ICT-2011.8
GET Service Project
2012-318275

Deliverable D4.1

Language and Meta-Model for Transport Processes and -Snippets

18 February 2015
Public Document

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.

SEVENTH FRAMEWORK PROGRAMME

Project acronym:	GET Service
Project full title:	Service Platform for Green European Transportation

Work package:	4
Document number:	D4.1
Document title:	Language and Meta-Model for Transport Processes and -Snippets
Version:	1.1
Attachments:	Formal meta-model (vsd, pdf) Process models and process snippet models (bpmn, pdf)

Delivery date:	3 June 2014 (20)
Revised:	18 February 2015
Actual publication date:	
Dissemination level:	Public
Nature:	Report

Editor(s) / lead beneficiary:	
Authors(s):	M. Botezatu (IBM) H. Völzer (IBM)
Reviewer(s):	Jan Mendling (WU-IS) Remco Dijkman (TUE-IS)

Version History

Version	Changes	Authors
0.1	Structure and methodology	H. Voelzer
0.2	First draft use cases for language	M. Botezatu, H. Voelzer
0.3	First draft requirements	M. Botezatu, H. Voelzer
0.4	Survey of time annotations and process languages	M. Botezatu
0.5	Revision requirements, inclusion of process models	M. Botezatu, H. Voelzer
0.6	Addressed comments by Jan Mendling	M. Botezatu
0.7	Definition of language and meta-model	M. Botezatu, H. Voelzer
0.8	Revision of presentation	H. Voelzer
0.9	Addition of event subscriptions and finalization for internal review	M. Botezatu, H. Voelzer with input from S. Pourmiza and A. Baumgrass
1.0	Addressing comments of internal reviewers	H. Voelzer, M. Botezatu
1.1	Addressing comments of EU reviewers	H. Voelzer

Table of Contents

Executive summary

This document reports on the derivation and definition of BPMN-T, a language to model transport processes. A transport process model can be seen as a refinement of a transport plan, specifying for each leg and transshipment of the transport plan, the individual tasks, transport or administrative, their dependencies, their associated actors as well as their communication, the main documents involved as well as their flow through the process, and the time constraints under which the individual tasks have to be executed.

Because the transport process is a refinement of the transport plan, each leg of the transport plan corresponds to a part of the transport process, which we call a process snippet, such that the entire process is the composition of snippets. This is a main idea underlying Work Package 4.

BPMN-T is an extension of the well-known and widely adopted OMG standard BPMN 2 [20] for business process modeling. The main extensions proposed in this document can be grouped into three packages: (1) various time annotations are important to plan the schedule for individual activities and to monitor the timeliness of their execution at runtime, (2) snippet interfaces are needed to specify process snippets in isolation and to enable their meaningful composition, (3) event subscriptions are necessary to enable status monitoring and to leverage the work on event processing in Work Package 6.

This document analyzes general usages for process models in the transport domain with an emphasis on usages for GET Service scenarios. It derives high-level as well as detailed requirements for the language and finally presents the meta-model for BPMN-T, which is also attached as a separate MS Visio file.

The language is a necessary requirement for the transport process composition (Work Package 4), the process execution (Work Package 7), status tracking and monitoring (Work Package 6 and 7) and for status migration after unexpected events and re-planning (Work Package 7).

1 Introduction

This deliverable documents the derivation and definition of BPMN-T, a language to model transport processes and process snippets. This section provides the background to this 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 platform provides transportation planners with the means to plan transportation routes more efficiently and to respond quickly to unexpected events during transportation. To this end, it connects to existing transportation management systems 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 this functionality, the GET Service platform aims to reduce the number of empty miles that is driven, improve the modal split, and reduce transportation times and slack, as well as response times to unexpected events during transportation. Thus it reduces CO_2 emissions and improves efficiency.

1.2 Work package goal

The fulfilment of a transport order can be described as a transport process, which is a set of interdependent tasks, such as "pick up container from location X", "load consignment onto container", "reserve train", "send invoice to client". A transport process model captures these tasks, shows their dependencies graphically, but also specifies the main business objects such as a waybill, their flow between the tasks, furthermore the actors of the tasks and possible additional technical information that enables the automation of some tasks in the process.

In the GET Service project, process models are leveraged to facilitate detailed monitoring and control including correlation of unexpected events and adverse conditions to transport process instances and to support cost-effective reconfiguration during execution in case an unexpected event disrupts the plan.

Work package 4 of the GET Service project develops techniques and prototypes for the creation of process models out of transport plans. The underlying idea is that the structure of the transport process corresponds to the structure of the transport plan, i.e., each leg and transshipment step of the transport plan corresponds to a well-separated part of the process, which we call a *process snippet*. Hence, the entire transport process, also referred to as an *end-to-end process* is a composition of process snippets. A snippet can be understood as a procedural representation of a logistics service. Hence, an end-to-end process is the composition of such logistics services, usually across different providers.

The composition of an end-to-end transport process is driven by a given transport plan and consists of two steps: Firstly, for each transport leg and transshipment step of the transport plan, a suitable snippet is retrieved from the *snippet repository*. How this is accomplished will be the subject of deliverable D4.2. Secondly, all those retrieved snippets are composed into an end-to-end process along the given structure of the transport plan. This composition step will be subject of deliverable D4.3.

1.3 Deliverable goal

The goal of deliverable D4.1 is to develop a language for modeling end-to-end transport processes as well as for modeling their constituent snippets in isolation. This is preliminary to create transport process models and to use them to enable transport-typical use cases.

In this report, we develop such a language, called BPMN-T, propose a formal meta-model for it and explain how it can be used. The main elements are re-used from existing process modeling languages, in particular from BPMN, the Object Management Group (OMG) standard for the Business Process Model and Notation [6] for business process modeling. However, we also identify the need for extensions to address some logistics specific requirements. The largest set of extensions come from the need (i) to express snippet interfaces in order to support the above mentioned composition and (ii) to express time constraints that are essential for transport processes. Furthermore, we introduce certain event subscription annotations to support individual transport status tracking.

1.4 Deliverable structure

This report is structured as follows: To present BPMN-T, we proceed using a dedicated methodology, which we present next, in Section 2. A survey of process modeling languages in transport and logistics is presented in Section 3. Use cases for the language are presented in Section 4. Subsequently, requirements are derived and design decisions are presented together with process examples in Sections 5 and 6. The choice of a baseline language is discussed in Section 7 and the meta-model is finally presented in Section 8.

2 Methodology for Developing the Language

In this section, we present the methodology that we used for developing the transport process modeling language BPMN-T. The overall design of this language is driven by the idea to maximize reuse of existing concepts, compatibility with industry standards and coverage of logistics use cases as exemplified by the GET Project.

Our methodology consists of the following steps:

1. Survey existing process languages and their use in transport/logistics
2. Identify domain-specific and project-related usages that use a process model
3. Identify high-level requirements for D4.1 derived from the usages
4. Iteratively transform as-is process models from D1.2 into process models that are fit for the use cases
5. Analyze transformed process models and identify detailed requirements for D4.1
6. Select an existing process language as a starting point
    a. Review candidates from the survey of step 1
    b. Select a suitable process language candidate as development baseline
    c. Analyze the candidate: where it is sufficient and where it needs extension
7. Define the extended language and the meta-model

In the following, we explain this methodology in more detail. We started with surveying existing languages to describe processes in the transport and logistics domain and the application of traditional business process modeling languages to that domain (step 1).

It was becoming quickly clear that there is no “off-the-shelf” process modeling language that is fully adequate for the transport and logistics domain. Therefore we had to develop a language, preferably not from scratch, but as an adaptation and extension of an existing process modeling language to the transport and logistics domain.

When defining a new language for a domain, it is good to be aware of the possible usages for the language in that domain, even if not all of these can be explored or even implemented within the GET Service project. We therefore brainstormed such possible usages (step 2). We documented the usages in detail that are closely related to the GET Service project.

Subsequently, we captured high-level requirements for the language, the supporting development and the execution environment, which were directly derived from the use cases (step 3). A high-level requirement typically describes a particular function or aspect within a use case from a particular stakeholder’s point of view. For example: “ability of the language to describe customized milestones for status reporting to the client of a transport order”.

Next, we wanted to derive more detailed, concrete requirements to eventually obtain the design of the language. Such detailed requirements and possible associated design decisions should be driven by, and later validated with, concrete transport process examples. We have therefore taken the as-is process models that were created for the GET Service scenarios described in D1.1 and which were part of the requirements analysis documented in D1.2, and transformed them into process models which would be suitable to support the previously identified use cases and to meet the previously captured high-level requirements. In this step, we started to restrict our focus to the use cases that are relevant for the GET Service project. This transformation was conducted in several iterations, either performing a restructuring, change of abstraction level, deletion of unnecessary elements, extension with new information, refinement or clarification of detail with the project partners (step 4).

This step included the decomposition of the end-to-end process models into snippets. It is important that the modeling methodology for processes and their snippets makes the composition as simple as possible.

We then captured detailed requirements as generalization of the new features of the process model examples (step 5). This drove further development of the process model examples and vice versa.

In the next step, we selected an existing process modeling language (“baseline”) that fitted best our captured requirements (step 6). Special emphasis was given to the languages for which we could re-use existing technology as much as possible. This step was coordinated with the selection of an execution engine in work package 7.

Finally, we formalized the syntax of the necessary extensions to the baseline language as a meta-model, and we documented their semantics (step 7).

3 Survey of Process Modeling Languages for Transport and Logistics

In this section, we present a literature overview of the currently existing process modeling languages and their suitability for representing transport processes. We review the literature from two angles: first, we review which languages from the Business Process Management (BPM) community have been applied within that community to the transport and logistics domain. Then we review to what extent the Transport and Logistics community itself has already developed or considered process modeling.

3.1 Transport process modeling within the BPM community

Process modeling languages are a means to define the business activities and the order in which they should be executed in a uniform and structured way. Among the popular languages, we can distinguish two categories: On the one hand those proposed and adopted by industry, such as: Business Process Model and Notation (BPMN) [20], UML Activity Diagrams [21], Event-driven Process Chains (EPC) [22], Business Process Execution Language (BPEL) [23]. On the other hand, there are those languages proposed and used by academia: Yet Another Workflow Language (YAWL) [24] and Petri Nets [25].

Business process modeling languages enjoy a wide adoption in domains like insurance or banking. Nevertheless their scope is much wider. Some specialized process modeling languages stem from the supply chain management field. Their capability of capturing the physical logistic operations was used by the scientific community. For example, in [4], the authors analyze the suitability of UML as a language for specifying the requirements of the logistics processes. They distinguish between the static and the dynamic view, as the former is concerned with the relevant logistics objects, their abilities and the relations between them whereas the latter is concerned with the dynamic behavior – the temporal sequence of activities and the states of the logistic objects. They state that UML activity diagrams are the most often used diagram type when applying UML to business process modeling.

The authors of [5] leverage the expressivity of BPMN to model multimodal logistic chains and they map the BPMN model to a Petri Net representation to obtain a simulation model for the entire logistics chain of operations.

Böse and Windt [6] attempt to give a solution to the problem of decentralization in the planning and control systems introducing the concept of autonomously controlled logistic systems for efficient order processing. They use ARIS[1] to model the processes and information systems in logistics with EPC flowcharts. They also model different views such as: the data view, the function view, the organizational view and the control view.

YAWL's modelling capabilities in control-flow, data and resource requirements for the logistics domain were shown in an order fulfillment process model in [7]. The modeling scope is not limited to the transport activities, but the process also contains ordering, logistics (carrier appointment, freight in transit, freight delivery) and payment.

With an eye on the current trends and the potential future shifts in industry, the authors in [8] investigate the fitness of BPEL as a language to meet the requirements of the Internet of Things[2] (IoT) concept. In the paper, they present the main technologies related to IoT together with the automated support from the business processes in logistics domain. They also present the shortcomings of WS-BPEL when it comes to design and runtime changes in the processes. Some relevant such shortcomings are: the lack of support for changing a business process instance in the case of unexpected circumstances, or the lack of support for migrating instances of old process definitions to new process definitions.

3.2 Process modeling within the transport and logistics community

The Supply Chain Operations Reference (SCOR) Model is a management tool that is used as the standard tool in supply chain management to address, improve and communicate supply chain management decisions within a company and with suppliers and customers of a company[3]. In contrast to BPMN and UML, the SCOR model does not have modeling constructs for modeling dynamic behavior. It focuses more on structural dependencies between service providers and consumers such as participation in customer interactions, product transactions and market interactions. Business process modeling capabilities as reflected in Section 3.1 are not provided by SCOR [15]. An attempt to mitigate this is the work of the authors in [26], who specify a grammar and analysis technique for SCOR-based supply chain design.

[1] ARIS is a business process management modelling software tool

[2] Internet of Things, refers to a large network of interconnected objects, enabled by technologies such as wireless, RFID and others.

[3] http://scm.ncsu.edu/scm-articles/article/the-scor-model-for-supply-chain-strategic-decisions

In [16], process mining techniques are used to derive supply chain processes from RFID events exchanged through the EPCglobal[4] standard. The authors use a process based on the SCOR model to evaluate their result. Another industry approach for logistics, which comes as a software product is [17]. It offers the possibility to model the supply chain network, to vary the transportation alternatives and to capture information regarding cost, time, capacity and delivery.

It turns out that there is no preexisting language for modeling detailed process behavior in the transport domain. We therefore have to base further work on the business process languages mentioned in Section 3.1. We will revisit these languages and select a baseline language for our work in Section 7.1.

4 Usages for a Transport Process Modeling Language

This section describes potential usages of process models in the logistics and transport domain. These usages are derived from the scenarios that were captured for the GET Service project and which are documented in Deliverables 1.1 and 1.2. Additional usages of process models within the transport domain could be imagined but are out of scope of the GET Service project. All requirements for the transport process modeling language that we present in this document will be derived from the usages in this section.

4.1 Process automation

At some point in time, a transport plan has to be executed. Its detailed “operational” version is the transport process. Already today, the transport process is partially automated, i.e., some of the tasks, e.g., resource reservation, arrival pre-notification, document handling are support by IT. For maximal automation, these tasks, or IT services, need to be integrated. A process model describes the dynamic integration for the purpose of fulfilling a specific transport order.

A natural way of integration is a process engine. There, a dedicated engine interprets the process model and calls individual services as the process model prescribes. In this sense, the process model serves as a “program” for the process engine. Process data and documents are shared between the process participants through the process model in a standardized way. The separation of the process model as a special software artifact makes it easier to change the process (facilitating re-planning, discussed below), but also enables additional use cases based on process models, such as process analysis and –improvement, and generic status tracking and –prediction. Many commercial and public-domain process engines exist. A survey of process engines suitable for the GET Service project is provided in deliverable D7.1. A process engine offers not only the capability to deploy and run a process model, but typically also to monitor, manage and administrate process instances. Administration of process instances can include capabilities to rollback, recover, or reconfigure “broken” process instances.

An engine typically comes with a process development environment, which offers various tools for system integration. Among them, there is often:
(i) A generator and an engine for *forms* to capture data from human operators,

[4] EPCglobal provides a standardized way to represent information related to RFID data for applications that need to make use of it

(ii) *A business rule* editor and -engine to separate dynamic business logic with an interface to business analysts,
(iii) Information modeling and data transformation tooling,
(iv) Script editors and engines to enable the use of short software scripts for integration,
(v) Component architecture tools to wrap applications into components to be able to call them in the process as services.

Development environment and engine are furthermore often packaged together with process analysis tools into what is called a Business Process Management Suite (BPMS). Various standards such as WS-BPEL or BPMN 2.0, SOAML, WSDL and others make BPMSs an increasingly attractive proposition for process automation and system integration.

4.2 Planning, composition, and temporal validation

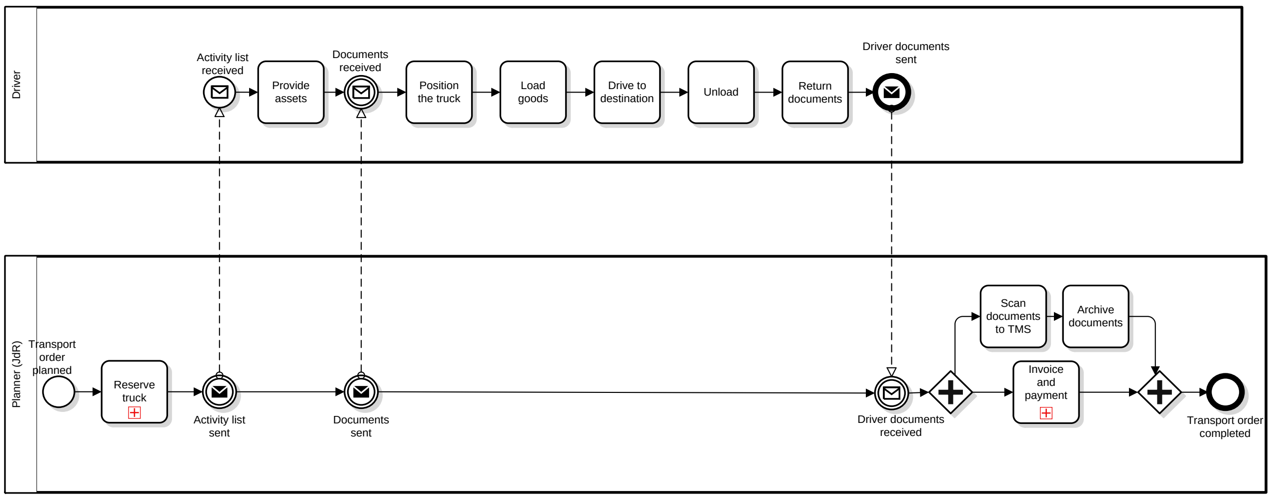


Figure 1: Simplified Process Model of Road Case 2 from D1.2

Figure 1 shows an example of a transport process model. It is a simplified model of the process for Road Case 2 as it was captured in D1.2. It shows the activities of a planner from Jan de Rijk and its interaction with the truck driver. Note that this is a single-mode example with just one leg.

A process model can be helpful in planning a transport, especially when the transport process is complex (e.g. multimodal) and contains concurrent activities. (Such examples are shown below). A process model documents the refined transport plan. This document can help make sure that the transport process is executed exactly as planned, no steps are forgotten, all steps are executed in the right order and at the right time to meet all deadlines. It graphically shows the synchronization points of different operators and formalizes their dependencies. An adequate process model contains only the required dependencies and temporal constraints, leaving room for flexible execution.

The visualization of a process model can help the planner to plan the individual tasks for each logistic step, i.e., transport leg or transshipment, but also for the composition of logistics steps. In the composition, the planner makes sure that one logistics step provides all necessary documents and notification that the next logistics step needs and that the handover is as smooth and efficient as possible. The documentation of where, when and how that happens helps later to monitor the status of the transport process and to reconfigure it dynamically in case of (unexpected) changes.

The planner can be supported by software tools that check that the planned transport process is free of simple errors and inconsistencies. For example, a software tool can check that the schedule for all activities is consistent with all known durations, waiting times, opening hours and other temporal constraints. We call this check *temporal validation*. Temporal validation is based on a process model that is annotated with temporal constraints. Note that temporal validation of a transport process model takes places on a more fine-grained level compared to route planning. Process execution data can be used to improve the accuracy of temporal planning and validation over time. Specifying only the externally required temporal constraints enables dynamic adaptation of schedule under unexpected events.

4.3 Status tracking and automated escalation triggering

Asset[5] tracking is important for optimal real-time resource allocation, as it provides visibility on available capacities and their location. *Status tracking* refers to the tracking of the status of the transport process, which includes the *process status* (which tasks have been already performed, which are next and which are in the future), the *task status* (how much of a running task has already been completed), and *schedule status* (whether the executing process is on time).

The process model can be used to visualize the process status to any interested stakeholder, such as the planner or the client. Standard process technology, i.e., monitors on top of a process engine, can be used to track process status generically. A time-annotated process model is the basis for tracking schedule status. Tracking the schedule status allows us to raise and escalate alarms in case that the transport process is seriously behind schedule. In this case, we could predict, again based on the time-annotated process model, whether and how much the deadlines are likely to be transgressed. If implemented well, such automated alarms can help to decrease the workload of transport supervisors, especially when a single supervisor monitors a large number of transport processes running simultaneously. For automated escalation triggering it is not only important to process the local data (i.e. the status and the time) but it is useful to also have access to external information that concerns the logistics process, such as weather and GPS coordinates. This would be provisioned by the information store, through events.

Note that an accurate up-to-date Estimated Time of Arrival (ETA) is not only useful to the client, who wants to know when the goods are delivered, but also to the operators in a chain who wait to take over the consignment from a previous step of the transport chain.

4.4 Process re-planning and status migration

When unexpected events occur, their severity might be so high that re-planning is required (e.g. in the freight shift scenario from the Road Case 2 described in Deliverable 1.1). In such situations,

1. A new transport plan must be created (re-planning),
2. A new transport process must be created as described above, and
3. The current status of the old transport process must be migrated to the new process.

Step 2 can be thought of as a snippet replacement: The current and future process snippets are replaced by the re-planned snippets. Step 3 also uses the process models to determine a correct

[5] By logistics assets we denote freight distribution equipment and truck drivers.

migration. Besides the functional correctness, the migration should also take into account the cost such as time, money and carbon emissions.

4.5 Data collection and offline analytics

Data collection in real-time, across all operations at every location, enables improved planning and control of transport processes.

As stated in Deliverable D1.2, offline planning uses among others:

- time-dependent deterministic information like different trip durations for different times of the day, and/or
- time-dependent stochastic information (e.g. anticipation on potential congestion on certain roads prone to high traffic loads on certain peak times)

Therefore it is desirable that the automated execution of transport processes logs duration information for individual tasks in the transport process. Process engines typically provide capabilities to log such information.

5 High-Level Requirements

This section collects the high-level requirements for the process modeling language, which were derived from the usages for process models in transport presented in Section 4. We list below, for each of the use cases, all involved stakeholders and the derived high-level requirements. This information will be used later for deriving detailed requirements for the language.

5.1 Process automation

Stakeholders:

- Transport planner and transport process supervisor
- Transport operators, e.g. truck driver
- Administrative clerks, e.g. accounting

High-level requirements:

1. The language must support modeling transport order fulfillment on a suitable abstraction level including main tasks, events, messages, data objects and ordering.
2. The language must have execution semantics and suitable engines supporting this semantics must exist.
3. The development and execution environments for the language must be able to support the integration of preexisting IT systems, e.g. reservation/booking, transport order management, accounting etc.
4. The language must include a mechanism to specify that some transport activities, e.g. driving tasks, are completed through receipt of external events, such as GPS events.

5.2 Planning, composition, and temporal validation

Stakeholders:
- Transport planner

High-level requirements:

1. The language must be able to represent the main details of a transport plan, e.g. locations, actors and schedule of main activities.
2. The language must be able to represent an end-to-end process as a composition of snippets. A snippet should be traceable to its corresponding leg or transshipment step in the transport plan.
3. The language must be able to represent a process snippet including its interfaces.
4. The language must include means to specify time related constraints in the process model that are relevant for checking the feasibility of a schedule.

5.3 Status tracking and automated escalation triggering

Stakeholders:
- Transport operators, e.g. truck driver
- Transport process supervisor: The supervisor may see the full status of the process that the planner has defined.
- Client: The client's view is a custom defined abstraction of the view of the supervisor.
- Other process participants, e.g. customs, harbourmaster, terminal supervisor

High-level requirements:

1. The language must be able to represent and visualize the status of a process model.
2. The language must be able to represent a custom defined abstraction view for a client to track status of the process.
3. The language must support specification of escalation triggers, e.g. when a deadline becomes likely to be transgressed.

5.4 Process re-planning and status migration

Stakeholders:
- Transport planner and transport process supervisor
- Transport operators, e.g. truck driver
- Client
- Other process participants, e.g. customs, harbourmaster, terminal supervisor

High-level requirements:

1. The language should facilitate the adaptation of process models to the dynamic changes in the transport plan in case of exceptional events.
2. The language must support specification of exception handling and compensation.

5.5 Data collection and offline analytics

Stakeholders:

- Transport planner

High-level requirements:

1. The language should support the specification of points in the process execution that should be logged (this would enable post mortem analysis and thus identification of patterns of undesired behavior from past transport executions and optimization possibilities)
2. The language should support configuration which data should be collected during process execution.

6 Process Model Examples and Detailed Requirements

This section reports on the transformation of the as-is process model examples that were created for Deliverable D1.2 into process models that are suitable to support the use cases discussed in Section 4. Furthermore, we derived the detailed requirements for the transport modeling language, which we also collect in this section. The detailed requirements contain design decisions that we made in order to be able to satisfy the high-level requirements listed in Section 5. This section is the core section of this document.

6.1 Process models

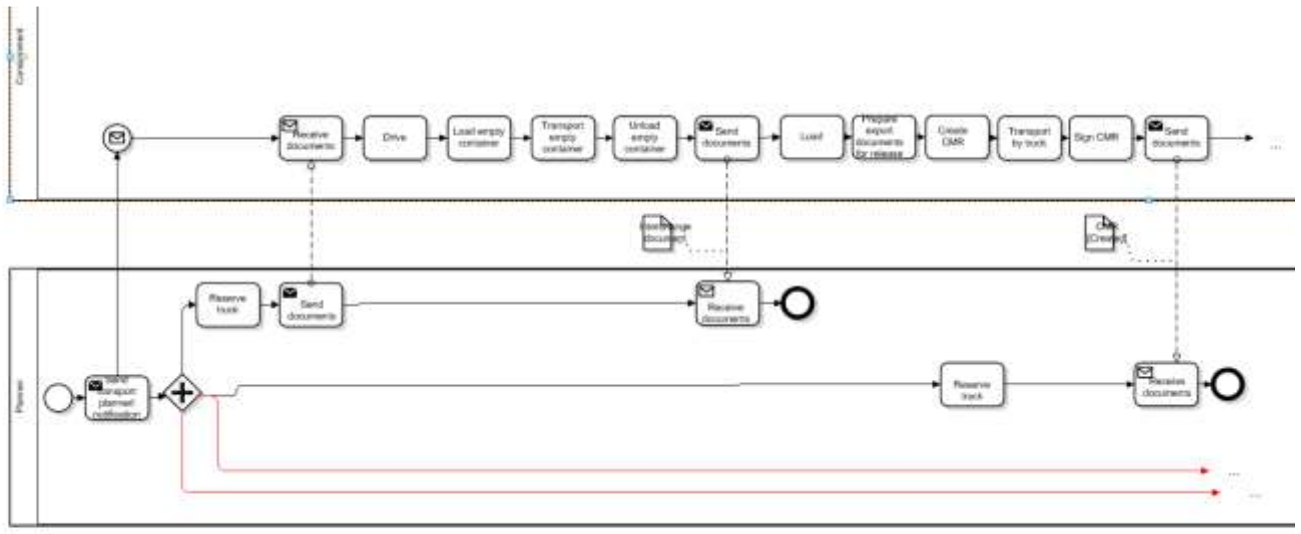


Figure 2: Initial part of the transformed process model for intermodal case 2

Figure 2 shows the initial part of the process model that we iteratively derived for the intermodal case 2, not yet showing any extensions that we introduce below. The full process model is shown in

Appendix B and also attached to this document as a separate file. We made the following basic design decisions for modeling transport processes:

1) We only model activities that are related to one consignment and its transport status. Additional activities, such as drivers break, re-fueling the truck, truck maintenance activities that are specific for the resources but not for the consignment, are not modeled.

2) We only model two pools. The upper pool is the pool for the consignment and the transport operators. The lower pool is for the planner/supervisor at the LSP(s). This greatly simplifies the process model, which would otherwise become much more complex, but it implies that multiple organizations (LSPs, TSPs) may be incorporated in the same pool, a modeling decision which may not be common for other business process models.

3) We did not model exceptional events. There are too many exceptional events that may delay or disrupt the process, so that it is not reasonable to model them. Exceptional events, e.g., “strike at the port” can be modeled if they occur frequently enough that there is some predefined exception handling, which can be incorporated into the model, cf. also Section 6.6. More commonly however, an exceptional event is recognized by a human planner and not dealt with completely automatically. Rather, the planner decides whether the event disrupts or just delays the process. In the former case, GET Service provides support for re-planning and reconfiguration of the process, cf. Section 6.6.

4) We only modeled the main data objects and documents (e.g. CMR, export declaration) of the transport process. Additional data objects can easily be added, however data objects typically clutter a diagram fairly quickly.

5) We did not model activities that have to happen before the creation of a transport process model. For example, a transport plan has to be created for each consignment of a transport order. However since the transport plan is a necessary input for the composition of a transport process model, this activity “plan transport” cannot be contained in the process model. The process model contains only activities that belong to the “operations”, i.e. the fulfillment of the plan.

6) We added information to the process models that is necessary for realizing the use cases discussed in this document. This will be explained in more detail for each use case separately below.

7) The planner pool contains multiple concurrent threads, one for each logistics step of the transport plan. This design decision will be explained in Section 6.3.2.1.

8) Dedicated status reporting messages that were modeled in the process models in D.1.2, are not shown explicitly anymore. Status reporting is done generically through the representation of the status of the executing process model in the process engine, cf. Section 6.5. As already indicated above, this is one of the major benefits of using a process model for sharing process status – explicit status messages are not needed anymore.

In the following, we derive detailed requirements for each use case that was introduced in Section 4. For each of these use cases, we first present the list of requirements and subsequently elaborate on them in a separate subsection.

6.2 Process automation

Process execution entails, similarly to other use cases, that the language supports the modeling elements that we have used in the modeling examples. Those are listed below. Automation-specific requirements were then added to the list of requirements (such as location information which could be used to trigger the start/end of an activity).

6.2.1 List of detailed requirements

The process language should support the following process modeling elements, to be able to describe the general behavior of a process. These elements are named in the terminology of BPMN 2.0 [20]

- Flow objects:
  - Event (start event, end event, message event, timer event)
  - Activity (user task, service task, script task, send and receive task, subprocess)
  - Gateway (exclusive gateway, parallel gateway)
- Connecting objects
  - Sequence flow
  - Message flow
  - Data association
- Containers
  - Pool
  - Process
- Artifacts
  - Data objects with lifecycle state

The language should support the annotation of event subscriptions to special transport tasks, e.g. “truck driving”.

6.2.2 Elaboration

The modeling elements needed in the language were directly derived from the high-level requirement 5.1.1 and from the to-be-process model examples (attached to this document). The remaining high-level requirements remain valid but they do not need to be detailed further.

The “transport” tasks should be annotated with location information, i.e. origin and destination, and the engine should be able to connect to the GPS tracking in order to be able to mark the transport activities as completed, automatically.

The engine receives external information such as GPS data by subscribing to events. A process or elements of it can be annotated with event subscriptions. When the “drive” activity subscribes to, for example, floating truck events, an event subscription for receiving floating truck data for the truck with the corresponding Id is added. The engine links the GPS information from the information store to the process data (such as location annotation for a driving activity) and can, this way, automatically trigger an activity or mark it as completed.

Event subscriptions will also be used in the status tracking use case and examples of event subscriptions are given in Section 6.5.1.

6.3 Planning, composition, and temporal validation

Recall that composition requires process snippets to be representable in the language. Process snippet decomposition will be investigated in depth in Deliverable D4.3. Here we lay some necessary foundation with the design of snippet interfaces. Before we state the detailed requirements for snippet interfaces, we first discuss some basic principles for composition in Section 6.3.1, introduce snippet interfaces in Section 6.3.2 and give examples of interfaces in Section 6.3.3.

6.3.1 Basic principles and design decisions for composition

In order to find a way of composition for end-to-end processes from simple snippets, we inspected the simplified end-to-end processes and looked for ways to decompose them into snippets. The decomposition is constrained by the following principles:

- Each snippet should be meaningful in isolation. Ideally, it should represent exactly one logistic step, i.e., leg or transshipment, or a meaningful part of it.
- The composition of snippets should be as simple as possible. Since logistics steps are composed sequentially into a chain in the transport plan, the preferred way is sequential composition.
- The interface between snippets should be easy to understand, as simple as possible but rich enough to specify basic aspects of the contract between two logistics steps in a chain.
- The end-to-end process obtained by composing the snippets must be an adequate representation of the original scenario.

The only major obstacle to these principles that we found was that the activities in the end-to-end process are not necessarily executed in the order of the logistics steps. For example if a train leg is preceded by a truck leg, the planner might still do first a reservation for the train before he reserves the truck. Therefore we decided to put the activities of the planner/supervisor onto separate concurrent threads, i.e. we have for each logistics step/snippet a separate concurrent thread of the planner/supervisor. This way, the reservation of the train and the reservation of the truck are causally independent, and hence can be done in any order.

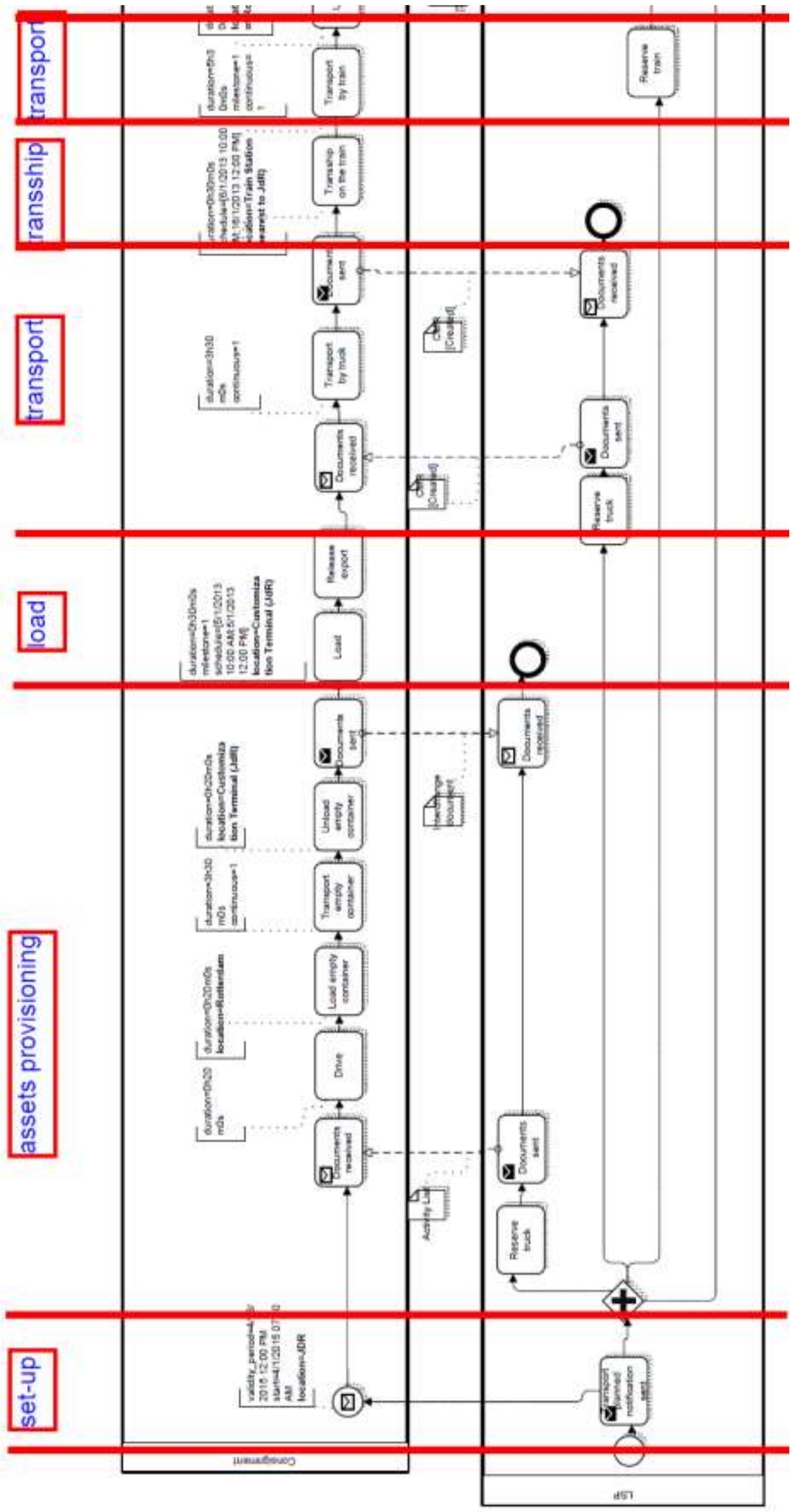


Figure 3: Snippet decomposition for initial part of process model for intermodal case 2

This can be seen in the example shown in Figure 3, which shows the snippet decomposition for the initial part of the process model constructed for intermodal case 2. The lower pool starts, after some initial task, four concurrent threads. On the first thread, there are only activities that belong to the snippet “asset provisioning”. On the second thread, there are only activities that belong to the first “transport” snippet and so forth. Two activities on different threads can be executed in any order if we only consider the control flow.

Nevertheless, the time when they will be done may still be restricted by an additional time constraint (introduced below), e.g. a reservation must be done at least 48 hours before the transport commences.

6.3.2 Snippet interfaces

A snippet can only be understood in isolation if it specifies what input it relies on and what output it produces. This information is represented in a snippet interface, which consists of an input interface and an output interface. Two snippets can be sequentially composed only if the input interface of one snippet matches the output interface of the other. A snippet may be replaced with another snippet as long as the interfaces match.

Based on the two-pool design for transport processes that was introduced above, a cut of the end-to-end process between two snippets in general cuts both pools, cf. Fig. 4. In the planner pool, mainly documents and control flow are passed on from one snippet to the next. In the consignment pool, it is documents and the consignment. Each snippet may transform the state of an *object*, i.e., consignment or document. For example, the state of the consignment may change from “loaded” to “unloaded”. The state of the CMR may change from “created” to “signed”. Because this transformation is an important aspect of what a snippet actually does, we include it in its interface description. We illustrate and detail this further in some examples in the next section.

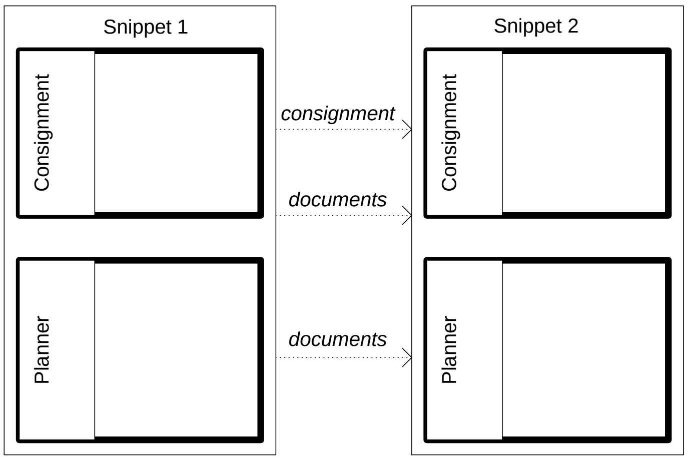


Figure 4: Snippet composition

6.3.3 Snippet examples and detailed description

For a better understanding of the high level description given above, we will provide an example of a snippet, zoom into it and offer a more detailed description of its parts and how the composition is manifested given this information.

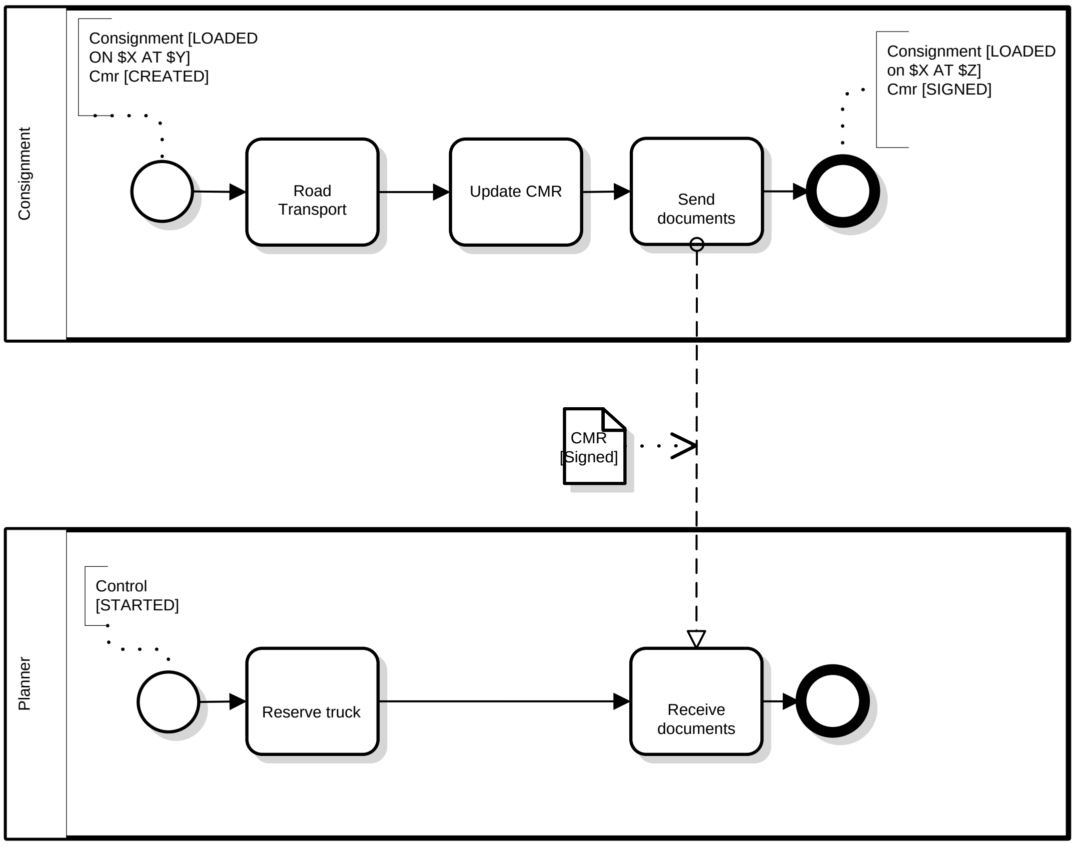


Figure 5: Example of a transport snippet

Figure 5 shows a generic snippet with a transport activity. It relies on that a CMR was created in a previous step and it is expected to be handed over in the consignment pool – denoted by the annotation of the start event "Cmr [CREATED]". Following the same representation, a loaded consignment is expected by the operator in the consignment pool, denoted by the annotation of the start event "Consignment [LOADED ON $X AT $Y]", where $X is a variable of an asset, e.g. a specific truck, and $Y is a variable for a location, e.g. "warehouse in Eindhoven". The CMR gets updated and signed in the "update CMR" activity and hence the CMR status is "SIGNED" when the CMR is handed over to the next snippet in the consignment pool. At the end of this snippet, the consignment is still loaded on the same asset but now, because of the "road transport" activity, at a different location ($Z). Thus, the entire process snippet is parameterized by the origin ($Y) and the destination ($Z) of this transport leg.

The transshipment snippet, shown in Figure 6, usually occurs between two transport snippets. Note that a snippet may, as in this case, contain only one pool. The CMR status is parameterized ($S), but it is the same in input and output as it is not changed in the snippet. The consignment changes the asset on which it is loaded but the location ($Y) remains the same in this snippet.

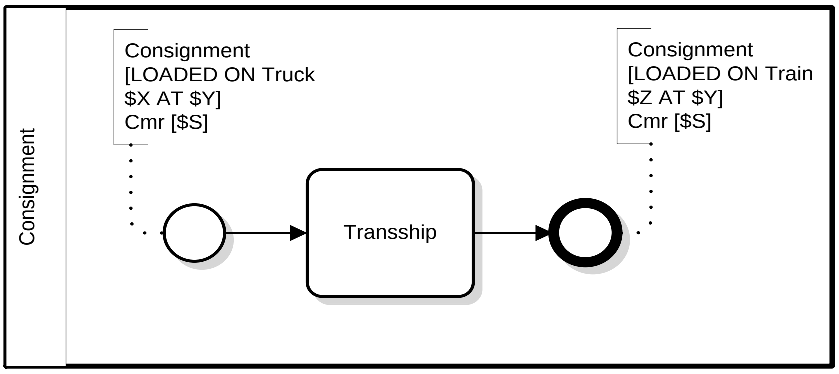


Figure 6: Example of a transshipment snippet

A "*" annotation may be attached to an interface expression at an output interface. It denotes an unbounded multiplicity of the corresponding tokens that are provided by that snippet, which is relevant for correct composition and will be described and motivated in detail in deliverable D4.3.

6.3.4 List of detailed requirements

We are now ready to list the detailed requirements for the planning, composition and temporal validation usage.

- Each process snippet has one or more start events and one or more end events.
- The language must be able to represent *snippet interfaces:*
    - A snippet interface is composed of *input interfaces* and *output interfaces.*
    - Each start event of a snippet has at most one input interface; each end event has at most one output interface.
    - An input interface (likewise: output interface) is a non-empty set of *interface expressions*. Each interface expression is of the form
        - OBJECT_ID | "Control", "[", *parameterized_state_expression, [*]"*]", <EOL>; OBJECT_ID is either "CONSIGNMENT" or "CONTAINER" or DOCUMENT_NAME, and DOCUMENT_NAME is a string (i.e. "CMR", "WAY BILL", etc.).
        - A *parameterized_state_expression* is of the form: *state_expression,* ["ON", *asset_expression*] ["AT", *location_expression*].
        - An *asset_expression* is of the form *asset_class_expression*, "VehicleId", *asset_id_expression*;
        - A *state_expression* is either a *state_id* or a *state_variable*;
        - A *location_expression* is either a *location_constant* or a *location_variable*;

  - An *asset_class_expression* is either an *asset_class_constant* or an *asset_class_variable*;
  - An *asset_id_expression* is either an *asset_id_constant* or an *asset_id_variable*;

This syntax for interfaces formalizes input and output constraints on the main business objects, i.e., the documents and physical objects used in the process as we exemplified them in Section 6.3.3. The utility of individual sub-expressions will be explained in detail in Deliverable 4.3 when the matching for interfaces will be developed.

We furthermore derived the following detailed requirements for the planning and temporal validation usages.

- The language should support *time annotations*. The selected set of time annotations is described in detail in sub-section 6.4.3.
- The language includes a set of validation rules for *well-formed end-to-end process models*, specified in sub-section 6.3.5.2.
- Each actual transport activity should be annotated with or linked to location information.

6.3.5 Further elaboration

To represent the main planning information, we included the requirements for time (see Sect. 6.4.3) and location annotations. Further elaboration for the composition is provided in Sections 6.3.1-6.3.5.2 below. Temporal validation requires time annotations which are given and elaborated on in Section 6.4.

6.3.5.1 Snippet types

After the re-design of the process models for the three scenarios that were selected for the Deliverable 1.2, we have identified a set of *snippet types* that seem to be representative and that will form the basis of the snippet repository for composing end-to-end transport processes. The snippet types are:

- **Set-up** – snippets to notify the operator of the existence of a planned transport order that commences its execution
- **Provision and position** – snippets where activities are represented that are preliminary for the transport of goods such as pick-up truck or the empty container. There are several variations for these snippets depending of the activities represented in it.
- **Load** – snippets that comprise the loading activity. There are variations for this snippet, as other related activities might occur before or after the loading activity.

- **Transport** – snippets that comprise the actual transport activity, e.g. "truck driving". Variations for this snippet exist as well and they are mainly due to the way documents are handled in this snippet.
- **Transship** – snippets to reflect the transshipment activities. Variations for this snippet also exist due to variations in the documents handling.
- **Wrap-up** – snippet that is typically present by the end of the process, where unloading of the goods and the returning of the documents are represented.
- **Invoice and payment** – snippets to represent the administrative financial activities that follow the execution of the transport order.
- **Export** – snippet to represent the particular activities that have to be performed when goods need to be exported

This classification of snippet types is based on the process examples we have modeled so far. It can be extended anytime based on new process examples and the composition approach is not restricted by this classification. The classification is intended to help to construct a snippet repository, which will be prototyped in D4.2.

6.3.5.2 The end-to-end process

Two general assumptions with respect to the process and the snippet composition are noteworthy:

- Each end-to-end process is essentially acyclic, i.e., each loop is fully contained in some snippet.
- Any separate interface expressions at the same interface (input or output) are implicitly concurrent. Specification of alternative inputs (resp. outputs) is not possible.

Both assumptions are made for simplification of the technical design, i.e., a design without these assumptions is technically feasible but considerably more complex, while the simplified design is sufficient for modeling transport processes.

The composition of two snippets matches interface expressions and possibly binds variables to each other. This will be described in detail in D4.3. Here we add some semantic composition rules specifically for application for transport processes These rules are not meant to restrict the applicability of the overall approach, but rather as a useful check to help finding modeling errors in this particular application domain.

Rules for a valid composition:

- The process has to start with a "set-up" snippet. There can be only one "set-up" snippet in the process.
- The process has to contain "transport", "wrap-up" and "invoice and payment" snippets.
- Two "transport" snippets have to be connected by a "transshipment" snippet.

- A “transport” snippet must be preceded either by a “load” snippet either by a “transshipment” snippet.
- A “wrap-up” snippet or the “invoice and payment” snippet must have at least one “load” and “transport” snippet preceding them.

6.4 Time in the process languages

In this section, we present the requirements for time annotations in the process language. Time annotations deserve special attention because (i) they are crucial for the modeling of transport processes and (ii) they are not yet supported by popular process modeling languages such as BPMN.

6.4.1 Time annotated process models in the literature

The ability to specify time related aspects, such as: deadlines, durations, service level agreements is not yet possible with the existing workflow modeling languages, however the research community has looked into this aspect and the temporal annotation of the business processes has been tackled in works like [11, 12, 13].
Surveys on time aware business process modeling, where an overview of the efforts of incorporating time patterns in workflow models in the past years can be found in [2, 14, 26]. A summary of the time patterns identified in [2] together with a brief description and an example is depicted in Table 1 from the Annex (entries 1 to 11). After performing this literature survey on time patterns we selected a set of time patterns that are applicable and relevant in the logistics domain. The time annotation patterns with a description and the motivation for incorporating them are presented in the following.

6.4.2 Time annotations

The envisaged temporal validation of a process model takes as input the constraints that are associated with the end-to-end business process and the estimated durations for the activities. In case the time annotations are not consistent with the constraints (i.e. the start of a certain activity that is constrained by a time window, does not fit within the time window), the planner is notified.
The estimation of task durations can be based on historical data. Each executed transport process logs duration and can therefore contribute to improved estimation in the future.

6.4.3 List of time annotations

The following time annotations were identified as detailed requirements for the process modeling language:

- Deadline
  - A deadline is associated with an activity or event. Specification of a deadline for the execution of an activity or of the entire process represents the fact that that activity or the process has to be completed before the date specified as a deadline. Each transport order has a due date for its completion. This due date is a key parameter

for whether a transport order is accepted and remains key during fulfillment, since violation of the service level agreement incurs non-negligible costs.

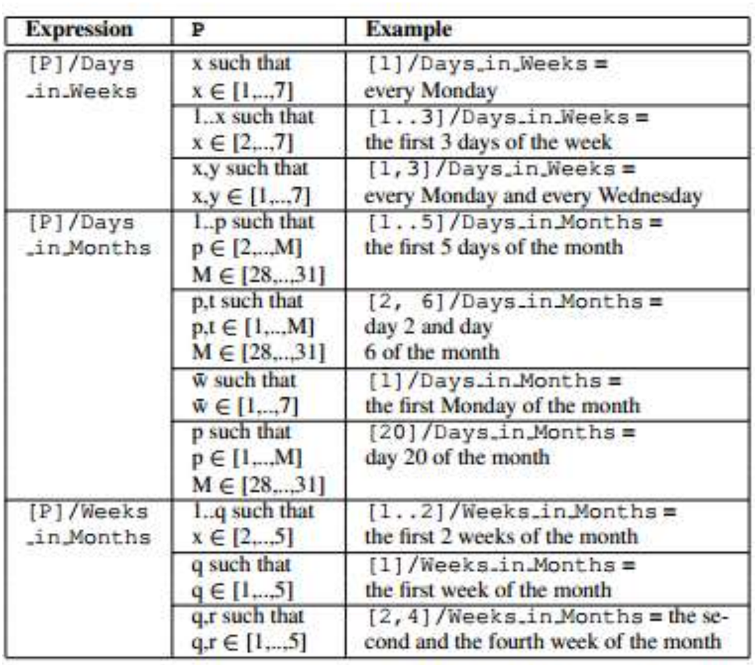
- Deadline is specified as a date, in the format YYYY-MM-DD'T'hh:mm
  - Example: the transport process has to end before 8 am on the 4th of July 2014, therefore deadline= 2014-07-04T08:00

- Starting time
  - Starting time is associated with an activity or an event. Specification of a starting time represents the exact date when an activity or the entire process is planned to start.
  - A starting time is specified as a date, in the format YYYY-MM-DD'T'hh:mm
    - Example: the transport process is planned to start on the 1st of December 2014 at 8 am. Therefore, starting_time=2014-12-01T08:00.

- Duration
  - Duration is associated with an activity. Annotating the activities with the expected time necessary for their completion is crucial for computing the estimated time of arrival (ETA) and for assessing the risk of deadline transgression. Duration estimations can be based on historical transport order execution log data and continuously refined through comparison of the estimated duration with the actual duration observed in the transport process execution.
  - A duration is specified as an Integer that represents the number of minutes
    - Example: the estimated duration for the activity of transporting the goods from A to B is 3 hours and 27 minutes, therefore duration=207.

- Schedule (time window)
  - A schedule can be associated with an activity. By a schedule restricted activity we denote an activity that can only be effectuated within a given time window. This is relevant in logistics, as often, drivers have to stop at facilities that operate under certain opening hours or have to arrive at locations within pre-allocated time slots. As mentioned above, this can be daily time when we want to specify one or more time windows within a day or periodic time when we want to specify more complex schedules.
  - A schedule is specified either as *daily time* or as *periodic time* (see below.)
  - Daily time: when we refer to one or more time windows in a day, therefore we have a set of pairs of type [date, date] ([YYYY-MM-DD,hh:mm; YYYY-MM-DD,hh:mm]). A case of such a daily time is when the planner has agreed a time window for loading and unloading.
    - Example daily time: the truck is allowed to unload between 7am and 9am. Therefore schedule=daily_time@07:00-09:00

- Periodic time: when there is temporal information that repeats periodically, and which therefore needs to be expressed in a more complex way through some so called periodic expressions. An easy way of representing such expressions is given in [18], which in turn was inspired from [19]. An example of such a periodic time is when we want to specify the opening hours for an institution that needs to be visited, and it might be needed to state the opening hours for the first five days of the week and opening hours for the 6th day of the week and other opening hours for the 7th day of the week. The way these periodic expressions are specified is expressed in the following table taken from [18]:



Expression	P	Example
[P]/Days_in_Weeks	x such that $x \in [1,...,7]$	[1]/Days_in_Weeks = every Monday
	1..x such that $x \in [2,...,7]$	[1..3]/Days_in_Weeks = the first 3 days of the week
	x,y such that $x,y \in [1,...,7]$	[1,3]/Days_in_Weeks = every Monday and every Wednesday
[P]/Days_in_Months	1..p such that $p \in [2,...,M]$ $M \in [28,...,31]$	[1..5]/Days_in_Months = the first 5 days of the month
	p,t such that $p,t \in [1,...,M]$ $M \in [28,...,31]$	[2, 6]/Days_in_Months = day 2 and day 6 of the month
	$\bar{w}$ such that $\bar{w} \in [1,...,7]$	[1]/Days_in_Months = the first Monday of the month
	p such that $p \in [1,...,M]$ $M \in [28,...,31]$	[20]/Days_in_Months = day 20 of the month
[P]/Weeks_in_Months	1..q such that $x \in [2,...,5]$	[1..2]/Weeks_in_Months = the first 2 weeks of the month
	q such that $q \in [1,...,5]$	[1]/Weeks_in_Months = the first week of the month
	q,r such that $q,r \in [1,...,5]$	[2,4]/Weeks_in_Months = the second and the fourth week of the month

- Example periodic time: the driver has to arrive at a facility "X" that operates from Monday to Friday between 9am – 6pm and on Saturday and Sunday between 9am – 13pm,thus, schedule_periodic_time= {09:00-18:00=[1..5]/Days_in_Weeks; 09:00-13:00=[6..7]/Days_in_Weeks}

• Time dependency between activities or events. This is a temporal specification which states that one activity's start or finish time depend on the start or the finish time of another activity. This can be a *lead time* or a *lag time*.

  o To specify a lead or a lag temporal dependency for one activity one needs to specify the following:

    - <amount>: the amount of time that is specified as time constraint between the two activities – an Integer.

    - <predIsStart>: a Boolean specifying whether the time dependency is relative to the start of the preceding activity

    - <succIsStart>: a Boolean specifying whether the time dependency is relative to the start of the succeeding activity

    - Lead: Activity B follows activity A in the process such that B shall start (resp. end) only if A has started (or ended) by at least the specified amount of time.
        - Example: the truck should be reserved at least 12 hours before the truck driver will start driving it. Therefore the “lead” temporal dependency would be specified as follows: amount=-720, predIsStart=0, succIsStart=1.
    - Lag: Activity B follows activity A in the process such that B can only start (resp. end) only after at most the amount of time specified in the constraint since A has started or ended.
        - Example: the “process invoice and payment” task should start at most 5 days after the goods were unloaded. Therefore the ”lag” temporal dependency would be specified as follows: amount= 7200, predIsStart=0, succIsStart=1
- Validity period
    - A validity period is associated with an entire process or snippet. It is a special time annotation, different from the above in that it does not apply to the process elements at runtime, but rather to the process model at deployment/planning time. It defines the time frame in which the process model is *valid*, i.e., should be used. For example, if a new legislation becomes effective, old procedures may not be used anymore and new ones should be used instead. Also organizational changes can be the reason for a dedicated validity period.
    - A validity period is specified as a pair of dates: start date, expiry date: (YYYY-MM-DD’T’hh:mm; YYYY-MM-DD’T’hh:mm)
        - Example: The validity period for the current logistics process is: ValidityPeriod=(2014-01-01T08:00,2014-12-01T08:00)

6.4.4 Further elaboration

Time annotations are required to represent relevant planning details in the process model, to conduct temporal verification on task level, and to provide the preliminaries for timing watchdogs, which are introduced below.

The selection of time annotations was partly made based on a survey on how time related aspects were integrated in the workflows in the literature. We also added some new time patterns that we found relevant. Table 1 from the Annex presents a summary of the time annotations found in the literature.

For the current work, we left aside time patterns such as: time based restriction, time dependent variability, and required delay. Time constraints with calendar support and resource utilization constraints, are modeled through the “schedule” time annotation and they are both concerned with the resources involved in executing the tasks and the language we are developing is both concerned with the process view and with the resources involved in the execution. For required delay, no convincing logistics example was identified. Time dependent variability can be captured at different levels (in the planning algorithms for instance), and therefore it is not used for annotation at the process level. An example of time dependent variability is: if an order is received before 12pm, the products will be delivered the same day otherwise the next day.

6.5 Status tracking and automated escalation triggering

6.5.1 List of detailed requirements

A process may contain *timing watchdogs*, which are associated with activities.

- There is at most one watchdog associated with each activity.
- A watchdog is associated with a task only if there is also an (absolute or relative) deadline associated with it.
- A watchdog contains a <tolerance level>, which is an integer that specifies the threshold (in percentages) at which an alarm should be triggered, relative to the total duration of the process. For example, a value of 1 for the tolerance level means that if a deadline transgression is observed or foreseen for one task, an alarm is raised only if the transgression represents at least 1% of the total estimated duration for the process.

A process may contain client milestone elements.

- A milestone can be represented by a separate flow element (signal event), which is incorporated into the process flow or by a "MILESTONE" annotation of an activity.

A process may contain *event subscriptions,* which can be associated with activities, events, sub-processes or the entire process.

Examples:

1. `SELECT * FROM traceGETService(source = $truckid);` returns all the trace information related to the truck with id $truckid.
2. `Select * FROM pattern [every a=ContainerLoaded(source= 'ABC') and b=Confirmation(source= 'ABC')];` returns all *ContainerLoaded* for one Confirmation for a shipment with an ID 'ABC'

The allowed syntax for event subscriptions is a subset of the ESPER language and will be detailed within Work Package 6. The association of a process element (activity, process) either represents the scope of the event subscription or its trigger. If the scope is a particular task (e.g. "truck driving"), then the subscription starts when the task starts and ends when the task is completed. The associated element (task, event) can also represent the trigger for the start or the end of a subscription. The distinction between trigger and scope is made via an attribute of the query, which is initialized when the query is attached to its corresponding process element.

One should be able to track process status:

- The process status should be represented in terms of activities completed, activities being executed, and activities that remain possible in the future.

One could be able to track task status:

- Each task may be labeled “CONTINUOUS” to distinguish between the discrete and continuous task, i.e., to signify for which tasks the internal status should be tracked. For this purpose a progress bar can be implemented which would allow to visualize the ratio between the elapsed time since the start of the execution and the expected duration.

6.5.2 Elaboration

The correspondence of the detailed requirements to the high-level requirements in Section 5.3 is clear: Watchdogs implement escalation triggers, milestone annotations implement the customized clients view, and the remaining requirements enable detailed status tracking for the planner.

Status tracking enables the visualization of the process status to the planner/supervisor at runtime. The representation of process status in terms of three groups of tasks (past, presence, future) is a simplification of the usual representation in terms of a control flow token distribution over the process model. This simplification is meaningful because we assume the processes to be mainly acyclic. This representation in terms of the three groups of tasks can be visualized by coloring each group in a separate color in the process model. Another group, the set of tasks that were not executed but are also not possible in the future anymore remains uncolored.

The functionality of a watchdog is encoded in a query (event subscription) that is executed when a task (with the watchdog flag on) is enabled. This query takes as parameter the temporal annotations for that task (i.e. duration), the estimated duration of the process, and a tolerance level ε. It will raise an alarm in real time, in case transgressions occur, or a high risk of transgression is foreseen (which can also apply for future tasks). The deadline transgressions or the risk of transgressions are signaled only if the current or the foreseen values of the transgression are higher than a value computed relative to the tolerance level ε specified and to the time buffer that that activity has. The precise formula will be given in D4.3. One query to implement the watchdog functionality is:

```
Select * from pattern [every a= activityLog(activityID = $activityID,
activityState = 'Executing') -> (timer:interval($duration*$tolerance) and
not activityLog(activityName=a.activityName, activityState =
'Finished'))].
```

Subscription to other types of events can be done similarly. A subscription query is associated to control-flow point or scope. The execution engine then receives a stream of events from the event engine for this subscription and can trigger associated actions, e.g., display of a message.

We specified in the high level requirements what the client's view is, namely, a custom defined abstraction of the planner's view. By custom defined abstraction we mean a set of client defined milestones – points in the process execution which, when reached, the client has to be notified. To support such functionality we will define “milestone” attributes for tasks. The significance of this attribute is the following: when a process is enacted and the currently executing task is labeled as a milestone, then the client is notified when that task is completed.

The different views (client’s view and planner’s view) entail the distinction of special tasks that can be tracked internally in a more continuous way (to distinguish between the tasks that are discrete and those that are continuous). This can be specified by an attribute “continuous” which differentiates between discrete and continuous tasks (value 1 implies that the tasks is a continuous task and value 0 – the default value, implies it is a discrete task). In order to track the status of these tasks, there has to be a tracking service that is called when an executing task with the “continuous” flag set to value 1 is encountered. A possibility to implement such a tracking service would be: one knows when the task was started, how much time has elapsed since it started and its estimated duration

$$progress = \frac{current\_time - starting\_time}{estimated\_duration} * 100$$

As in the equation above, one can give, as an estimation of the progress, the ratio between the elapsed time since the start of the execution and the expected duration.

6.6 Process re-planning and status migration

6.6.1 List of detailed requirements

- A process may include explicit behavior for compensating cancellations due to unexpected events. Such behavior can be specified with the following modeling elements:
    - Compensation task, compensation events, event sub-process

6.6.2 Elaboration

Process re-planning, from an implementation view, represents adjusting the existing process model based on a new transport plan. Adjusting the current process model is mainly performed by deleting/adding elements (snippets) from/to the process model.
From a composition view, re-planning has the same requirements as planning. However, when doing the transition from an old process model to a new process model (subsequent to re-planning) we must ensure a “valid” transition. By valid we denote for example if “transport by train” was replaced with transport by vessel, the train reservation should be cancelled (i.e. the “Reserve train” task should be accompanied by a compensatory boundary event, and in this case a compensatory task “De-reserve train” may also exist. BPMN supports cancelation and compensation, as in BPMN specification, page 302 "Compensation is concerned with undoing steps that were already successfully completed, because their results and possibly side effects are no longer desired and need to be reversed."

6.7 Offline optimization

6.7.1 List of detailed requirements

- Tasks for which execution information is to be logged at runtime can be labeled with “LOG”

6.7.2 Elaboration

Storing the information regarding the execution of transport orders is valuable data for offline optimization. Execution logs can be customized by having the possibility of expressing precisely the information related to which tasks should be logged. This information can be used to identify patterns for deadline transgressions or other undesired situations, per specific activities. During the execution of a process, the logs will be populated with all available static and dynamic content. By static content, we refer to the information that is known before the execution of the process (deadline/start-end locations/expected durations) and by dynamic content we refer to all the information that arises at runtime (actual departure/arrival time, actual start-end time and the real durations for each task). Those tasks for which information will be logged are labeled with "LOG".

7 Selecting a Baseline Language

In this section, we argue that BPMN 2.0 is the best baseline language for our purposes, i.e., the language for modeling transport processes should be derived by extending BPMN 2.0.

Since BPM provides concepts, techniques and tools that are applicable to transport processes, it is reasonable to apply those to the transport domain and hence use existing modeling languages and tools as much as possible rather than develop them from scratch for the transport domain. This way, we hope to be able to re-use existing modeling tools, execution engines and analysis techniques. Therefore we aim at extending one of the business process modeling languages surveyed in Section 3.

7.1 Revisiting the surveyed languages and selecting a baseline

The languages surveyed in Section 3 have known advantages and drawbacks. From the modeling perspective, UML Activity Diagrams (UML-AD) and BPMN have quite similar modeling capabilities, just using different notations. When it comes to execution, a shortcoming for UML-AD is that there is no mapping to any business process execution language specified in the normative documents so far (OMG, 2011 – the latest). There has been a focus in the past years on such a mapping, as in [9] where the authors propose a meta-model based transformation from UML-AD to the WS-BPEL language, but there is not a complete automatic mapping solution yet. WS-BPEL on the other hand, has a standardized execution semantics and strong industry adoption, but it has no standard graphical notation and is not suited for business analysts.

BPMN 2.0 was developed to bridge this gap. Initially designed as a process description language for business analysts with easily understandable graphic notation, it was later complemented with means to express all necessary execution detail and with a formal execution semantics. Many execution engines exist; a survey is provided in D7.1. Examples of freely available engines include Activiti, Bonita BPM, Camunda BPM, and jBPM. BPMN is currently widely supported: The bpmn.org website currently lists 74 tool vendors that support BPMN in their tools.

YAWL on the other hand was, as an academic language, not designed for wide industry adoption. YAWL does have a set of strong tools supporting it, but BPMN based developments will be more likely to find exploitation in the future because of the larger community.

To summarize, BPMN 2.0 is a good starting point for our purposes, because it is widely supported by tools and engines in the market, it has the same or better expressiveness as the other candidates and it has high emphasis on both, the design and analysis as well as the process automation use cases.

7.2 Necessary extensions

Reviewing the detailed requirements from Section 6 against the capabilities of BPMN 2.0, we identify the following necessary extensions to BPMN 2.0:

- Snippet interfaces associated with start and end events with the details specified above
- Selected time annotations
    - Deadline
    - Planned starting time (starting time can be represented in BPMN through timer events, however their semantics is not what we intend here: In BPMN, an internal clock throws an event which is caught by the timer event, which directly influences the control flow. A planned starting time is merely an annotation without operational execution semantics, which is however used for other use cases such as schedule validation and automated alarm triggering.)
    - Durations
    - Schedule constraints
    - Lead and lag time constraints
    - Validity period
- Time watchdog specification
- Event subscription annotation
- Distinction of "continuous" tasks
- Distinction of "loggable" tasks
- Location information associated with tasks (In BPMN, *pools* and *swimlanes* are often used to represent location information. This would be inconvenient for transport processes as we would get too many pools/swimlanes which would result in cluttered models. Therefore, we prefer a separate annotation)

8 The Metamodel of BPMN-T

In this section, we present the metamodel that we have created to meet the detailed requirements presented above. We call it BPMN-T, where "T" stands for "transport". BPMN-T extends the BPMN 2.0 metamodel and it connects to, and is part of, the overall data model for the GET Service project. The formal metamodel is attached as MS Visio (vsd) and pdf files to this document.

We first present an overview in Section 8.1, then we detail the snippet interfaces extensions in Section 8.2 and finally we discuss the time annotations in Section 8.3.

8.1 Overview

Figure 7 shows the part of the BPMN-T metamodel that is connected to the other parts of the GET Service data model, viz. to parts of the planning and the administrative data model. The classes labeled "BPMN::*x*" are taken from the original BPMN 2.0 specification [20]. Those original BPMN 2.0 classes and their associations are completely maintained in BPMN-T, where we only add new classes and associations. However, not all original classes of BPMN 2.0 are shown in this document or the attached metamodel, only those to which we directly relate with BPMN-T.

The classes labeled "Planning::*x*" and "Administrative::*x*" are taken from the GET Service shared data model.

The root element of the BPMN models we are using here is called a *collaboration or collaboration diagram* in BPMN 2.0. There is exactly one collaboration diagram for each consignment as well as a (multi-modal) *transport chain* that represents the transport plan for that consignment. A transport chain in turn is a sequence of *logistics steps*, where each logistics step is either a (transport) *leg*, or a step between two legs, which is called here a *connection point*, e.g., a transshipment. More detail about the planning concepts can be found in the GET Service data model.

A collaboration diagram consists of *participants*, where each participant is represented by a pool, i.e. we have typically two participants in our models. Each participant has its own *process*, where the processes of different participants communicate by messages. A process is composed of *flow elements*, where two particular classes of flow elements are *events* and *activities*. Each activity is associated with one logistics step and each activity can be associated with multiple resources.

An activity is either atomic, i.e., a task, or composite, i.e., a subprocess. Activities are extended in BPMN-T with the various flags (milestone, continuous, log) that were introduced above. Each activity has a location annotation, which can also be derived from the associated logistics step in the transport plan.

Also, each flow element has associated documentation, which is needed for holding the event subscriptions. The documentation field holds the textual representation of the event subscription which leads to event publications/subscriptions towards the event aggregation engine.

Finally, please note that the validity period is associated with the entire collaboration diagram, i.e. the root element of the BPMN-T model.

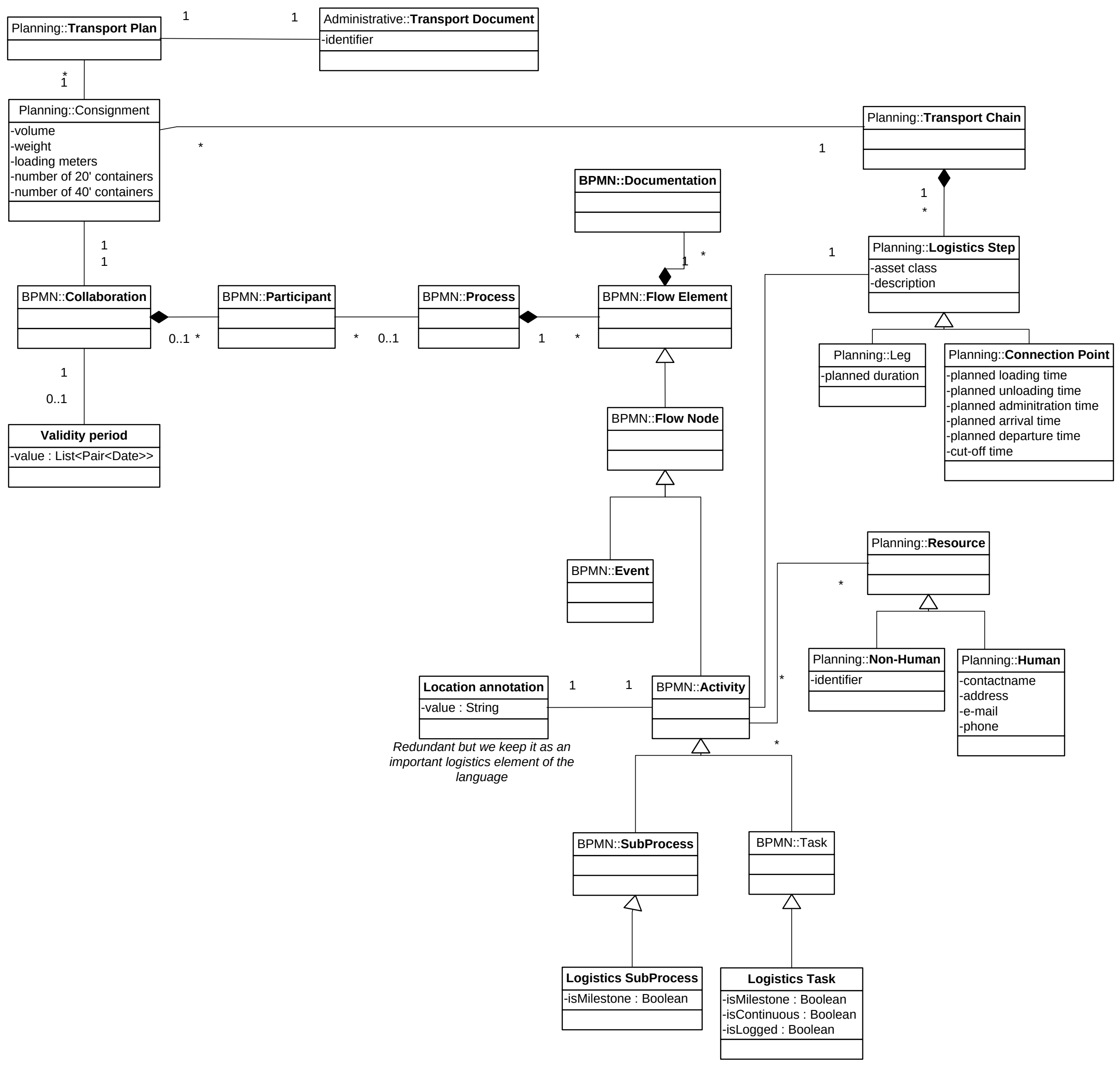
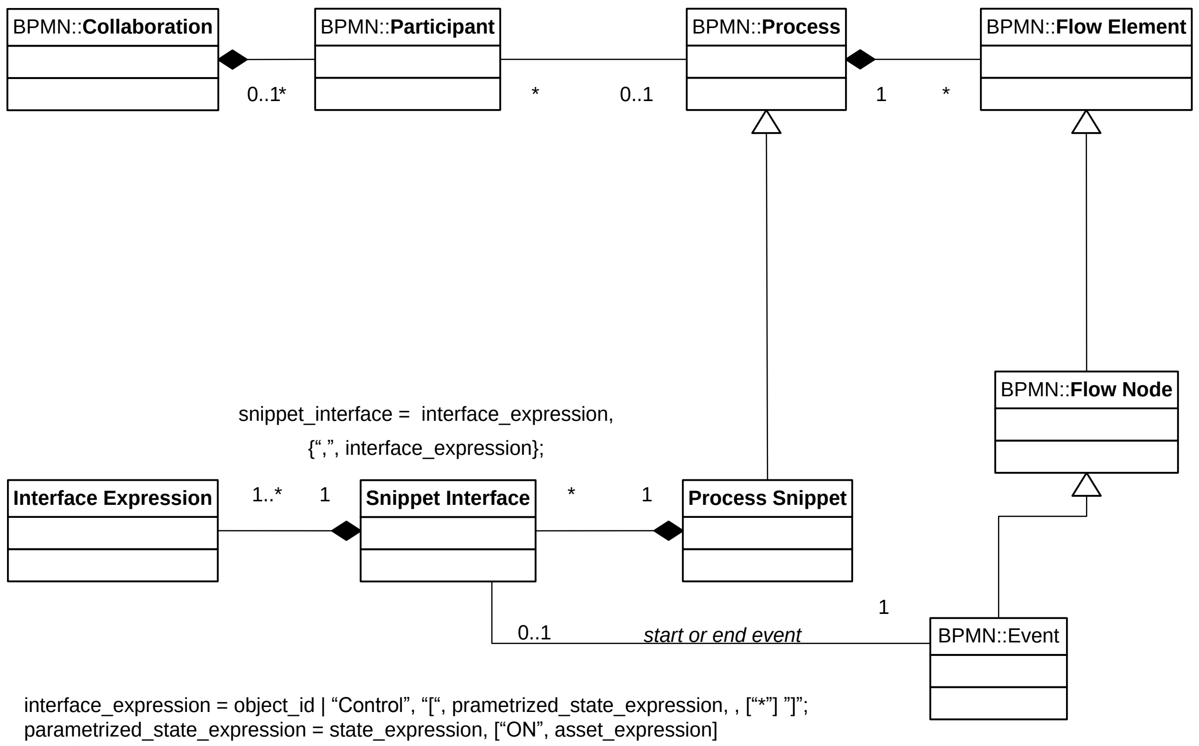


Figure 7: Part of BPMN-T meta-model that is linked to the planning data model

8.2 Snippet interfaces

Figure 8 shows the part of the BPMN-T metamodel that represents the snippet interface design. A process snippet is a special kind of process, one which is extended by snippet interfaces. There is at most one interface for each start and end event and it consists of one or more interface expressions. The syntax of an interface expression is given as a grammar in EBNF in the bottom of the Figure 4.



interface_expression = object_id | “Control”, “[“, prametrized_state_expression, , [“*”] ”]”;
parametrized_state_expression = state_expression, [“ON”, asset_expression]
, [“AT”, location_expression];
asset_expression = asset_class_expression, “VehicleID”, asset_id_expression;
state_expression = state_id | state_variable;
location_expression = location_constat | location_variable;
asset_class_expression = asset_class_constant | asset_class_variable;
asset_id_expression = asset_id_constant | asset_id_variable;

object_id= ^[a-zA-Z0-9_][a-zA-Z0-9_]*$
state_id = ^[a-zA-Z]+(?:[\s-][a-zA-Z]+)*$
state_variable = (?<=\s+|^)\$[a-zA-Z_][a-zA-Z0-9_]*$ // starts with $
location_constant = ^[a-zA-Z]+(?:[\s-][a-zA-Z]+)*$
location_variable = (?<=\s+|^)\$[a-zA-Z_][a-zA-Z0-9_]*$ // starts with $
asset_id_constant = ^[a-zA-Z_][a-zA-Z0-9_]*$
asset_id_variable = (?<=\s+|^)\$[a-zA-Z_][a-zA-Z0-9_]*$ // starts with $
asset_class_constant = ^[a-zA-Z_][a-zA-Z0-9_]*$
asset_class_variable = (?<=\s+|^)\$[a-zA-Z_][a-zA-Z0-9_]*$ // starts with $

Figure 8: Metamodel for snippet interfaces

8.3 Time annotations

Figure 9 shows the meta-model for time annotations in BPMN-T (except for validity periods which were shown above in Section 8.1). Each flow node, i.e., event or activity may be annotated with a deadline, duration, starting time or schedule. A schedule constraint is specified either as a “daily time” constraint or a “periodic time” constraint. For each of these is specified further by the grammar below each of the concepts in Figure 6.

A lead/lag constraint is associated with two flow nodes. One is thought to precede the constraint (“from” association) and one is thought to succeed it (“to” association). If the amount in the lead/lag constraint is negative, we have a lead time specified, otherwise a lag time. The Boolean flags determine whether the constraint is a Start-to-Start constraint (both flags true), Finish-to-finish constraint (both flags false) or mixed.

A watchdog specification is associated with each task.

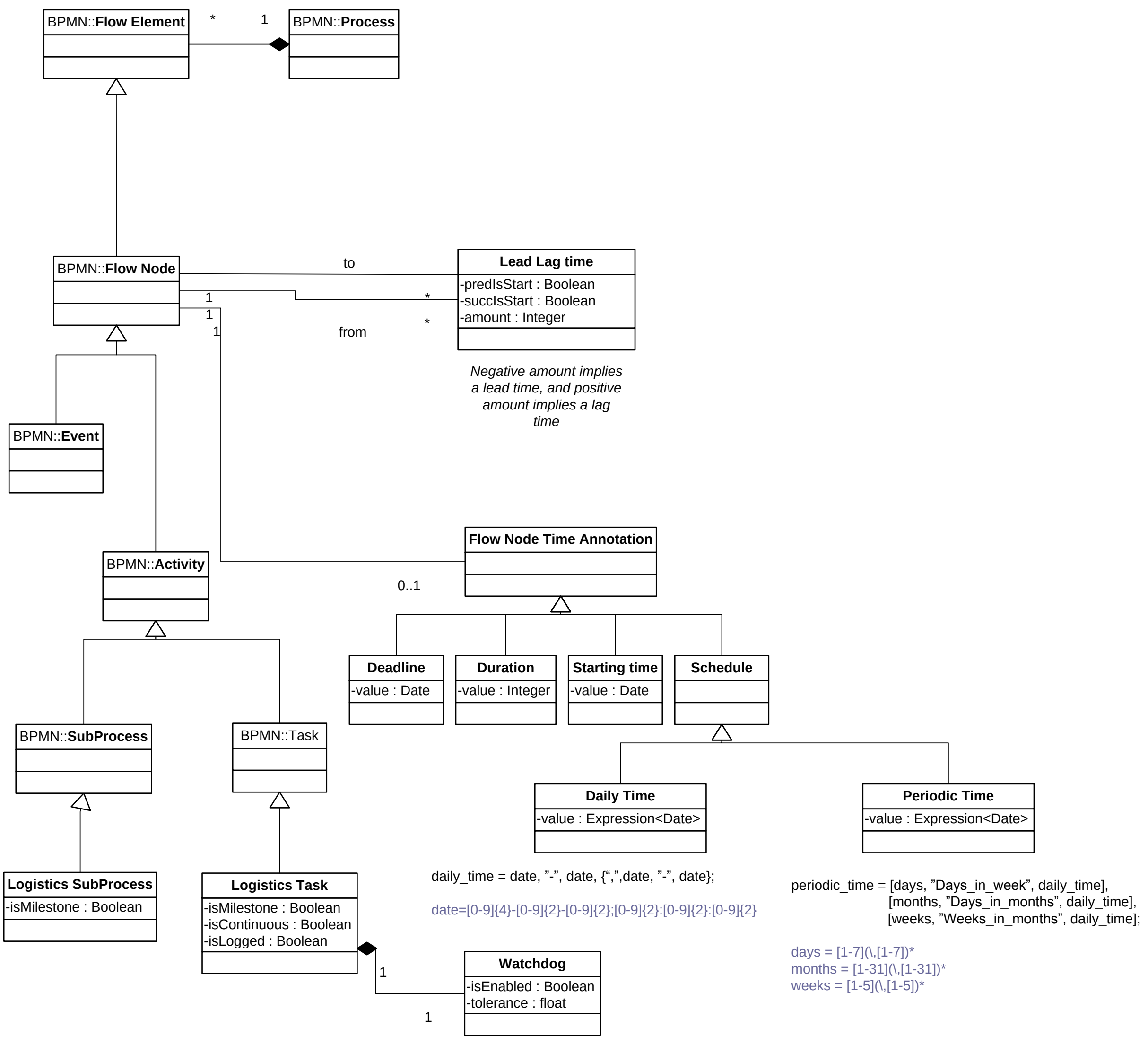


Figure 9: Meta-model for time annotations in BPMN-T

8.4 Event subscriptions

The meta-model for event subscriptions is derived in a straight-forward way from the detailed requirements and is presented in Fig. 10. Multiple event subscriptions may be associated with a process, subprocess, task or event. A subscription is represented by a string of the Esper event query language. More detail about the Esper language can be found in the corresponding deliverable D6.2.

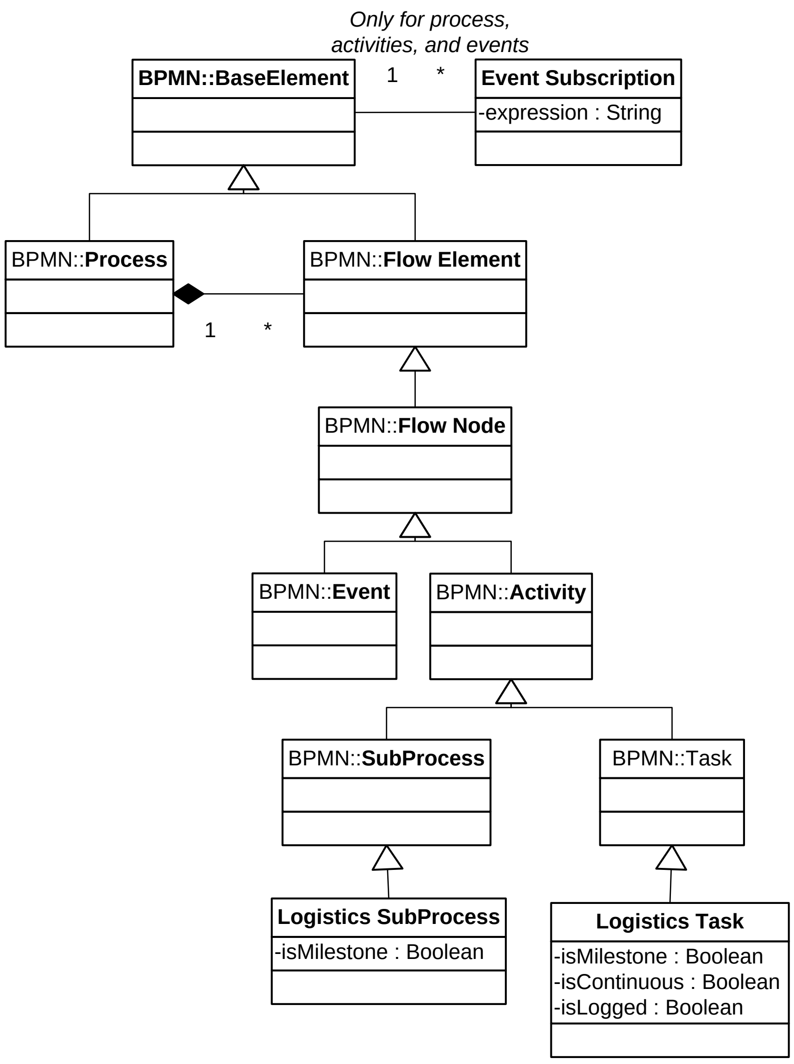


Figure 10: Meta-model for event subscriptions

9 Conclusion

In this report, we derived BPMN-T, an extension of the business process modeling language BPMN 2.0 for modeling transport processes. We argued that BPMN 2.0 is the best starting point for deriving such a language and we obtained requirements for the extension from use cases from the transport domain. The main extensions are various kinds of time annotations, interface specifications for transport process snippets, and event subscription annotations.

We also derived modeling assumptions to deal with the complexity of real, in particular multimodal transport processes and to enable flexible process composition and re-composition from snippets.

In the next steps within the GET Service project, we will further validate the suitability of our design by developing and testing the composition of process snippets as well as by the use of BPMN-T processes in the prototypical realization of transport process status tracking, automated alarm triggering, process execution, and process reconfiguration.

References

[1] H Mili et al. - Business Process Modeling Languages: Sorting Through the Alphabet Soup, ACM Computing Surveys, Vol. 43, No. 1, Article 4, Nov. 2010

[2] C. Niculae - Time Patterns in Workflow Management Systems, Master's Thesis Department of Mathematics and Computer Science, Eindhoven University of Technology, 2011

[3] Tadao Murata - Petri Nets: Properties, Analysis and Applications, Proceedings of the IEEE, vol. 77, no. 4, April 1989

[4] Scholz-Reiter B, Kolditz J, Hildebrandt T, UML as a basis to model autonomous production systems. In: Cunha PF, Maropoulos PG (eds) Digital enterprise technology: perspectives and future challenges, 1st edn. Springer, Berlin, 2007

[5] Ryszard Koniewski, Andrzej Dzielinski and Krzysztof Amborski ,Use of Petri Nets and Business Processes Management Notation in Modelling and Simulation of Multimodal Logistics Chain in Proceedings 20th European Conference on Modelling and Simulation ECMS, 2006

[6] Felix Böse, Katja Windt, Business Process Modelling of Autonomously Controlled Production Systems - Understanding Autonomous Cooperation and Control in Logistics – The Impact on Management,Information and Communication and Material Flow. Springer, Berlin, 2007

[7] Marcello La Rosa, Stephan Clemens and Arthur ter Hofstede, The Order Fulfillment Process Model, Modern Business Process Automation, Springer, 2010

[8] Pedro Ferreira, Ricardo Martinho, Dulce Domingos, IoT-aware business processes for logistics: limitations of current approaches - INForum, 2010

[9] Hlaoui, Y.B. Benayed, L.J. “A Model Transformation Approach Based on Homomorphic Mappings between UML Activity Diagrams and BPEL4WS Specifications of Grid Service Workflows”, Computer Software and Applications Conference Workshops (COMPSACW) – 2011

[10] D. Rossi,E. Turrini - What your next workflow language should look like, CoOrg 2006

[11] Gagne, D. Trudel, A. Time-BPMN. In Proceedings of the IEEE Conference on Commerce and Enterprise Computing (CEC), pages 361–367. IEEE Computer Society 2009

[12] Guermouche, N. and Zilio, S. D. Towards Timed Requirement Verification for Service Choreographies. In 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, page 10, Pittsburgh, 2012

[13] C. Combi, M. Gozzi, J.M. Juarez, B. Oliboni, and G. Pozzi. Conceptual modeling of temporal clinical workows. In Temporal Representation and Reasoning, 14th International Symposium on, pages 70-81, 2007.

[14] Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, Mohamed Jmaiel, A Survey on Time-aware Business Process Modeling. In Proceedings of the 15th International Conference on Enterprise Information Systems (ICEIS), ScitePress, 2013.

[15] http://en.wikipedia.org/wiki/Supply-chain_operations_reference

[16] Jan Mendling, Kerstin Gerke, Alexander Claus, