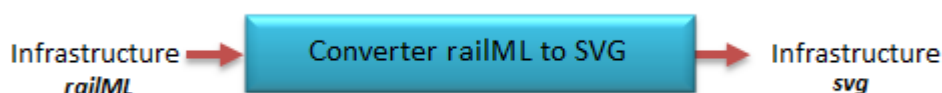


Figure 3 - RailML to svg conversion process - Step 1

The target is to get a set of files. Each file represents a micro areas as requested by the infrastructure manager.

This is not possible in automatic way; the operator must manually improve the graphics and decide the division of areas.

Step 2

The operator can make the proper changes on files to make them more understandable (step 2).

A program, called SVGEditorASTS, allow the operator to manage the elements and associate them to the logic of animation.

The Figure 4 represents the step 2.

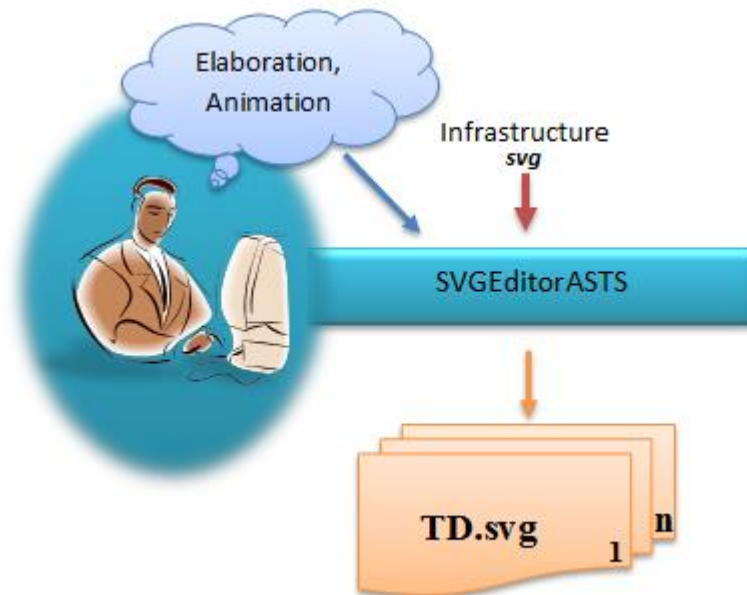


Figure 4 – RailML to svg conversion process – Step 2

In the second step, the operator:

- Elaborates the animation rules and the file Infrastructure.svg (the one that contains all the line) to define n files, each of them is a TD (a part of the line only).

An example of animation rule is represented in the table.



Type wayside point	Id event	Id animation state	Graphic
Track	Track_free	1	
Track	Track_occupation	2	

Table 1 - Animation rule example

This information shall be determined by infrastructure manager. It decides the logic of animation (colours, flashes) to be allocated to wayside points in certain conditions.

- Subsequently, SVGEditorASTS generates “n” TD.svg files adding the information related to animation.

Step 3

In Figure 5 is represented the last step.

In this step “n” zip files are generated, one for each TD and one for the animation logic; this last file is called “graphic library”.

These final TD files are all written in svg too, but now every element is linked to an “ancestor” of the graphic library from which it inherits a specific animation/behaviour.

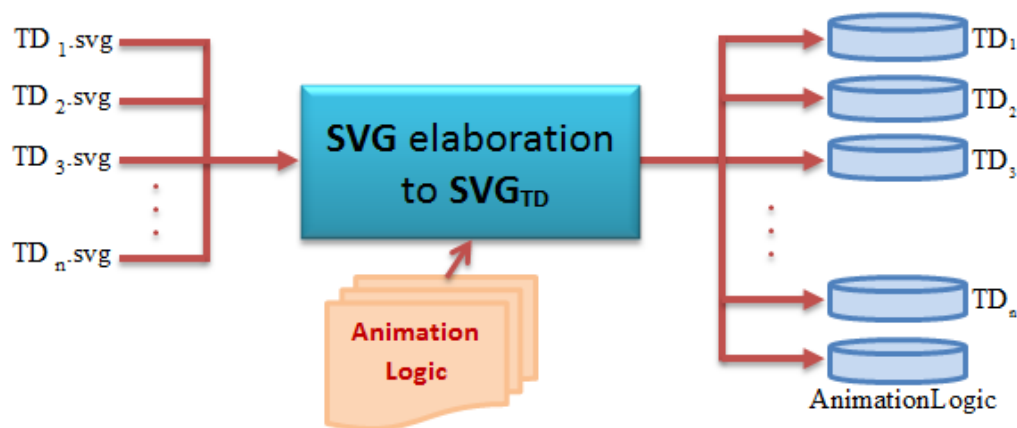


Figure 5 - RailML to svg conversion process – Step 3

Figure 6 shows the process of TD conversion.

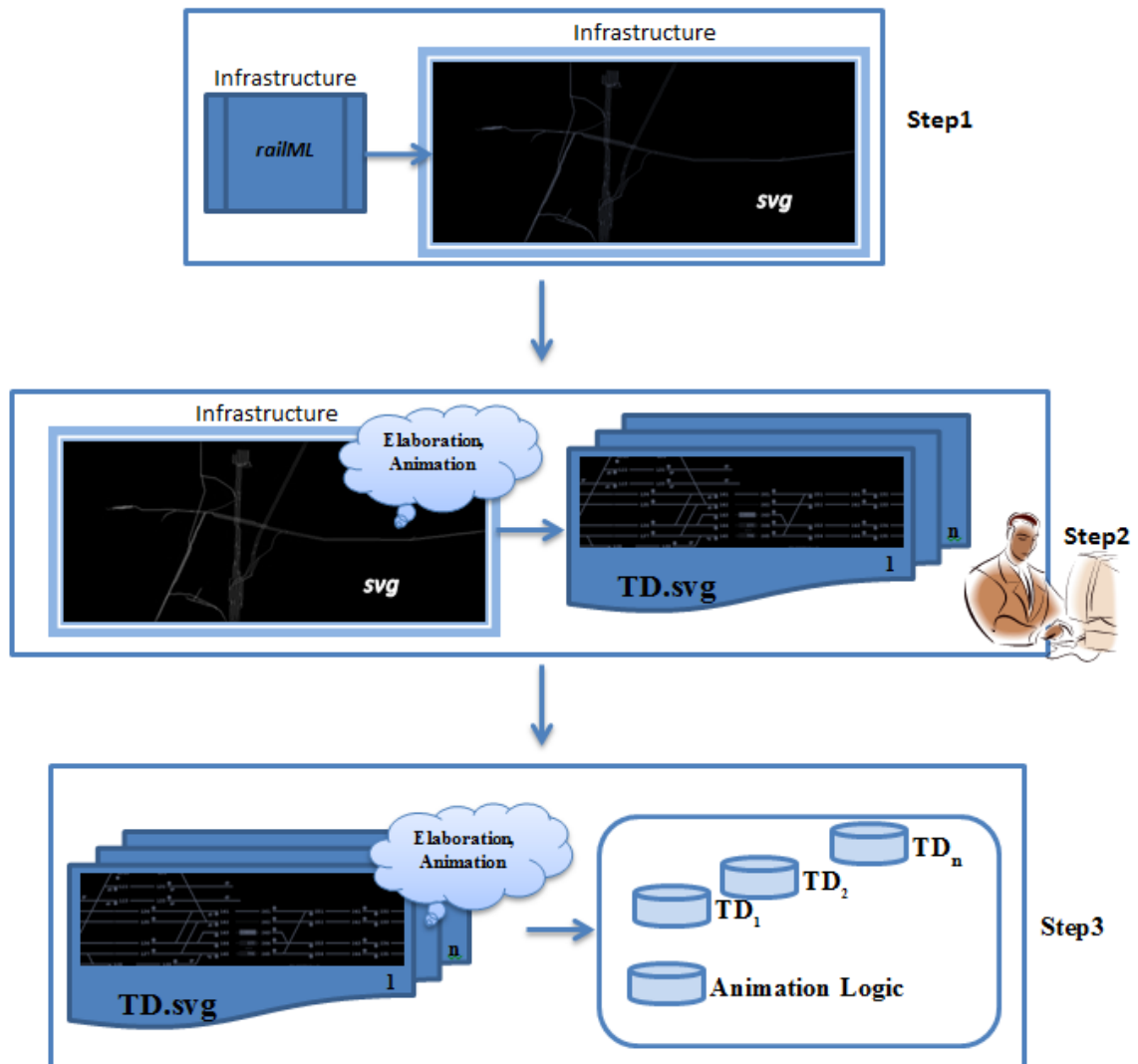


Figure 6 - Conversion process

In Figure 7 is shown how the animation logic works for a track (TrackId1) after a TDSectionOccupationEvent.

