ICT-2011.8
GET Service Project
2012-318275

Deliverable D7.3

Prototype of reconfigurable transportation orchestration engine

24 July 2015
Public Document

GET SERVICE

Project acronym:	GET Service
Project full title:	Service Platform for Green European Transportation
Work package:	7
Document number:	D7.3
Document title:	Prototype of reconfigurable transportation orchestration engine
Version:	1
Delivery date:	31 July 2015
Actual publication date:	31 July 2015
Dissemination level:	Public
Nature:	Prototype
Editor(s) / lead beneficiary:	Remco Dijkman
Authors(s):	Shaya Pourmirza Remco Dijkman
Reviewer(s):	Martin Hrušovský Emrah Demir

Executive Summary

This deliverable documents the prototype of the reconfigurable transportation orchestration engine that is developed in the GET Service project.

When a transportation plan requires a re-planning, the orchestration engine that controls the transportation process needs to be reconfigured as well. The main challenge is to determine how the new requirements affect the running of the transportation process. For example, some transportation tasks that are being executed may have to be cancelled and some tasks that have been executed may have to be rolled back. Therefore, a reconfigurable technique is required to enable the migration of the information and states of a transportation process that cannot finish successfully to a new transportation process that is composed when re-planning a transportation order.

The main contribution of this deliverable, namely D7.3, is to provide the implementation details of the reconfiguration technique that has been implemented as a part of the orchestration engine, which has been selected for the GET Service project.

The developed reconfiguration technique consists of two distinct phases: (i) pre-migration and (ii) migration. The first phase is mainly responsible for storing the information and states of the old transportation process and for creating a new transportation process based on the updated (re-planned) transportation plan. The second phase is the actual migration procedure that transfers the stored information and states to the updated transportation process.

The applicability of the developed reconfiguration technique is shown by applying it to one of the GET Service project demo scenarios.

Contents

List of Figures

1 Introduction

This deliverable documents the prototype of the reconfigurable transportation orchestration engine that is developed in the GET Service project. This section provides the background to the deliverable, by presenting the goal of the project as a whole, the goal of the work package of which the deliverable is a part, and the goal of the deliverable itself. Finally, it presents the structure of the remainder of the deliverable.

1.1 Project Goal

The GET Service project develops a platform that enables transportation planners and drivers with the means to plan and execute transportation routes more efficiently and to respond quickly to unexpected events during transportation. To this end, it connects to existing transportation management systems (TMS) and improves on their performance by enabling sharing of selected information between transportation partners, logistics service providers and authorities. In particular, the GET Service platform consists of components that:

(i) enable aggregation of information from the raw data that is shared between partners and transportation information providers;
(ii) facilitate planning and re-planning of transportation based on that real-time information; and
(iii) facilitate real-time monitoring and control of transportation, as it is being carried out by own resources and partner resources.

By providing these functionalities, the GET Service platform aims to reduce the number of empty miles that is driven (by an empty truck), improve the modal split, and reduce transportation times and slacks, as well as response times to unexpected events during the transportation. Thus, it reduces carbon dioxide equivalent (CO_2e) emissions and improves the transportation efficiency.

1.2 Work Package Goal

The goal of Work Package 7 (WP7) is to develop support for monitoring and executing a detailed transportation plan, also called a 'transportation process'. In particular, it supports the automated adaptation of transportation process when the corresponding transportation plan changes.

Figure 1 illustrates the relation between a transportation plan and a transportation process with an example. In the example, a planner of a transportation company must plan the bulk transportation of several electronics components on behalf of one of the company's clients, from different locations in Europe to the Far East. The planner enters the pick-up locations of the orders into the system, as well as the times at which the components are estimated to be available for pick-up, the time at which the components should be in the factory in the Far East and several other criteria for planning, such as the desire to minimize CO_2e emissions. The planner achieves this through the graphical user interface (GUI) that is provided by the transportation planning service. The transportation planning service computes an optimal route, which may look like the one shown in Figure 1.i. The planner is informed of this option through the GUI. Alternatively, the planner may be provided with multiple alternatives from which he can select one, or the planner can manually adapt the planned route via the graphical user interface. After the planner selects a route, a service composition is created that will control the route. This composition is created based on the planned route and individual control services that are associated with the different segments of the route. This composition may look like the one shown in Figure 1-ii.

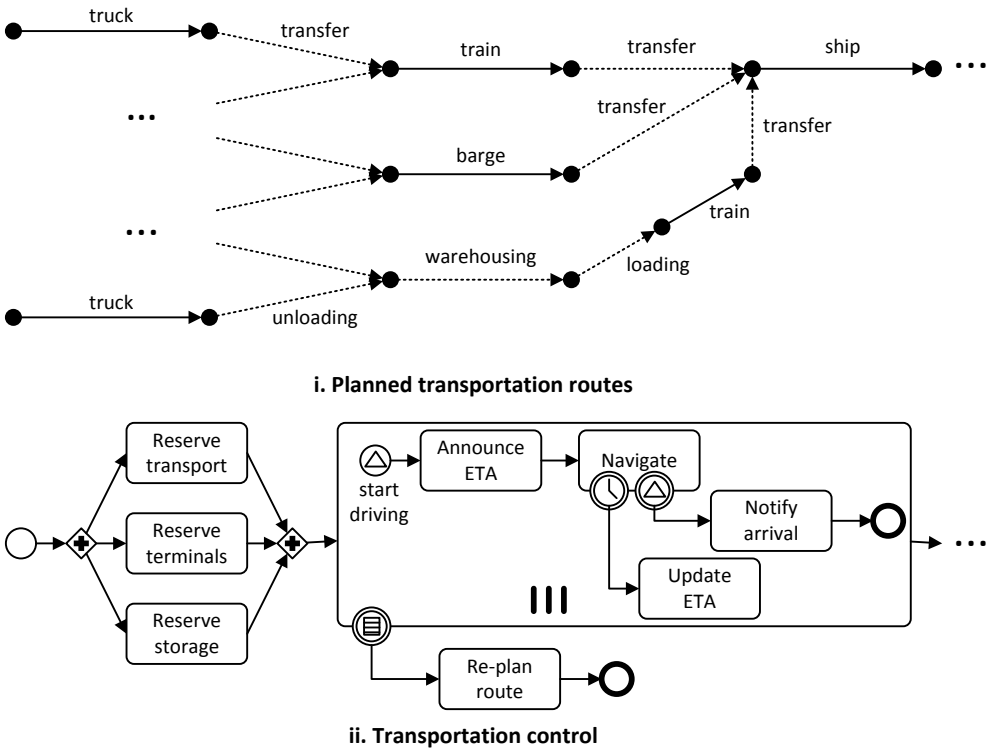

Figure 1 - Example planning and control output

1.3 Deliverable Goal

One of the explicit aims of the GET Service platform is to be prepared for unforeseen circumstances that may change the transportation plan, such as unexpected traffic congestions, delays of onward transportation and changes in the plans of the customers. Consequently, when the transportation plan changes, the orchestration that controls the transportation plan also changes. The main challenge, which must be addressed here, is to determine how this affects the running process instances: tasks that are being executed may have to be cancelled, tasks that have been executed may have to be rolled back, and the control structure must be reconfigured in such a way that it affects the tasks that have been executed and that are being executed as little as possible, while creating a new control structure that can replace the previous one successfully.

A design for a technique that supports this was delivered in Deliverable 7.2 (Pourmirza, et al., 2014), where we presented an algorithm that facilitates the change in the detailed transportation plan, while the abstract-view of the plan remains the same before and after the change. The details and formal specifications of the technique have been elaborated in (Pourmirza, et al., 2014). This document presents the implementation of the algorithms that are presented in these documents and implemented as part of the orchestration engine that was selected for the GET Service project (cf. (Pourmirza & Dijkman, 2014)).

1.4 Deliverable Structure

The remainder of this deliverable is organized as follows. Section 2 provides details of developed reconfiguration technique. Section 3 demonstrates the applicability of this technique using one of the demo scenarios in the context of the GET Service platform. Finally, Section 4 presents the conclusions.

Note that, most of the figures in this document include an online version with a higher-resolution quality to increase the readability. One can click a figure or its caption in order to see its online version.

2 Reconfiguration Technique

As discussed in Deliverable 7.2 (Pourmirza, et al., 2014), during the execution of a transportation, a transport plan might need to be re-planned. When this happens, the transport process, which has been composed based on initial transport plan, must be replaced by a new transport process, which is based on the new transport plan. This must be done at run-time. Preferably, the execution of the new transport process begins where the initial transport process terminated. Thus, a technique is required to enable such reconfigurations in the orchestration engine of the GET Controller[1] at run-time.

The main user interface of the GET Controller is illustrated in Figure 2. At the left side, it shows the running transportation cases (i.e., *case handler*), and at the right side it shows the tasks that belong to the selected transportation case (i.e., *task handler*). The case that is currently being inspected and the tasks that are currently executing are marked. Lastly, at the bottom, it displays a transportation process of the selected case and the red borders around some tasks elucidate the active (i.e., executing) tasks therein.

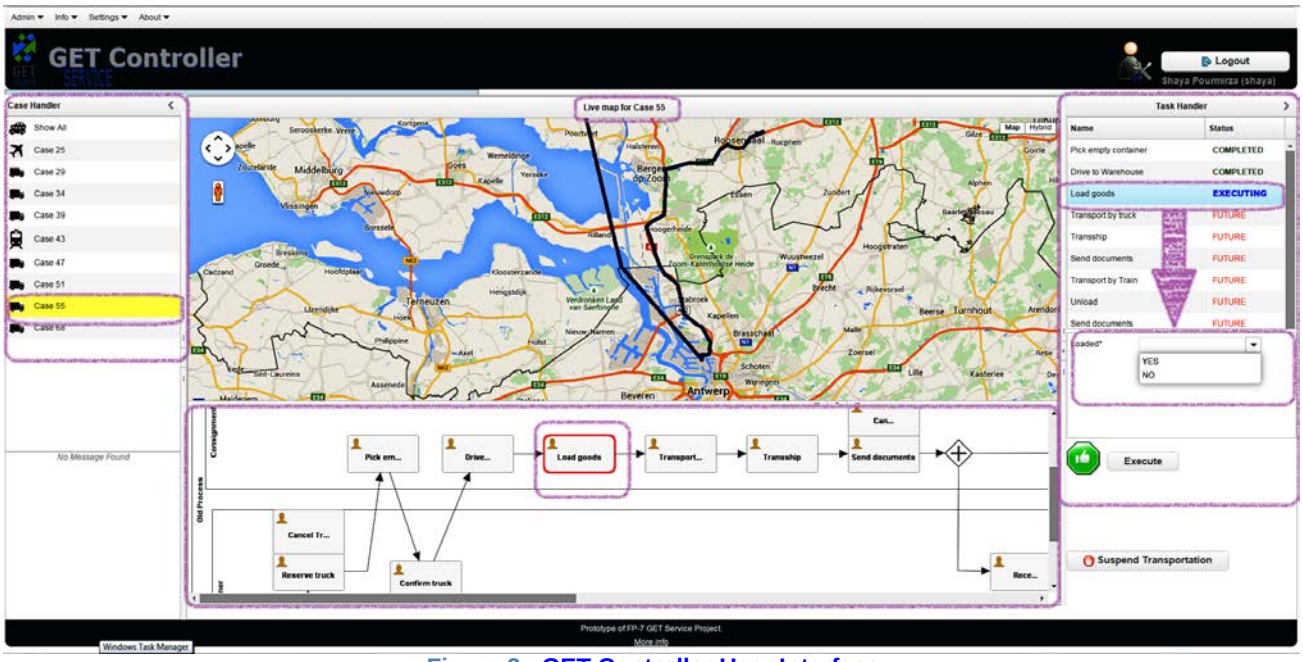

Figure 2 - GET Controller User Interface

This section provides a technical description of the developed reconfiguration technique. This technique consists of two phases: (i) pre-migration phase, and (ii) migration phase. The first phase mainly includes the re-planning of the transportation plan (Arikan, et al., 2014) by a planner (Section 2.1) and the second phase mainly includes the actual algorithm that runs at the backend (Section 2.2).

2.1 Pre-migration Phase

In this section, we describe the first phase of the employed technique in more detail. The goal of this phase is to suspend and re-plan a transportation due to unexpected events. The outputs of this phase includes:

(i) a detailed history of a suspended transportation process; and
(ii) an updated transportation process based on the updated transportation plan.

[1] http://is.ieis.tue.nl/research/getservice/

Figure 3 illustrates the sequence diagram for the pre-migration phase. This phase triggers when a transportation cannot be finished successfully. When that happens, the planner needs to *suspend* the current transportation and *update the transportation plan*, for example by choosing different transportation resources or a different route. After updating the transportation plan, a new *transportation process* must be composed based on the updated plan, which can be handled by the composition engine (Botezatu & Völzer, 2015). Subsequently, the planner needs to deploy the updated transportation process instead of the suspended one. Finally, the controller must ensure that the transportation process that was associated with the initial transportation plan is properly migrated to the updated transportation process.

The pre-migration phase is explained in more detail in the remainder of this section.

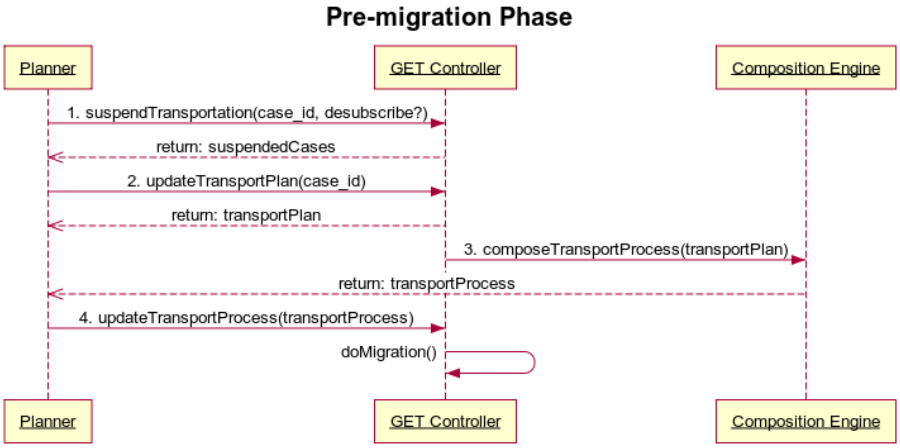

Figure 3 - Sequence Diagram for the Pre-migration Phase

2.1.1 Transportation Suspension Request

The first arrow in the sequence diagram in Figure 3 is a suspension request, which a planner uses to suspend a transport process. The planner can also request the controller to unsubscribe from all the registered notifications about that specific transportation. However, in some physical tasks such as 'drive' task, there might be a short delay between suspension (in the digital world) and the effect of suspension (in the physical world – practice), and therefore, a planner may want to allow the orchestration engine to consider the events that arrive after the suspension.

When receiving the suspension request, the orchestration engine stores all the information about the suspended transportation. This information can be classified into two groups:

(i) *control-flow information*, which contains the state of *past*, *executing* and *future* tasks, and
(ii) *data-flow information*, which contains the data-fields and forms that were filled in during the transportation.

Afterwards, the suspended transport process will be removed from the list of active transport processes, and consequently, it will be removed from the case handler of the involved users in the transportation including its planner.
The GET Controller supports this functionality by providing a button for the planner to suspend a transportation at any time. When a planner clicks this button, a new window pops up, which consists of a text field that is used to write a reason of suspension and a button to suspend the transportation. By filling in the reason and clicking on the 'Suspend Transportation' button, the selected transportation case will be wiped out from the 'Case Handler' of the involved users. Figure 4 illustrates the transportation suspension request in the user interface.

Figure 4 - GET Controller GUI: Suspend Transportation

2.1.2 Update Transportation Plan Request

The second arrow in the sequence diagram in Figure 3 is a re-planning request. During re-planning, the planning component of the GET is called to request different possible alternative plans based on the transport order. After receiving the alternative plans, the planner can update the plan of suspended transportation by selecting a new transportation plan.

Figure 5 - GET Controller GUI: Update Transport Plan (Step 1)

As shown in Figure 5, the GET Service controller supports this, by providing a menu item ‘update transport order’. When the planner clicks on this menu item, a new window pops up, which consists of a list of suspended transportation orders with reason(s) in which they are suspended, a button to

update a transport plan, a button to update a transport process and a button to remove a transportation case from the engine. The first two buttons are used for online re-planning purposes and the third one is used to remove the suspended transportation completely without online re-planning. Since one transport plan can contain multiple transportation cases, it is possible to select multiple cases for re-planning. By clicking on the 'Update Transport Plan', the GET Controller shows different alternative transportation plans and a planner can select one of them (see Figure 6).

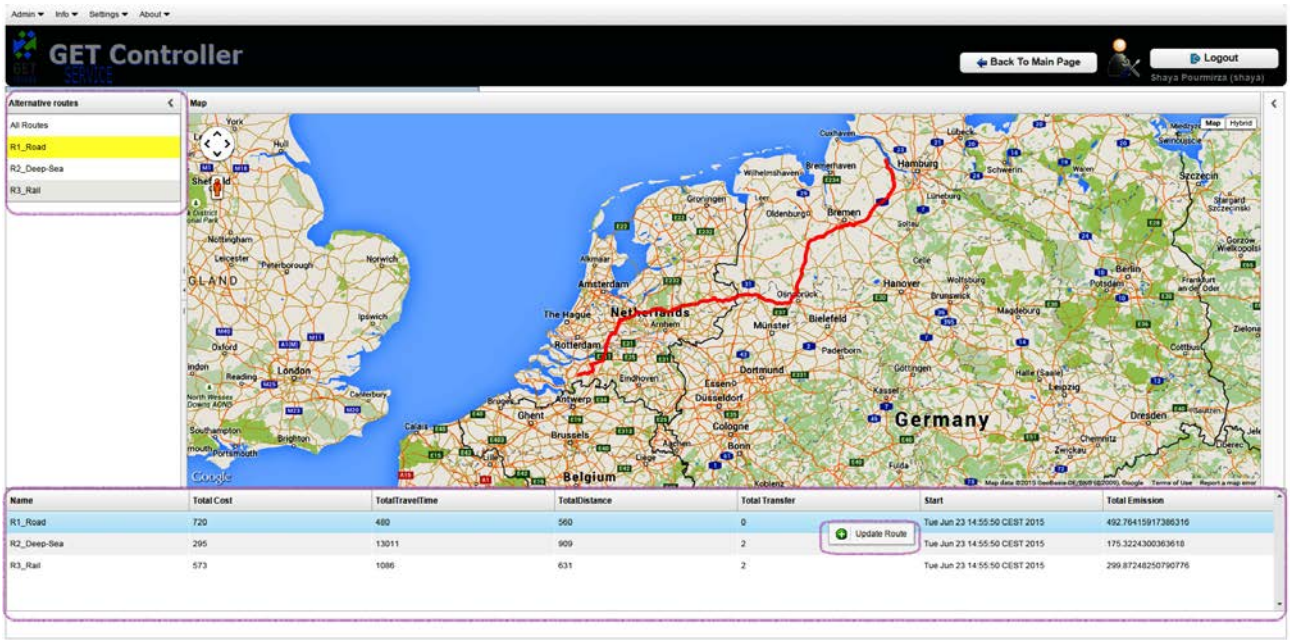

Figure 6 - GET Controller GUI: Update Transport Plan (Step 2)

2.1.3 Compose Transportation Process

The third arrow in the sequence diagram in Figure 3 is a composition request. Having selected an updated transportation plan, the orchestration engine informs the composition engine that is needs to compose a new transportation process. To this end, the GET Controller provides a service that can be used by the composition engine. This service contains an updated transportation plan in JSON[2] format.

When receiving the composition request, the composition engine composes a new transportation process that needs to be deployed by a planner using the 'Update Transport Process' button in the 'Update Case' window (see Figure 5).

2.2 Migration Phase

In this section, we describe the second phase of our technique in further details. The goal of this phase is to resume the transportation activities based on the updated transportation process that is deployed in the first phase. The challenge here is to migrate the stored information from the suspended transportation process (cf. Section 2.1.1) to the updated transportation process (cf. Section 2.1.3), in order to resume the transportation from *the closest possible state* to where the previous transportation was suspended.

[2] http://json.org/

To this end, we have annotated the transportation process with an extra element called *re-planning annotation* (Section 2.2.1). Based on this element we have defined a new *migration algorithm* (Section 2.2) that can be employed to evaluate the current state of an updated transportation with regard to the state in which previous transportation was suspended.

2.2.1 Re-planning Annotation

We have further extended BPMN-T modeling language (Botezatu & Völzer, 2014), which is used to model the transportation processes by introducing a new element. Each task in a transportation process can be annotated by a re-planning annotation element. This element can be used to find the closest possible state in an updated transportation based on the last known state of a suspended transportation in order to enable the migration procedure. This procedure mainly consists of:

(i) backward recovery (i.e., rolling back and cancelling tasks), and
(ii) forward recovery (i.e., performing compensatory tasks).

A task in a transportation process can be one of these below:

(i) user task, which refers to an activity that needs to be performed by a specific user,
(ii) compensatory task, which refers to an activity that needs to be performed when the task for which it compensates has been executing and re-planning occurs (e.g., cancelling a reservation is compensation for making a reservation), or
(iii) confirmation task, which is a milestone in a transportation that can be used in case of re-planning to assess whether a compensatory task can still be performed or not (e.g., after confirming a reservation it is not possible to cancel it anymore).

Transportation processes contain physical tasks that cannot be rolled back at all times. For example, considering a scenario in which some goods will be transported by a train, but due to an unexpected event, a planner might decide to change the plan. If the reserved train does not start its journey, the planner will be able to cancel the train reservation. However, if the train is already halfway through its journey, it will not be possible to cancel the train reservation; nevertheless, a potential solution might be to unload containers at the closest train station.

Therefore, the re-planning annotation can be also used to assess whether one task can be rolled back to its previous state or not. The re-planning annotation consists of following sub elements:

(i) a rollback flag, which determines whether the task can be reverted,
(ii) a confirmation flag, which determines whether the task is a confirmation task,
(iii) a compensatory task, which points to the task for which it is compensating,
(iv) a confirmation task, which points to the task for which it is a confirmation,
(v) a source, which is the location of origin of the transportation represented by a task, and
(vi) a destination, which is the destination of the transportation represented by a task.

The schema of the re-planning annotation is encoded in Figure 7.

```
<xs:element name="replanningAnnotation" minOccurs="0" maxOccurs="1">
   <xs:complexType>
      <xs:sequence>
         <xs:element type="xs:boolean" name="isRollbackable"/>
         <xs:element type="xs:boolean" name="isConfirmationTask"/>
         <xs:element type="xs:string"  name="compensatoryTask" minOccurs="0" maxOccurs="1"/>
         <xs:element type="xs:string"  name="confirmationTask" minOccurs="0" maxOccurs="1"/>
         <xs:element type="xs:string"  name="source"  minOccurs="0 maxOccurs="1"/>
         <xs:element type="xs:string"  name="destination" minOccurs="0 maxOccurs="1"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Figure 7 - XML Schema of the Re-planning Annotation Element

2.2.2 Migration Procedure

This section presents the steps that need to be taken when applying the migration algorithm. Figure 8 provides a high-level view of this algorithm, which is explained in more detail in the remainder of this section.

Figure 8 - Migration Steps

The first step of the migration algorithm is retrieving the *transportation trace* of the suspended process. In this step, the orchestration engine retrieves the stored data from a suspended transportation (see Section 2.1.1). Particularly concerning the control-flow information, it can retrieve a trace, which is the list of tasks that have been executed for the suspended transportation in chronological order. Concerning the data-flow information, it can retrieve the form-variable information that had been inserted by users, or generated automatically. The output of this step is a list of transportation tasks that were performed before including their form-variables information.

The second step of the migration algorithm is calculating the active compensatory tasks from the suspended process. In this step, the orchestration engine assesses the transactional properties (Grefen, et al., 2001) of the tasks in the transportation trace, which has been retrieved in the first step. To this end, it parses the re-planning annotation elements of the tasks in order to assign rollback flags and compensatory/confirmation tasks if required. As mentioned before, a task can be rolled back if its stored data can be reverted. Therefore, physical tasks such as 'drive' cannot be rolled back. In addition, a task may be annotated with a compensatory task (needs to be executed in case of re-planning) in order to eliminate its effect. Finally, a task can be a confirmation tasks if it affects the ability to execute a compensatory task. Figure 9 illustrates the triangular model for the tasks in the transportation process. We define following rule between a user task with its compensatory task and its confirmation task:

> ***Rule1****: When a user task becomes completed its compensatory task (if it exists) becomes active until its confirmation task (if it exists) becomes enabled.*

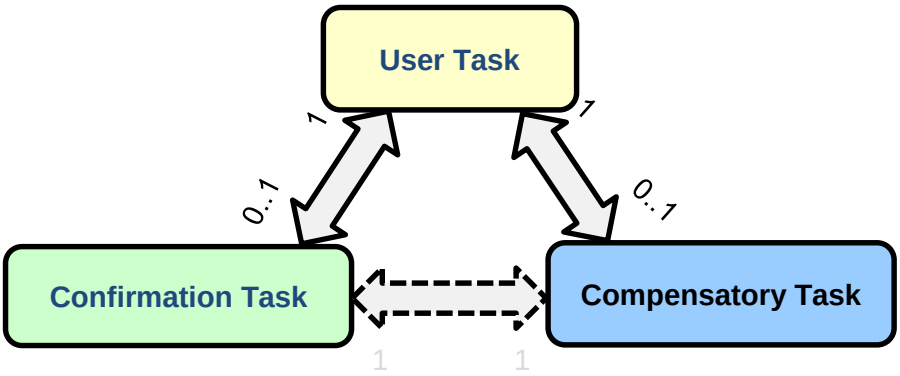

Figure 9 - Triangular Model for the Tasks in the Transportation Process

The orchestration engine computes a list of active compensatory tasks (i.e., compensatory tasks that need to be performed) by applying the *Rule1*. This list is the output of the second step of the algorithm.

The third step of the migration algorithm is finding similar tasks (Dijkman, et al., 2011) between the suspended and updated transportation processes. For each task in a transportation trace, the orchestration engine searches for an analogous task in the updated transportation process. If all of the following rules are valid, two tasks are considered similar. If one of the following rules is not valid, they are not similar and no further investigation will be required.

The main criterion that needs to be assessed is *Task Names*.

> ***Rule2****: If a task in the suspended transportation process has the same task name as a task in the updated transportation process, it can be similar to that task.*

The second criterion that needs to be assessed is form variables. A task in a transportation process may contain multiple form variables. The orchestration engine can produce a schema based on the form variables that are associated with a task.

> ***Rule3****: If a task in the suspended transportation process has the same form variable schema as a task in the updated transportation process, it can be similar to that task.*

The third criterion that needs to be assessed is the source and destination locations of two tasks with the same name. These locations are retrieved from the re-planning annotation element. A task may or may not have a source and a destination.

> ***Rule4****: If a task in the suspended transportation has the same source and destination locations as a task in the updated transportation, it can be similar to that task.*

Based on these three criteria, the orchestration engine is able to find the similar tasks between the suspended and updated transportations. These tasks are the output of the third step of the algorithm.

The fourth step of the migration algorithm is re-executing similar tasks with the variables of the suspended transportation. In this step, the orchestration engine uses the output of the third step (i.e., similar tasks) in order to re-execute them in the updated process. If a form-variable is required when executing a task, this value can be retrieved from the transportation trace (first step of the algorithm). The output of this step is a state in the updated process that is the closest possible state to where the previous transportation was suspended.

Finally, the last step of the migration algorithm is to attach compensatory tasks to the updated process. In this step, the orchestration engine uses the output of second phase (i.e., a list of active compensatory tasks) as an input. Then, it reverses the order of these tasks in the list and puts them in a sequence order for the execution. Finally, this sequence needs to be attached to the composed updated transportation process (see Section 2.1.3). For this purpose, the planner can choose between two strategies, namely *strict* and *flexible*. In the former case, this sequence is attached to the last same task from step 3, and the tasks in the sequence need to be performed first. In the latter strategy, this sequence is attached to the updated process in a completely different path, and therefore, tasks therein can be performed at any time without any restriction. The output of this step is the updated transportation process that contains active compensatory tasks.

Having taken these four steps, the orchestration engine can resume the execution of the updated transportation process that:

(i) is at the closest possible state to where the previous transportation was suspended (third step), and
(ii) includes active compensatory tasks, which are currently become part of its control-flow.

3 Scenario Example

In this section, we demonstrate the applicability of the developed reconfiguration technique via the Export scenario, which is one of the designed demo scenarios in the context of the GET Service project (Treitl, et al., 2013). In this scenario, a container needs to be transported from a warehouse near Venlo to Rotterdam harbor. The planner of this transportation has designed a multi-modal plan, which consists of:

(i) a truck journey to pick up the container and transport them to Eindhoven rail station, and
(ii) a train journey to transport the container from Eindhoven rail station toward the Rotterdam harbor.

The detailed transportation process of this transportation plan is shown in Figure 10, where the user tasks are shown with yellow boxes, compensatory tasks are shown with the blue boxes and confirmation tasks are shown with green boxes.

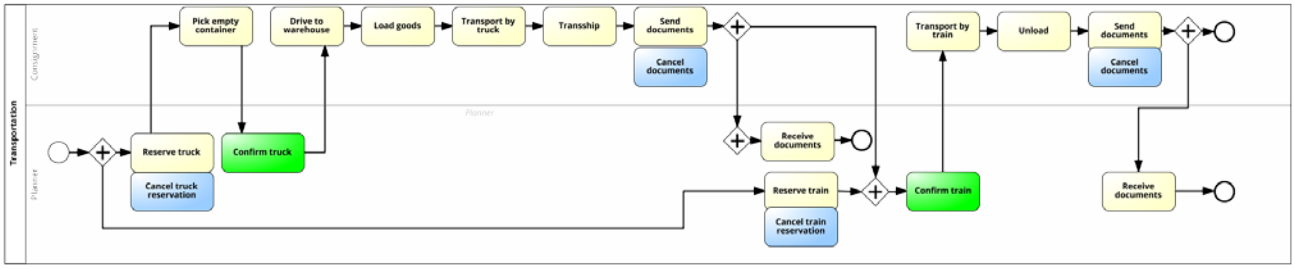

Figure 10 - Transportation Process for the Multi-modal Planning Scenario

Now, considering a situation in a transportation in which a planner has reserved a truck and a train, and the truck is driving from the warehouse toward Eindhoven rail station. Figure 11 depicts this situation in the transportation process that has been automatically generated by the GET Controller for the visualization purpose, and Figure 12 shows the corresponding task handler in the user interface.

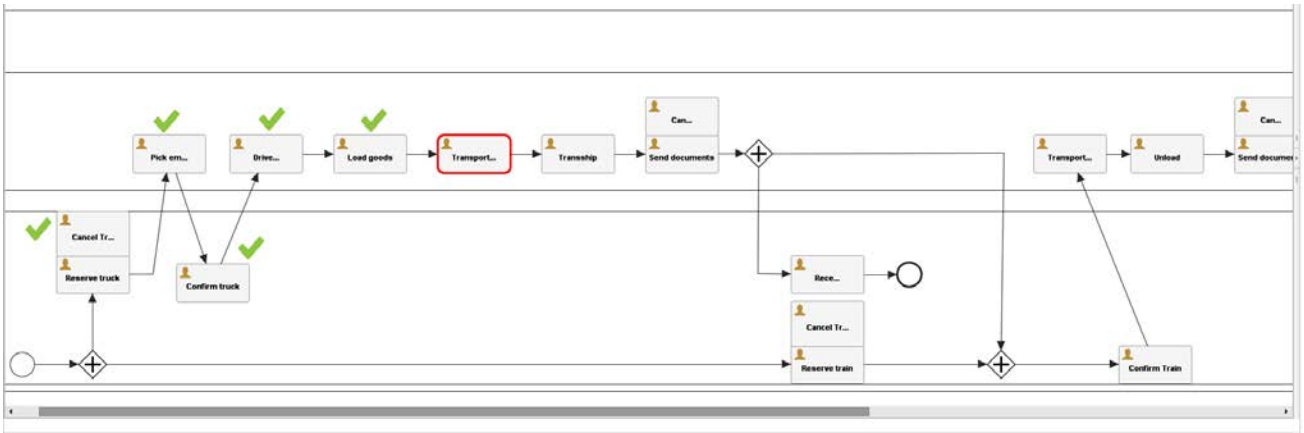

Figure 11 - Running Transportation Process – Before Re-Planning

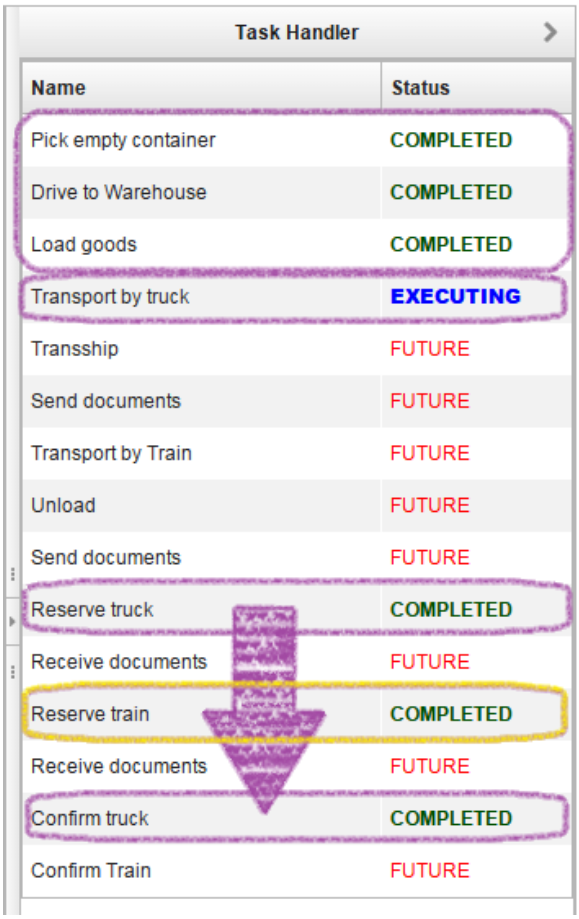

Figure 12 - Task Handler before Re-planning

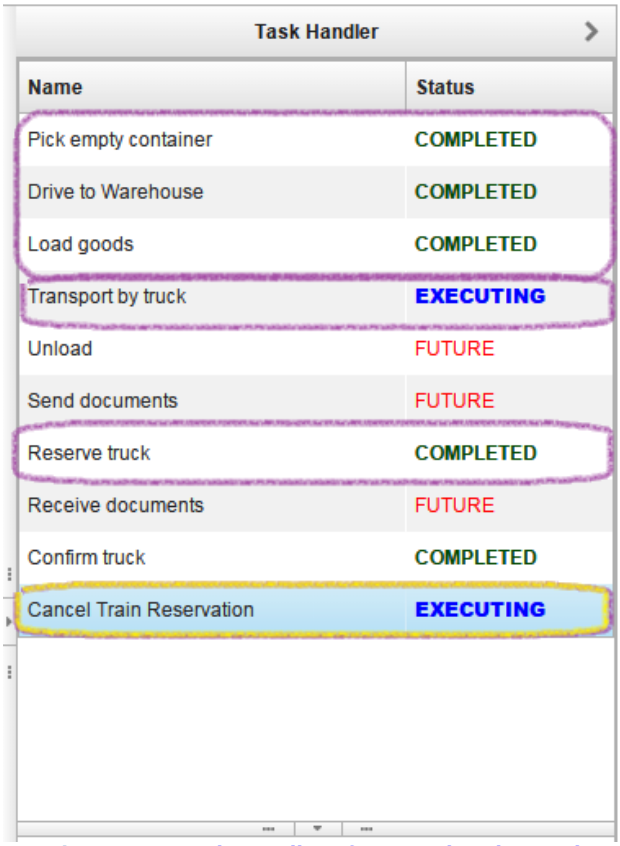

Figure 13 - Task Handler after Re-Planning and Migration

Due to a heavy congestion on the route from Venlo to Eindhoven, the estimated arrival time of the truck at the Eindhoven rail station is delayed by approximately 30 minutes, and consequently, the reserved train cannot be used for the second leg of the transportation. In this situation, the planner has to re-plan the transportation in such a way that the truck drives directly to the Rotterdam harbor instead of Eindhoven rail station. Therefore, in this transportation process, the train transportation will be removed from the list. Figure 14 shows the corresponding updated transportation process that is composed by the composition engine.

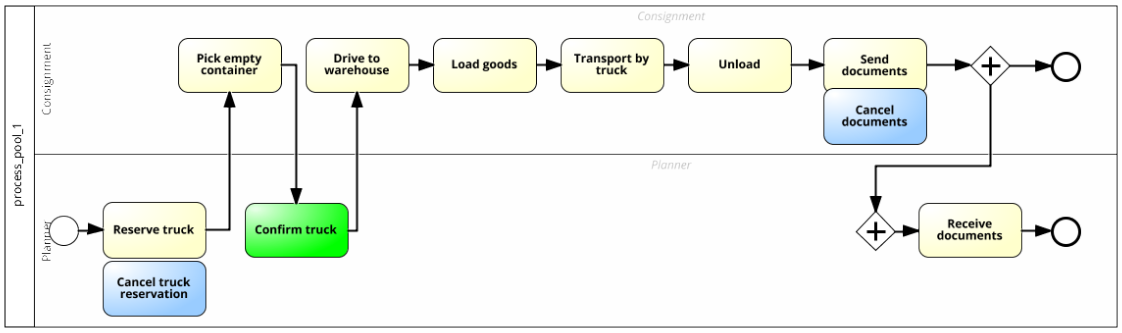

Figure 14 - Updated Transportation Process based on the Re-Planning

After deploying the updated transportation process, the orchestration engine applies the migration algorithm (see Section 2.2) in order to find the closest possible state to do the migration.

The first step is to derive the transportation trace based on the suspended transportation. This trace includes <*Reserve truck*, *Reserve train*, *Pick empty container*, *Confirm truck*, *Drive to warehouse*,

Load goods, *Transport by truck**>. The * on the ‘Transport by truck’ is used to denote that this task was executing (i.e., had not been finished) at the time of suspension.

The second step is to analyze the trace to find active compensatory tasks. Returning to the suspended transportation process (Figure 10), the ‘Reserve truck’ task has one compensatory task (i.e., ‘Cancel truck reservation’), and one confirmation task (i.e., ‘Confirm truck’). Similarly, for the ‘Reserve train’ task, the ‘Cancel train reservation’ task is compensatory and the ‘Confirm train’ task is the confirmation. Based on the first rule, the compensatory task for ‘Reserve truck’ is no longer active since its confirmation task has already been performed. However, the compensatory task for ‘Reserve train’ is still active since its confirmation task has not been performed. The other compensatory tasks of this transportation process (i.e., two ‘Cancel documents’ tasks) are not active yet since their referenced tasks have not been performed. Accordingly, the orchestration engine only finds one active compensatory task (i.e., ‘Cancel train reservation’) that should be attached to the transportation process of Figure 14.

The third step is to analyze the trace of the suspended process to find the similar tasks in the updated process. Based on the last three rules, all the tasks in the transportation trace except ‘Transport by truck’ tasks are similar based on the mentioned rules. The ‘Transport by truck’ tasks are not similar because in the suspended transportation process its locations were from the warehouse in Venlo to Eindhoven rail station while in the updated transportation its destination location has been changed to the harbor in Rotterdam. Consequently, the orchestration engine finds a list of similar tasks that need to be re-executed automatically in order to reach the closest possible state.

The fourth step is to re-execute similar tasks that have been retrieved in the previous step, and finally, in the fifth step the active compensatory task (i.e., ‘Cancel train reservation’) is attached to the updated transportation process. We have employed the first strategy (i.e., strict strategy) in this scenario.

Finally, the outcome of abovementioned steps is illustrated in Figure 15, where the transportation resumes from the state in which the ‘Cancel train reservation’ compensatory task, which has been attached to the updated transportation process, and the ‘Transport by truck’ task are enabled. Figure 13 shows the tasks handler of the current state after re-planning and migration.

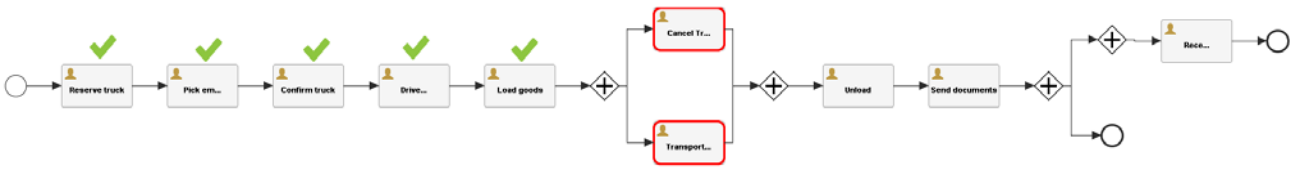

Figure 15 - Running Transportation Process – After Re-Planning and Migration

4 Conclusion

In this document, the prototype implementation of a reconfiguration algorithm in the orchestration engine of the GET Service project is described. This technique enables the migration of the information and states of a transportation process that cannot finish successfully to a new transportation process that is composed when re-planning a transportation order.

The reconfiguration technique consists of two phases: (i) pre-migration and (ii) migration. The first phase is used to store the information and states of the suspended transportation and compose a new transportation process based on the updated transportation plan. The second phase is the actual migration procedure that transfers the stored information and states to the updated transportation process. To this end, we have further extended BPMN-T modeling language (Botezatu & Völzer, 2014), which has been developed in the GET Service project, by introducing a new element, the re-planning annotation, in order to annotate the transportation process with transactional properties. In addition, we defined matching rules in order to compare the suspended and updated transportation processes. The orchestration engine employs this comparison to find the closest possible state in the updated transportation process based on the transportation trace of the suspended one.

Finally, we demonstrated the applicability of the developed technique by applying it to one of the GET Service project demo scenarios that is introduced in (Treitl, et al., 2013).

References

Arikan, E. et al., 2014. *D5.2 - Aggregated Planning Algorithms,* sl: GET Service Project.
Botezatu, M. & Völzer, H., 2014. *D4.1 - Language and Meta-Model for Transport Processes and Snippets,* sl: GET Service Project .

Botezatu, M. & Völzer, H., 2015. *D4.3- Prototype for automated composition,* sl: GET Service Project.
Dijkman, R. et al., 2011. Similarity of business process models: Metrics and evaluation. *Information Systems,* 36(2), pp. 498-516.

Grefen, P., Vonk, J. & Apers, P., 2001. Global transaction support for workflow management systems: from formal specification to practical implementation. *The Very Large Databases (VLDB) Journal,* 10(4), pp. 316-333.

Pourmirza, S. & Dijkman, R., 2014. *D7.1 - A Survey of Orchestration Engines,* sl: GET Service Project.

Pourmirza, S., Dijkman, R. & Grefen, P., 2014. *D7.2 - Design of a Reconfigurable Transportation Orchestration Engine,* sl: GET Service Project.

Pourmirza, S., Dijkman, R. & Grefen, P., 2014. *Switching Parties in a Collaboration at Run-time.* Ulm, IEEE, pp. 136-141.
Treitl, S. et al., 2013. *D1.1 - Use Cases, Success Criteria and Usage Scenarios,* sl: GET Service Project.

BLANK PAGE

The GET Service project (http://www.getservice-project.eu) has received funding from the European Commission under the 7th Framework Programme (FP7) for Research and Technological Development under grant agreement n°2012-318275.