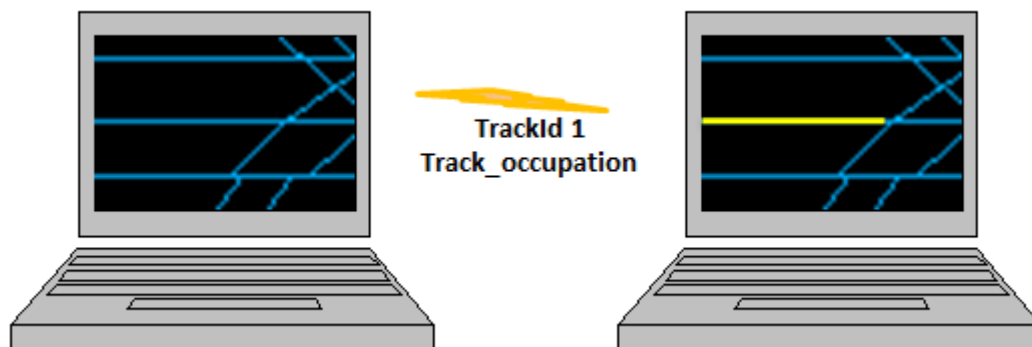


Figure 7 - Example of animation logic for TD

2 RAILML TO ASTS ENVIRONMENT CONVERTER

The ASTS data environment includes the following main elements:

- Wayside elements. They correspond to a physical railway device (signals, tracks, switch points);
- Logical elements. They correspond to “non-physical” elements that are typically an aggregation of physical elements (routes, platforms, block sections, train tracking berths);
- Mimic panels data;
- Controls and indications;
- ARS data. They are used by Automatic Route Setting and Regulation functions (platform priorities, preferred paths, alternative routes).

2.1 Wayside elements

These data can be obtained directly from the RailML stream. These elements are described in terms of id, name and relationships with other objects.

2.2 Logical elements

These data can be obtained elaborating the graph of wayside elements built up during the previous step. The elaboration process must rely on the following information.

- Infrastructure manager constraints
- Railway line constraints
- Tracks aggregation level

Due to the importance of this step, a further discussion must take place to identify all the RailML elements in more details. Beyond any doubt, this discussion will lead to a small refactoring of the RailML structure itself.

2.3 Mimic panels data

These data have already been described in paragraph 0 above.

2.4 Controls and indications

These data are not needed for this project.

Information about the state of wayside elements is provided by the simulator.

2.5 ARS data

These data are not needed for the Demonstrator.

The ARS and regulation logic is provided by WP4 and WP5.

3 GRAPHICAL INTERFACES

3.1 Train Graph

3.1.1 Protocols

This paragraph describes all data and events that shall be used by Train Graph.

3.1.1.1 GetInfrastructureData (DataProvider interface)

It provides the following information along with their geographical position:

- Tracks
- Lines
- Stations
- Signals
- Switch Points

As far as TG is concerned, only the distances between stations are important to draw the graph.

3.1.1.2 GetInfrastructureUnavailability (DataProvider interface)

It provides all information needed to draw disruptions at start time. At run time the disruption events shall be used.

3.1.1.3 GetTimetables (DataProvider interface)

It provides all information needed to draw the theoretical train tracks.

3.1.1.4 GetTrainDelayInfo (DataProvider interface)

It provides the delay for one or more trains. It shall be used to show the delay info for every train.

3.1.1.5 GetTimetableDelayInfo (DataProvider interface)

It provides the delay timetable for every train. It shall be used to show the delay info for every train.

3.1.1.6 GetRealTimeTrafficPlan (DataProvider interface) RTTP

It includes:

- Active lines
- Current Trains
- Timetables

It provides all information needed to draw all trains at start time. At run time the `TrainPositionChangeEvent` shall be used. The RTTP might be used in a polling loop in order to achieve the same behavior.

3.1.1.7 *TrainPositionChangeEvent (Event)*

It provides (asynchronously):

- Id of the train
- Position of the train head
- List of the occupied tracks

It shall be used to update TG train visualization. If the train is arriving/departing to/from a station, the corresponding line will be thickened. This shall be possible correlating train position with station position.

3.1.1.8 *TrainSuppressedEvent (Event)*

This event is fired when a train has been suppressed.

It provides:

- ID of the Train
- Time at which the train has been suppressed After the reception of this event, the corresponding train shall be cancelled from the graph.

3.1.1.9 *LineDisruptionEvent (Event)*

This event is fired in case of line disruptions.

It provides:

- Id of the RailML Line representation
- Starting time of the disruption
- Time at which the disruption will end (for planned operations)
- Reason of the disruption

Disruption starting time and, possibly, ending time shall be used to calculate the width of the box that will represent the disruption itself.

The geographical extension of the disruption (from the RailML ID) shall be used to determine the height of the box.

3.1.1.10 *TrackDisruptionEvent (Event)*

This event is fired in case of track disruptions.

It provides:

- Id of the Track
- Starting time of the disruption
- Time at which the disruption will end (for planned operations)

- Reason of the disruption

Disruption starting time and, possibly, ending time shall be used to calculate the width of the box that will represent the disruption itself.

The geographical extension of the disruption (from the RailML ID) shall be used to determine the height of the box.

3.1.1.11 StationDisruptionEvent (Event)

This event is fired in case of station disruptions.

It provides:

- Id of the Station
- Starting time of the disruption
- Time at which the disruption will end (for planned operations)
- Reason of the disruption

Disruption starting time and, possibly, ending time shall be used to calculate the width of the red line, along the station, that will represent the disruption itself.

3.1.1.12 TemporarySpeedRestrictionEvent (Event)

This event is fired in case of temporary speed restrictions.

It provides:

- List of IDs of the RailML representation of Tracks and/or Lines
- Speed restriction value

The geographical extension of the disruption (from the RailML ID list) shall be used to determine the height of the speed restriction box.

3.1.1.13 ConnectionConflictEvent (Event)

This event is fired in case of connection conflicts.

It provides:

- Conflicting trains
- Conflict time
- Conflict location

A yellow target shall be placed at time/location coordinates on TG to indicate the presence of the conflict.

3.1.2 Interactions

TG can interact with the user in several ways. Basically, using this task, it is possible to perform a subset of actions that HMI can perform. The most important difference is that the object to manipulate can be chosen simply by selecting the objects itself from the graphic and applying the correct action by choosing from a menu.

For instance, if the user wants to change the real time event for a train, using the HMI he/she has to input somewhere the train number, using TG he/she has just to select the train from the graphic and choose "change time" action from a menu.

A list of possible actions that can be initiated by TG follows.

- Change a train theoretical timetable
- Change real events time for a train (departure time, arrival time)
- Select a new path (this is possible only using the Path Selection HMI)
- Solve a conflict
- Get data about slowdowns and disruptions
- Get the list of trains that will cross a station

What actions are really necessary for OnTime is yet a matter of discussion.

3.2 Train Describer

3.2.1 Protocols

3.2.1.1 GetInfrastructureData (DataProvider interface)

It is the same interface used by TG. In this case all information (tracks, lines, stations, signals, switch points) shall be used to draw the graph.

3.2.1.2 GetInfrastructureUnavailability (DataProvider interface)

It is the same interface used by TG.

It provides all information needed to draw disruptions at start time. At run time the disruption events shall be used.

3.2.1.3 TrainPositionChangeEvent (Event)

This event is fired when a train changes its position.

It provides:

- ID of the train
- Position of the train head
- List of the occupied tracks

ID, position and tracks shall be used by TD to draw the train ID and to show in red all the occupied tracks. Tracks ID shall be the same used in the RailML data stream.

3.2.1.4 TrainSuppressedEvent (Event)

It is the same event used by TG.

After the reception of this event, the corresponding train shall be cancelled from the graph.

3.2.1.5 TrainEnterEvent (Event)

This event is fired when a train entered the scenario.

It provides:

- ID of the train
- Time at which the train entered

The train shall be drawn on a graph input point.

3.2.1.6 TrainExitEvent (Event)

This event is fired when a train left the scenario.

It provides:

- ID of the train
- Time at which the train left the scenario

The train shall be drawn on a graph output point.

3.2.1.7 LineDisruptionEvent (Event)

It is the same event used by TG.

The corresponding line on TD shall be marked as unavailable.

3.2.1.8 StationDisruptionEvent (Event)

It is the same event used by TG.

The corresponding station on TD shall be marked as unavailable.

3.2.1.9 PlatformDisruptionEvent (Event)

It is the same event used by TG.

The corresponding platform on TD shall be marked as unavailable.

3.2.1.10 TrackDisruptionEvent (Event)

It is the same event used by TG.

The corresponding track on TD shall be marked as unavailable.

3.2.1.11 SignalStateChangeEvent (Event)

This event is fired when a signal changes its state.

It provides:

- ID of the signal
- Signal aspect

The aspect of the corresponding signal on TD shall be changed accordingly.

3.2.1.12 *TdSectionOccupationEvent (Event)*

This event is fired when a section is occupied by a train.

It provides:

- ID of the section
- Train ID
- Time of occupation

It can be used in conjunction with *TrainPositionChangeEvent* to determine the which track must be drawn in red (occupied).

3.2.1.13 *TdSectionReleaseEvent (Event)*

This event is fired when a section, previously occupied by a train, is now released.

It provides:

- ID of the section
- Train ID
- Time of release

It can be used to determine which track must be drawn in the status of released.

3.2.1.14 *SetRouteEvent (Event)*

This event is fired when a route is set for a train.

It provides:

- Route ID

It shall be used to draw in yellow a route set for a train.

3.2.1.15 *UnsetRouteEvent (Event)*

This event is fired when a route is unset for a train.

Up to now this event is not provided.

3.2.2 Interactions

Regarding TD interactions, the same considerations about TG (see 3.1.2) can be made.

A list of possible actions that can be initiated by TD follows.

- Identify a train
- Cancel a train
- Substitute a train with a different one
- Switch two trains
- Select a new path (this is possible only using the Path Selection HMI)
- Set a route

What actions are really necessary for OnTime is yet a matter of discussion.

3.3 HMI

3.3.1 Protocols - Events

All events are implemented using the publish and subscribe paradigm and provide a standard data structure that is the following:

- Event ID [String] (can be incremented automatically)
- Event name [String]
- Version ID [int]
- Type ID [String] (needed by communication infrastructure to manage events)
- Timestamp [String]
- XmlPayload [String] (general purpose area defined by a proper xsd file)

In the description of the following events this standard structure is not shown. The “provides” section is related only to the structure defined inside the XmlPayload field.

3.3.1.1 UserRequestWPxEvent (Event)

This event is fired when the user needs to input data into the system. The subscriber of this event is WPx. Demonstrator (WP8) is the publisher.

It provides by means of XmlPayload:

- ID of operation to execute into the previous tasks
- Parameters of the previous operation

Although every WP can receive every operation request, it is clear that only a WP is responsible to answer the request itself.

Example. The “AddTrain” operation can be sent to WPa and WPb (UserRequest-WPaEvent and UserRequestWPbEvent respectively) but only one of the two can manage the request.

Operations list will be provided in the next paragraph.

3.3.1.2 SystemRequestEvent (Event)

This event is fired when the system needs to send data to HMI. The subscriber of this event is the demonstrator (WP8). Every WP can be the publisher.

It provides by means of XmlPayload:

- ID of the tasks that initiated the request
- Textual description of the operation
- ID of the operation to execute into HMI
- Parameters of the previous operation

Operations list will be provided in the next paragraph.

3.3.1.3 Operations

This paragraph shows the operations that can be performed by HMI and the related parameters used in the operation context.

The complete operations list is presented in the following table.

Operation	Direction	Purpose	Type
AddTrain	HMI=>Sys	To add a new train into the system	Next
CancelTrain		<i>Not necessary because it is possible to use the TrainSuppressed Event.</i>	Next
ModifyTime	HMI=>Sys	To modify a real/theoretical departure/arrival time	Rttp
ModifyDelay	HMI=>Sys	To modify/add a train delay	Next
ManageDisruption	HMI=>Sys	To add/modify/cancel a line/station disruption	Next
ManageSlowdown	HMI=>Sys	To add/modify/cancel a line/station slowdown	Next
ManageLink	HMI=>Sys	To add/modify/cancel a link between two trains	Next
ManageConstraint	HMI=>Sys	To add/modify/cancel a constrain	Next
SelectNewPath	HMI=>Sys	To select a new path for a train	Next
SolveConflict	HMI=>Sys	To solve a train conflict (using the possible solutions provided)	Next
UserResponse	HMI=>Sys	To use in response of an SystemRequest	Yes
SystemRequest	Sys=>HMI	To allow the user to answer a question with "Yes" or "No". To allow the user to choose one or more elements from a list. To allow the user to answer a question with a simple string. (See UserResponse)	Yes
ShowData	Sys=>HMI	To display data in a widget inside an HMI form To show system outcomes	Yes

Table 1 - HMI operations

The meaning of the "type" column is the following:

- 1) Next. Although HMI might manage the operation, due to non-interactively nature of the OnTime system, it has been chosen to use it in a future version.
- 2) Rttp. For now, Rttp module is the only in charge of dealing whit this data.
- 3) Yes. It is present in the actual version.

Because of the importance of cost functions in the OnTime project, the description of the events related to them are described in a separate paragraph (3.3.2).

3.3.1.4 addTrain

Direction: from HMI to System

Purpose: to add a new train into the system

UserRequestWPxEvent XmlPayload:

- Operation ID: addTrain
- Parameters: trainID, departure station, departure time, arrival station, arrival time
 - Example: "4568,Boden,18:45,Lulea,19:30" (The new train 4568 will depart from Boden at 18:45 and will arrive in Lulea at 19:30)

3.3.1.5 cancelTrain

Not necessary because it is possible to use the TrainSuppressed Event.

3.3.1.6 modifyTime

Direction: from HMI to System

Purpose: To modify a real/theoretical departure/arrival time

UserRequestWPxEvent XmlPayload:

- Operation ID: modifyTime
- Parameters: trainID, station, event, new time
 - Example: "4569,Boden, A, 18:50" (The train 4569 will arrive/arrived in Boden at 18:50)

If the train is in the future the new time will substitute the previous predicted time, if it is in the past it will substitute the previous real time.

3.3.1.7 modifyDelay

Direction: from HMI to System

Purpose: To modify/add a train delay

UserRequestWPxEvent XmlPayload:

- Operation ID: modifyDelay

- Parameters: trainID, station, event, delay in minutes
 - Example: "4567,Boden,D,34" (Departure from Boden of train 4567 will be delayed of 34 minutes)

3.3.1.8 manageDisruption

Direction: from HMI to System

Purpose: To add/modify/cancel a line/station disruption

UserRequestWPxEvent XmlPayload:

- Operation ID: manageDisruption
- Parameters: disruptionID, operation (A - add, M - modify, C - cancel), type (L - line, S - station), direction (O - Odd, E - Even, B - Both), start datetime, end datetime, start station, end station
 - Example: "X, A, L, O, 11/12/2013 12:30, 13/12/2013 18:00, Lulea, Boden" (To add [A] a new line [L] disruption [X means yet unknown ID] from Lulea to Boden, odd [O] track, starting at 12:30 on 11/12/2013 and ending at 18:00 on 13/12/2013)
 - Example: "D1234, C" (To cancel the disruption D1234)
 - Example: "D1235, M, S, B, 14/12/2013 05:00, 16/12/2013 03:00, Boden" (To modify the station [S] disruption D1235 in Boden, both [B] tracks, now starting at 05:00 on 14/12/2013 and ending at 03:00 on 16/12/2013)

3.3.1.9 manageSlowdown

Direction: from HMI to System

Purpose: To add/modify/cancel a line/station slowdown

UserRequestWPxEvent XmlPayload:

- OperationID: manageSlowdown
- Parameters: slowdownID, operation (A - add, M - modify, C - cancel), type (L - line, S - station), direction (O - Odd, E - Even, B - Both), start datetime, end datetime, start station, end station
 - Example: "X, A, L, E, 11/12/2013 12:30, 13/12/2013 18:00, Lulea, Boden" (To add [A] a new line [L] slowdown [X means yet unknown ID] from Lulea to Boden, even [E] track, starting at 12:30 on 11/12/2013 and ending at 18:00 on 13/12/2013)
 - Example: "S1234, C" (To cancel the slowdown S1234)
 - Example: "S1235, M, S, O, 14/12/2013 05:00, 16/12/2013 03:00, Boden" (To modify the station [S] slowdown D1235 in Boden, odd [O] track, now starting at 05:00 on 14/12/2013 and ending at 03:00 on 16/12/2013)

3.3.1.10 manageLink

Direction: from HMI to System

Purpose: To add/modify/cancel a link between two trains

UserRequestWPxEvent XmlPayload:

- Operation ID: manageLink
- Parameters: trainID1, trainID2, operation (A - add, M - modify, C - cancel), station, min time, max time, link type (R – rolling stock, C – crew, N - connect)
 - Example: 123, 456, A, Boden, 2, 5, R (to add rolling stock link between train 123 and 456 in Boden, minimum wait time 2 minutes, maximum wait time 5 minutes)

3.3.1.11 manageConstraint

Direction: from HMI to System

Purpose: To add/modify/cancel a constrain

UserRequestWPxEvent XmlPayload:

- Operation ID: manageConstraint
- Parameters: trainID,

3.3.1.12 selectNewPath

Direction: from HMI to System

Purpose: To select a new path for a train

UserRequestWPxEvent XmlPayload:

- Operation ID: selectNewPath
- Parameters: trainID, pathID

3.3.1.13 solveConflict

Direction: from HMI to System

Purpose: To solve a train conflict

UserRequestWPxEvent XmlPayload:

- OperationID: solveConflict
- Parameters: solutionID (one of the solutions got via PossibleConflictSolutionsResponse)

3.3.1.14 userResponse

Direction: from HMI to System

Purpose: To use in response of an SystemRequest

UserRequestWPxEvent XmlPayload:

- OperationID: userResponse
- Parameters: requestID, answerString
 - Example: R1234, Yes (answer to a Yes/No systemRequest)
 - Example: R1234, item1:item2:item3 (answer to a choose element SystemRequest)
 - Example: R1234, user response (answer to a simple question SystemRequest)

3.3.1.15 systemRequest

Direction: from System to HMI

Purpose: To allow the user to answer a question with "Yes" or "No".

To allow the user to choose one or more elements from a list.

To allow the user to answer a question with a simple string.

SystemRequestEvent XmlPayload:

- Task ID: identifier of the task that performed the request
- Description: request description (showed to the user by HMI)
- OperationID: systemRequest
- Parameters: requestID, type, (YN - Yes/No, L - List, Q - Question), dataset
 - Example: R1234, YN (to ask "Yes" or "No" to the question in description)
 - Example: R1234, L, item1:item2:item3:item4 (to choose one item of the list)
 - Example: R1234, Q (the answer to the question in description will be a free text)

3.3.1.16 showData

Direction: from System to HMI

Purpose: To display data in a widget inside an HMI form

To show system outcomes

SystemRequestEvent XmlPayload:

- Task ID: identifier of the task that performed the request
- Description: not required
- OperationID: showData
- Parameters: formID, widgetID, type (T - text, N - number, L - list (text+number), [others can be added]), data
 - Example: logForm, logWidget, T, "Current trains number: 56" (to show the text [T] "Current trains number: 56" in the logWidget into the logForm)

Note. Type and data structures can be chosen properly to display every kind of data using different visualization type.

3.3.2 Protocols - Cost Functions Parameters

A general formulation of an optimization algorithm focuses on an objective function that must be minimized or maximized, subject to a set of constraints. The objective function is likely to rely on some parameters (e.g., weights) which regulates the behavior of the algorithm and the structure of solutions.

Of course, this is the case with all the On-Time optimization modules. Hence, the system must allow the user to change interactively those parameters, in order to force algorithms to give more priority to some aspects before others.

According to the key-concepts of the WP7 architecture design, modules must be kept decoupled, thus they can subscribe to events and start working without the architecture being aware of their internal structure or configurations. Moreover, a hot-swap of modules should be made possible by the architecture.

We have then that cost function parameters could change over time, both in values and in structure. Moreover, the system should take into account that a crash of an optimization module could occur and, as soon as it restarts, the last configuration of cost function parameters must be set again. This leads to the involvement of the WP7 Data Provider module, which will persist those data for these needs.

The following activity diagram shows the dynamics of a computing module:

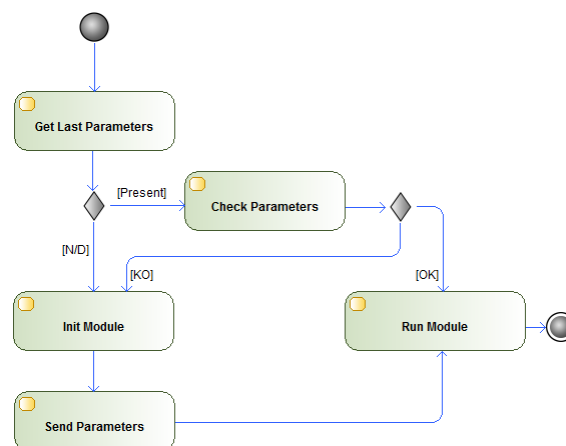


Figure 8 - Cost Function Parameters Set and Retrieve

First of all, a module must query the WP7 Data Provider in order to retrieve the last cost function parameters stored at it. Two scenarios are possible:

- The architecture does not have any previously stored values, and send a “not found” response. This happens at the first run of a module. In this case, the module will retrieve its internal default configuration of cost function parameters and will send back this configuration to the architecture.

- After a crash-and-restart, or a hot-swap of a module, the architecture maintains a copy of the last cost function configuration, and send it to the module. In case of a restart of a module, this configuration is likely to have the correct structure and values, but if the module has been replaced, probably it is not. The module must then execute a check procedure, in order to ensure that what it received is correct. If it is not, the module will act like a first-run, as described above. Otherwise, the module starts working directly, without sending back its actual cost function configuration.

A new configuration of cost function parameters is issued by a *CFParametersChange* event (3.3.2.1), to which the WP7 architecture is subscribed. Upon the receipt of such event, the received configuration is stored in the database and a *CFParametersConfigAvailable* event (3.3.2.3) is issued and received by all subscribed modules, that are thus notified of a new configuration available at the WP7 Data Provider. The following picture summarizes this scenario.

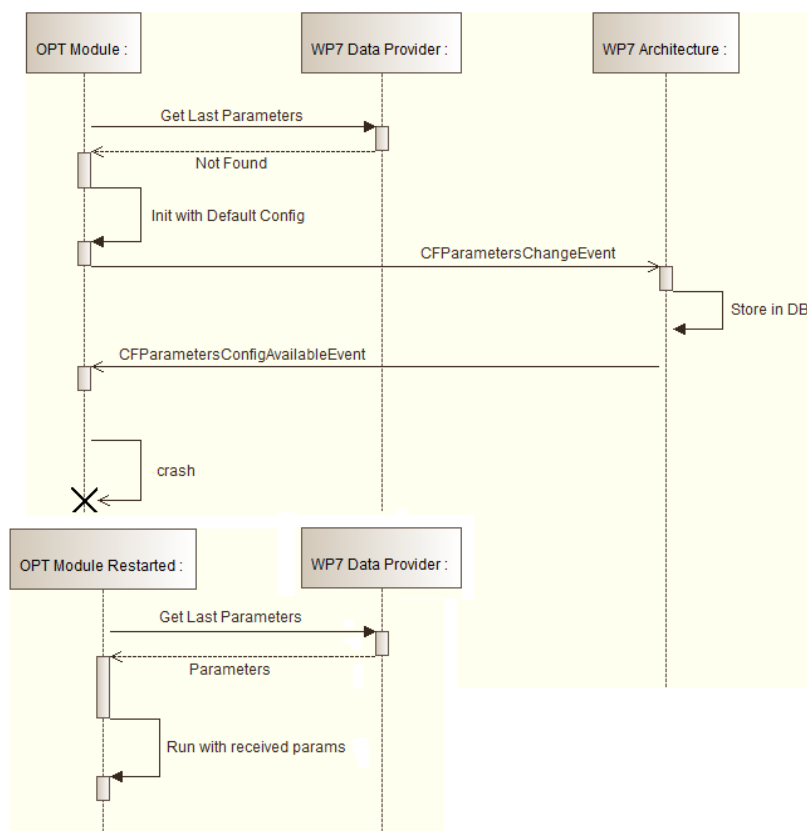


Figure 9 - Events Scenario

3.3.2.1 Changing parameters through HMI

The parameters can be changed at any moment during the system execution by HMI's Interface.

First of all the HMI must query the WP7 Data Provider in order to retrieve the last cost function parameters stored at it. This query must be done for every WP the HMI wants to manage.

For each WP two scenarios are possible:

- The architecture does not have any previously stored values, and send a “not found” response. This happens if the module had never run before. In this case the HMI interface for managing parameters for that specific WP will not be enabled.
- The architecture has a set of parameters for that specific WP and gives them to HMI in the response. On the base of the response the HMI creates the interface to manage the parameters for the specific WP (**Fel! Hittar inte referenskälla.**).

Once retrieved the parameters for a WP, using the HMI interface, a user can change the parameters' value. After confirming the change the HMI fires a *CFParametersChange* event (3.3.2.1).

The events, requests and responses involved in this process are described in detail in the next four paragraphs.

3.3.2.2 *CFParametersChangeEvent*

Direction: from HMI/OPT to System

Purpose: To set one or more tuples

CFParametersChangeEvent XmlPayload:

- OwnerWP
- Parameters - List of tuples:
 - DataType
 - Key
 - Value

3.3.2.3 *CFParametersConfigAvailableEvent*

Direction: from WP7 to System

Purpose: To confirm that all tuples have been correctly stored to the DB.

CFParametersConfigAvailableEvent XmlPayload:

- OwnerWP

3.3.2.4 *GetLastParametersRequest*

Direction: from HMI/OPT to DataProvider

Purpose: To request the list of cost function parameters

GetLastParametersRequest XmlPayload:

- OwnerWP (the WP who makes the request)

3.3.2.5 ParametersResponse

Direction: from DataProvider to System

Purpose: To send the list of cost function parameters

ParametersResponse XmlPayload:

- OwnerWP (the WP who makes the request)
- Parameters - List of tuples:
 - DataType
 - Key
 - Value

3.3.3 Protocols – Data Provider

The OnTime data source is the so called Data Provider and, in this case, it is not used the standard events publish/subscribe mechanism.

All data requests/responses are synchronous and don't provide a standard data structure. It depends on the nature of the request/response. When required, the related payload is referred as DataRequestPayload and DataResponsePayload.

The complete data requests/responses list is presented in the following table.

Data request/response	Direction	Purpose	Type
DisruptionsListRequest	HMI=>Sys	To request the disruptions list to the system (see DisruptionsListResponse)	Yes
SlowdownsListRequest	HMI=>Sys	To request the slowdowns list to the system (see SlowdownsListResponse)	Yes
LinksListRequest	HMI=>Sys	To request the slowdowns list to the system (see LinksListResponse)	Yes
ConstraintsListRequest	HMI=>Sys	To request the slowdowns list to the system (see ConstraintsListResponse)	Yes
PossibleTrainPathsRequest	HMI=>Sys	To request a list of possible train path (see PossibleTrainPathsResponse)	Next
PossibleConflictSolutionsRequest	HMI=>Sys	To request a list of possible conflict solutions (see PossibleConflictSolutionsResponse)	Next
DisruptionsListResponse	Sys=>HMI	To send the list of disruptions (see DisruptionsListRequest)	Yes

SlowdownsListResponse	Sys=>HMI	To send the list of slowdowns (see SlowdownsListRequest)	Yes
LinksListResponse	Sys=>HMI	To send the list of links (see LinksListRequest)	Yes
ConstraintsListResponse	Sys=>HMI	To send the list of constraints (see ConstraintsListRequest)	Yes
PossibleTrainPathsResponse	Sys=>HMI	To send a list a possible paths for a train (see PossibleTrainPathsRequest)	Next
PossibleConflictSolutionsResponse	Sys=>HMI	To send a list of possible conflict solutions (see PossibleConflictSolutionsRequest)	Next

Table 2 - HMI data requests/responses list

The meaning of the "type" column is the same of the previous table.

Because of the importance of cost functions in the OnTime project, the description of the requests/responses related to them are described in a separate paragraph (3.3.2).

3.3.3.1 disruptionsListRequest

Direction: from HMI to DataProvider

Purpose: To request the disruptions list to the system

DataRequestPayload:

- Parameters: none

3.3.3.2 slowdownsListRequest

Direction: from HMI to DataProvider

Purpose: To request the slowdowns list to the system

DataRequestPayload:

- Parameters: none

3.3.3.3 linksListRequest

Direction: from HMI to DataProvider

Purpose: To request the links list to the system

DataRequestPayload:

- Parameters: none

3.3.3.4 constraintsListRequest

Direction: from HMI to DataProvider

Purpose: To request the constraints list to the system

DataRequestPayload:

- Parameters: none

3.3.3.5 possibleTrainPathsRequest

Direction: from HMI to DataProvider

Purpose: To request a list of possible train path

DataRequestPayload:

- OperationID: possibleTrainPathsRequest
- Parameters: none

3.3.3.6 possibleConflictSolutionsRequest

Direction: from HMI to DataProvider

Purpose: To request a list of possible conflict solutions

DataRequestPayload:

- OperationID: possibleConflictSolutionsRequest
- Parameters: none

3.3.3.7 disruptionsListResponse

Direction: from DataProvider to HMI

Purpose: To send the list of disruptions

DataResponsePayload:

- Task ID: identifier of the task that performed the request
- Parameters: disruptions list

3.3.3.8 slowdownsListResponse

Direction: from DataProvider to HMI

Purpose: To send the list of slowdowns

DataResponsePayload:

- Task ID: identifier of the task that performed the request
- Parameters: slowdowns list

3.3.3.9 *linksListResponse*

Direction: from DataProvider to HMI

Purpose: To send the list of links

DataResponsePayload:

- Task ID: identifier of the task that performed the request
- Parameters: slowdowns list

3.3.3.10 *constraintsListResponse*

Direction: from DataProvider to HMI

Purpose: To send the list of constraints

DataResponsePayload:

- Task ID: identifier of the task that performed the request
- Parameters: constraints list

3.3.3.11 *possibleTrainPathsResponse*

Direction: from DataProvider to HMI

Purpose: To send a list a possible paths for a train

DataResponsePayload:

- Task ID: identifier of the task that performed the request
- Description: not required
- OperationID: disruptionListResponse
- Parameters: possible train paths list

3.3.3.12 *possibleConflictSolutionsResponse*

Direction: from DataProvider to HMI

Purpose: To send a list of possible conflict solutions

DataResponsePayload:

- Task ID: identifier of the task that performed the request
- Description: not required
- OperationID: disruptionListResponse
- Parameters: possible conflict solutions list

3.3.4 HMI forms description and interactions

This paragraph will describe all the HMI forms and their user interactions. Where possible, a draft form layout will be shown.

3.3.4.1 Question forms

These forms will be used to answer a system request (see 3.3.1.15 and 3.3.1.14).

Type "YN"

System Request payload example:

- Description: "Do you want to proceed anyway?"
- OperationID: systemRequest
- Parameters: [unique requestID], YN

Users response will be "Yes" or "No".

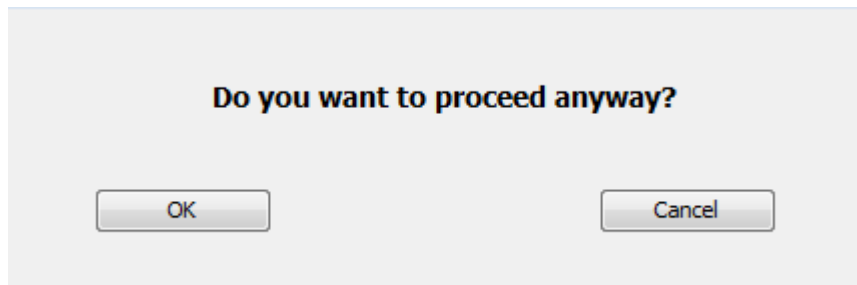


Figure 10 - System request - Type YN

Possible usage. A WP changes its own configuration and the whole system needs to be restarted. It can fire a SystemRequest of this type to ask the user if he/she agrees.

Type "Q"

System Request payload example:

- Description: "Input your code"
- OperationID: systemRequest
- Parameters: [unique requestID], Q

User response will be what typed in the text box.

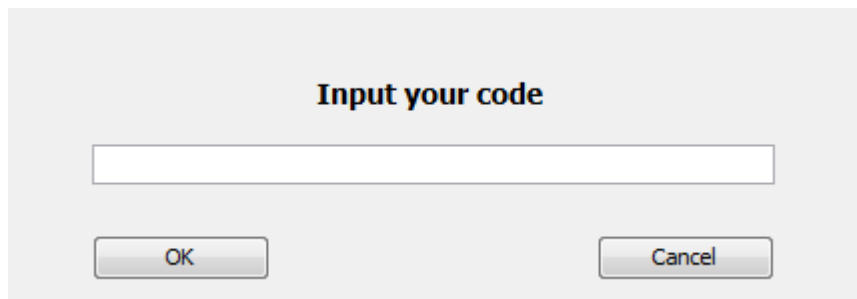


Figure 11 - System Request - Type Q

Possible usage. A WP needs a new parameter value at run time (for instance a cost function parameter). It can fire a SystemRequest of this type to ask user the new value.

Type "L"

System Request payload example:

- Description: "Choose one or more trains from the list"
- OperationID: systemRequest
- Parameters: [unique requestID], L

User response will be one or more items in the list.

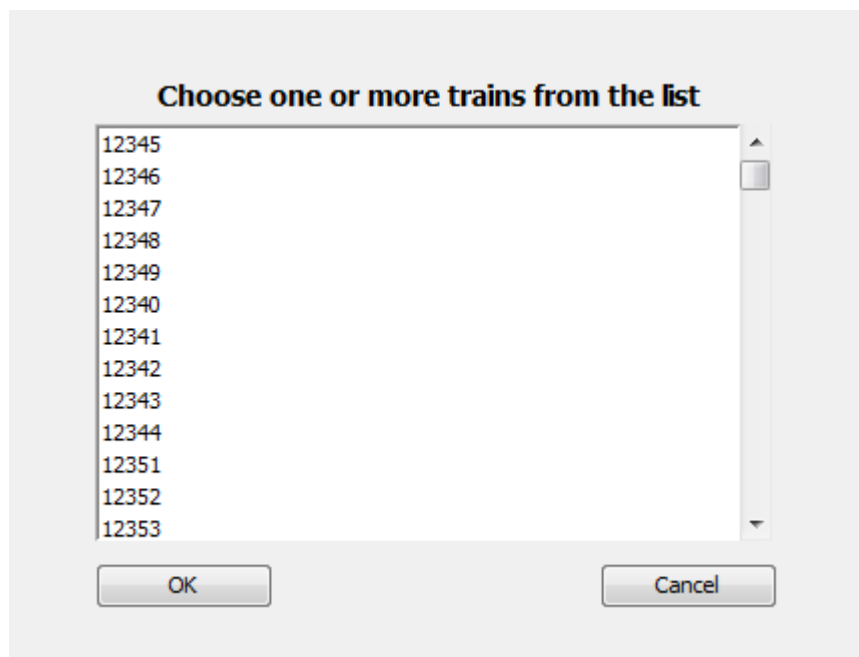


Figure 12 - System Request - Type L

Possible usage. A WP needs a subset of trains chosen from a bigger set. It can send the full set with a SystemRequest of this type and receive the required subset after the user choice.

3.3.4.2 Main window

The main window of the demonstrator is made up of the following part:

- 1) A menu bar with the following items:
 - a. Exit
 - b. System Boot (to enter the system boot parameters)
 - c. Cost Function (to enter the cost function parameters)
 - d. System Outcomes (to show the system outcomes)

If it is necessary, a WP related menu item can be added to show forms specific to one or more work packages.

- 2) A system log text area. It can be accessed using the showData event (formID=mainForm, widgetID=logWidget, type=T, data=<custom data>. See 3.3.1.16)
- 3) A system control area. It is made up of:

- a. Start system button. It starts all the tasks in the system using the current boot configuration. If the system is already started, the button is disabled.
 - b. Stop system button. It kills all the task in the system using the current boot configuration. If the system is already stopped, the button is disabled (ShutdownSystem event).
 - c. Boot configuration. It shows the current boot configuration.
- 4) A simulator control panel. It shall allow to:
- a. Start simulator (StartSystemClock event)
 - b. Stop simulator (StopSystemClock event)
 - c. Pause simulator (PauseSystemClock event)
 - d. Speed up simulator (SpeedUpSystemClock event)
 - e. Speed down simulator (SpeedDownSystemClock event)
- 5) Scenario selection. Choosing a new scenario will stop and restart the system.
- 6) Simulator speed control (for fine tuning and speed indication)

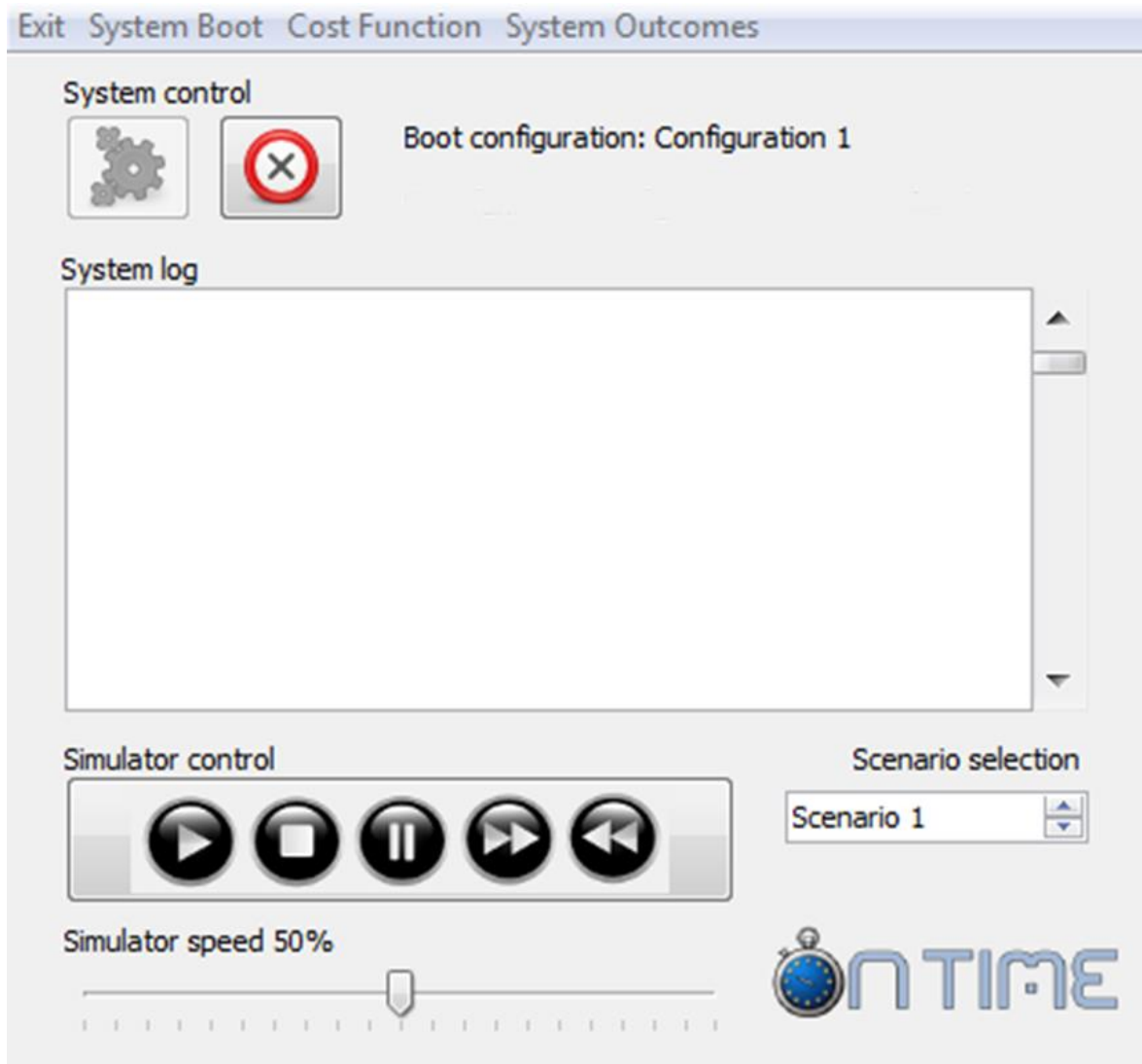


Figure 13 - Main demonstrator window

Shape and dimensions of this window will be adapted to the workstation monitors size and position.

3.3.4.3 System boot form

The purpose of this form is to provide a list of tasks and their related parameters to start the system. Every tasks/parameters configuration can be saved in an xml file.

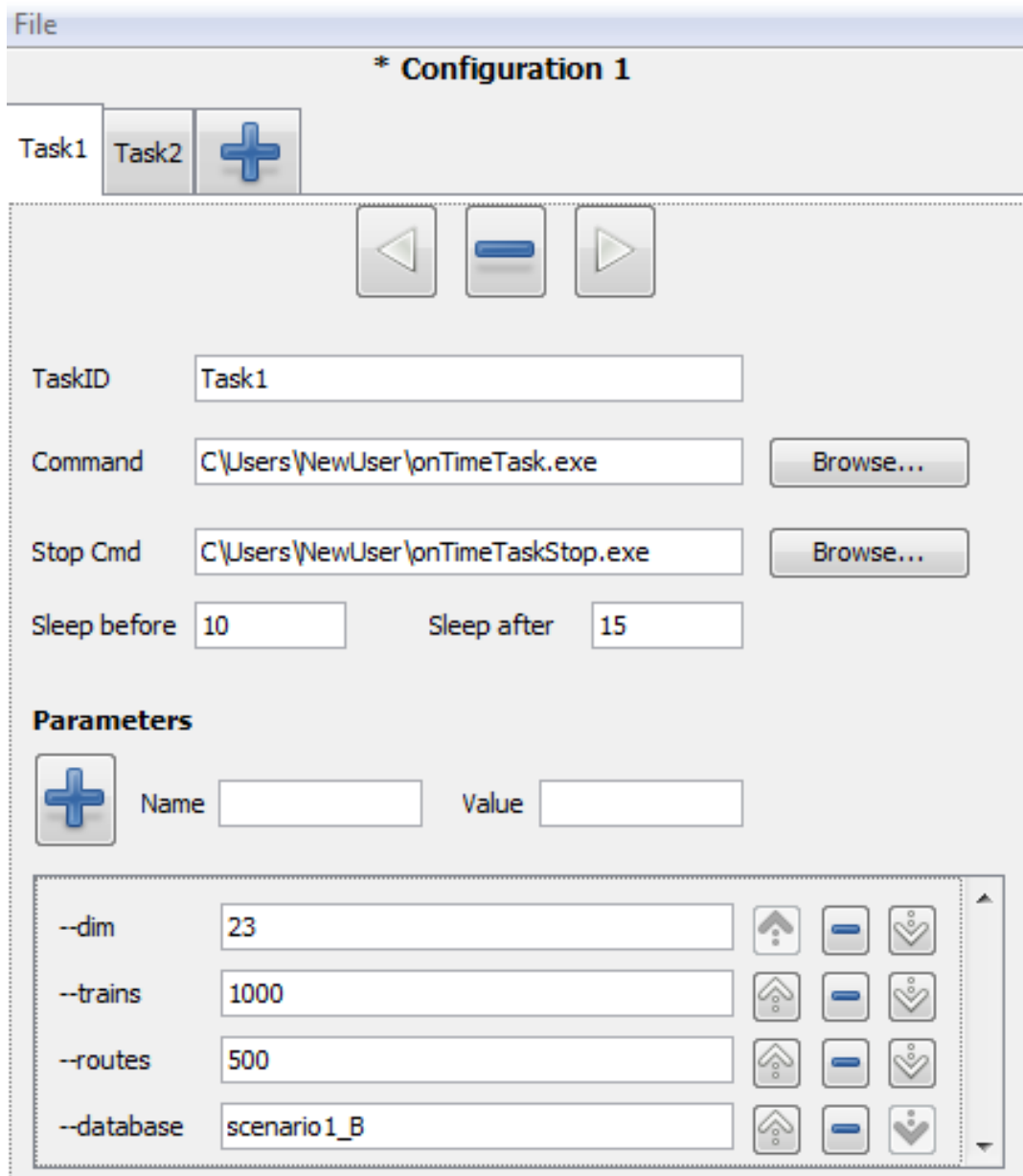


Figure 14 - System boot form

In the previous figure, it is possible to see how this window looks like.

The user can define all the system tasks, the order in which they must be launched and the parameters they need to start.

From top to bottom and from left to right, it is possible to identify several different areas.

- 1) Menu. The "File" item has the following sub-items:
 - a. Open... - To open a previously saved configuration.
 - b. Save - To save the current configuration with the same name.
 - c. Save as ... - To save the current configuration with a new name.

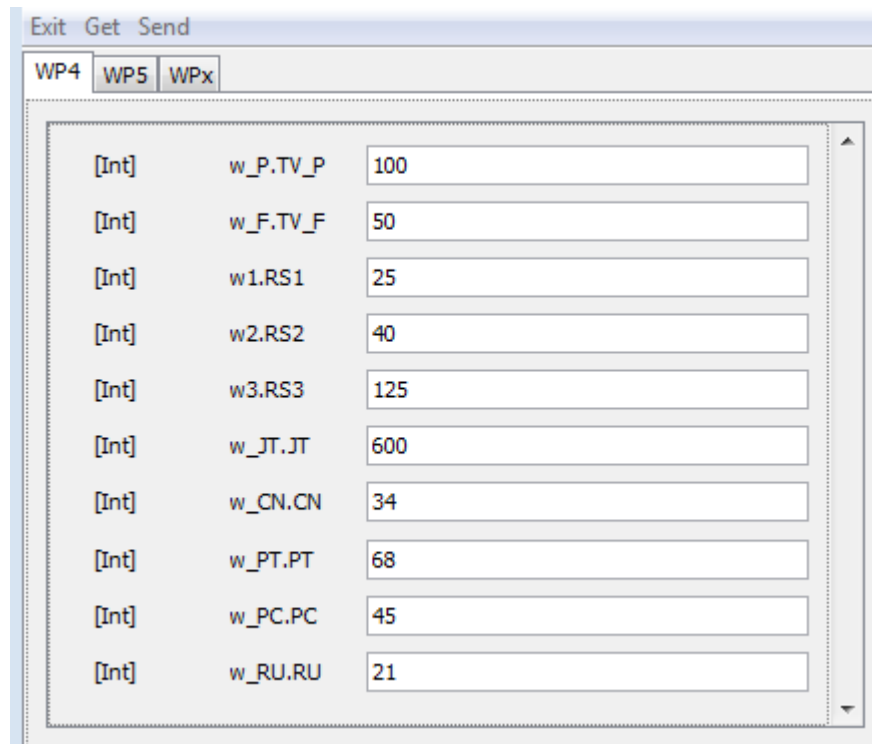
- d. Clone – To duplicate (copy) the current configuration in a new one.
- e. Close – To close the window.
- 2) Configuration name. In the example the name is "Configuration 1". The star sign (*) means that the current configuration has changed and not yet saved.
- 3) Task tabs. In the example, there are two tasks already configured (Task1 and Task2). The tab with the plus sign (+) shall be used to add a new task into the configuration.
- 4) Change position and cancel buttons. To change the task position (the launching order) or cancel the task itself. Task tabs shall change accordingly.
- 5) TaskID. To input the ID of the task. These ID shall be used to identify tasks when required by the previously described events.
- 6) Command. The command used to launch the task. The user can find the right path using the "Browse..." button.
- 7) Stop Cmd. The command used to stop the task (if it exists, otherwise a standard "kill" procedure shall be performed). The user can find the right path using the "Browse ..." button.
- 8) Sleep before. How many seconds HMI will wait before launching the task.
- 9) Sleep after. How many seconds HMI will wait after launching the task.
- 10) Add parameter (+ button). To add a new launch parameter.
- 11) Name. The name of the new parameter.
- 12) Value. The value of the new parameter.
- 13) Parameters list. An ordered list of the parameters used to launch the task. The user can:
 - a. Change the parameter order using the up and down buttons.
 - b. Cancel the parameter using the "minus" (-) button.

The task in the previous example will be launched using the following command:

```
C:\Users\NewUser\onTimeTask.exe -dm 23 -trains 1000 -routes 500 -database scenario1_B
```

3.3.4.4 Cost function parameters form

The purpose of this form is providing new parameters values (weights) for the cost functions (see "D1.2 - A framework for developing an objective function for evaluating work package solutions" about cost functions). Every WP cost function parameter can be get using the GetLastParameters message (3.3.2.4) from DataProvider, changed by the user and sent back to the WP using the CFPParametersChangeEvent event (3.3.2.1). See 3.3.2 (cost function parameter protocol) for more details.



Data Type	Parameter Name	Value
[Int]	w_P.TV_P	100
[Int]	w_F.TV_F	50
[Int]	w1.RS1	25
[Int]	w2.RS2	40
[Int]	w3.RS3	125
[Int]	w_JT.JT	600
[Int]	w_CN.CN	34
[Int]	w_PT.PT	68
[Int]	w_PC.PC	45
[Int]	w_RU.RU	21

Figure 15 - Cost function parameters form

In the previous figure, it is possible to see how this window looks like.

The user can define the values of all the cost function parameters divided by work package. Parameters order is not important in this case.

From top to bottom and from left to right, it is possible to identify several different areas.

- 1) "Exit" menu item. To close the window.
- 2) "Get" menu item. To get from DataProvider the current cost function parameters list using the GetLastParameters message. This operation is done for the currently selected work package (see point 4).
- 3) "Send" menu item. To send the current cost function parameters list using the CFPParametersChangeEvent. This operation is done for the currently selected work package (see point 4).
- 4) WP selector. In the example picture it is supposed that WP4, WP5 and WPx only need to deal with cost function parameters. Changing the selected WP (say from WPa to WPb), the demonstrator shall:
 - a. Send a GetLastParameters for WPb, if WPa cost function parameters haven't changed.
 - b. Ask the user a confirmation, if WPa cost function parameters have changed.
 - i. If the user continues, all WPa changes will be lost.

- ii. If the user doesn't continue, nothing happens and the user can use the "send" menu item to save the current changes before switching WP.
- 5) Parameters list. Type, name and value of parameters are shown in the list. The user can change any parameter value just rewriting it.

The cost function parameters (weights) used in the example are taken from the above mentioned D1.2 document.

3.3.4.5 System Outcomes

System outcomes can differ in terms of types and visualization.

Three primitive data types have been defined in 3.3.1.16 (showData): Text (T), Number (N), List (L [Text+Number]). Others and more structured types can be added to fulfill all system requests.

The visualization is another issue to face. The same number, for instance, can be represented inside a textbox, a spinner or using a progress bar or a pie diagram.

For the sake of generality and not knowing exactly the boundaries of the system, in this paragraph is proposed a solution that can deal with the most of situations.

Case 1 – Text

A text can be displayed in:

- 1) a label;
- 2) a text box;
- 3) a text area;
- 4) a banner.

The visualization depends on the widget which the text is displayed in.

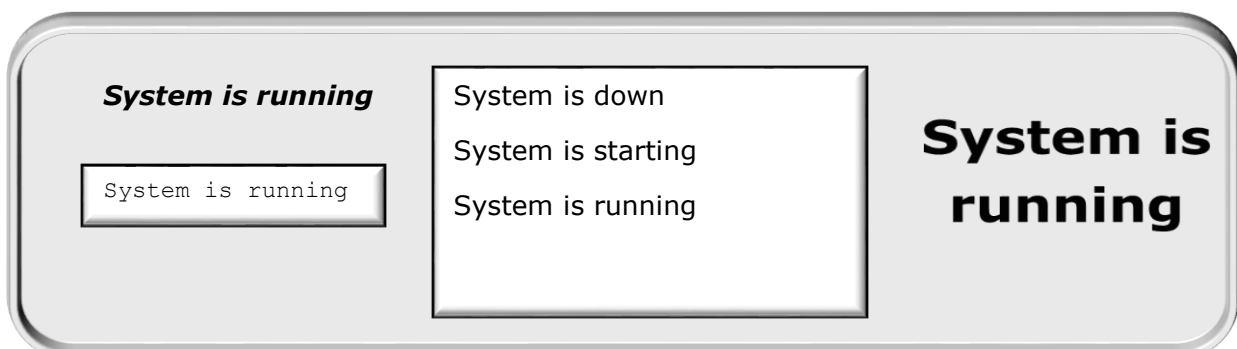


Figure 16 - Displaying a text

Considering only the "parameters part" of the showData operation described in 3.3.1.16, a task can send the following data to HMI to get the output depicted in the previous figure.

- textFormID, widgetLabel, T, "System is running"
- textFormID, widgetTextBox, T, "System is running"
- textFormID, widgetTextArea, T, "System is running"
- textFormID, widgetBanner, T, "System is running"

where textFormID is the ID of the previous form; widgetLabel, widgetTextBox, widgetTextArea, widgetBanner are the names of the widgets from top to bottom and left to right.

Note that in the widgetTextBox the text is added to the previous one. A maximum number of rows can be defined.

Case 2 – Number

A number can be displayed in:

- 1) a label;
- 2) a spinner;
- 3) a progress bar (if ≥ 0 and ≤ 1);
- 4) a pie diagram (if ≥ 0 and ≤ 1);
- 5) ... and more ...

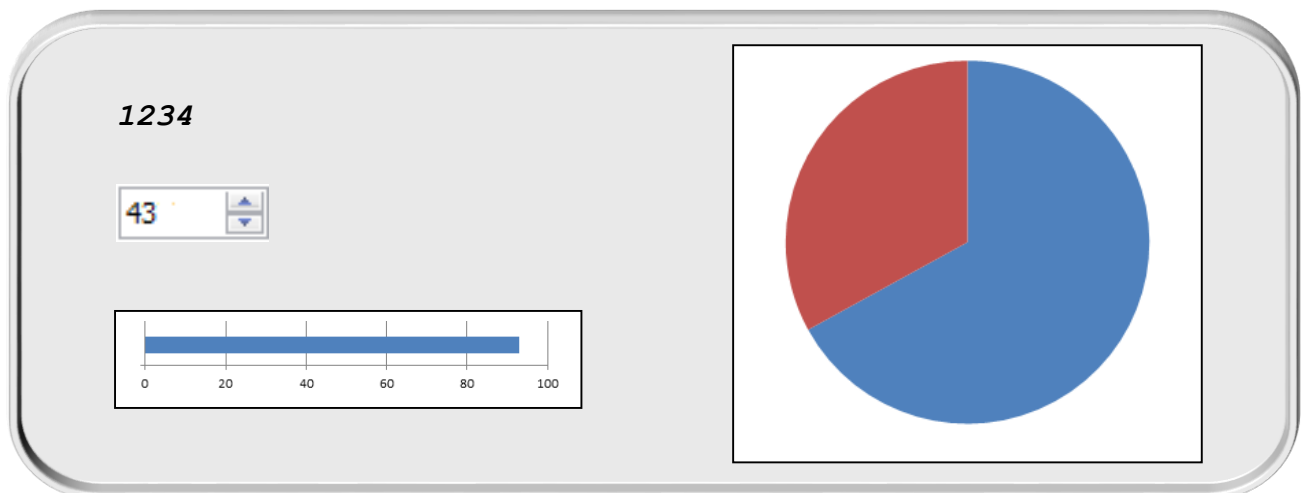


Figure 17 - Displaying a number

A task can send to HMI the following data to get the output depicted in the previous figure.

- numberFormID, widgetLabel, N, 1234
- numberFormID, widgetSpinner, N, 43
- numberFormID, widgetProgressBar, N, 0.93
- numberFormID, widgetPieDiagram, N, 0.67

where numberFormID is the ID of the previous form; widgetLabel, widgetSpinner, widgetProgressBar, widgetPieDiagram are the names of the widgets from top to bottom and left to right.

Case 3 – List

A list can be displayed in:

- 1) columns;
- 2) 3DLines;
- 3) 3DPie;
- 4) cylinders;
- 5) ... and more ...

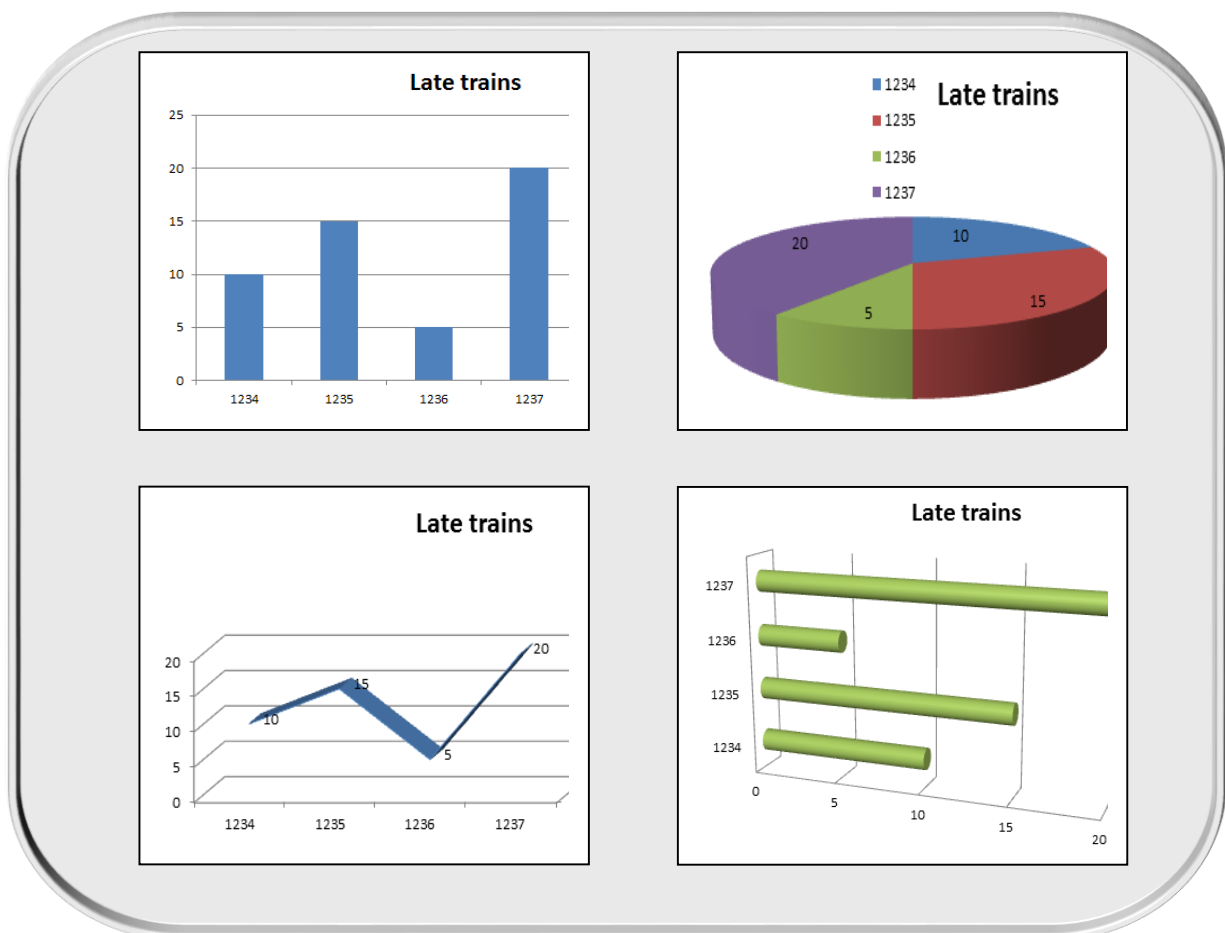


Figure 18 - Displaying a list

A task can send to HMI the following data to get the output depicted in the previous figure.

- listFormID, widgetColumn, L, 1234=10:1235=15:1236=5:1237=20
- listFormID, widget3DLine, L, 1234=10:1235=15:1236=5:1237=20
- listFormID, widget3DPie, L, 1234=10:1235=15:1236=5:1237=20
- listFormID, widgetCylinder, L, 1234=10:1235=15:1236=5:1237=20

where listFormID is the ID of the previous form; widgetColumn, widget3DLine, widget3DPie, widgetCylinder are the names of the widgets from top to bottom and left to right.

3.3.4.6 WP related forms

If needed, one or more WP specific forms can be added to HMI. Shape and contents of these forms must be agreed between WPs leaders and WP8 leader.

3.3.4.7 Conclusions

HMI is a fully configurable and general purpose graphical interface. Its architecture allows, however, to integrate more specific objects to deal with specific needs. The communication protocol, as well, is open enough to add easily new functions and operations.

4 REFERENCES

- 1) Clive Roberts, Lei Chen, Menglei Lu, Linsha Dai, Chris Bouch, Gemma Nicholson, Felix Schmid, *D1.2 A framework for developing an objective function for evaluating work package solutions (Cost function)*, 2012 Birmingham
- 2) Matteo Anelli, Bruno Ambrogio, Daniele Carcasole, Thomas Albrecht, John Easton, *D7.2 Architecture specification and integration requirements*, 2012 Rome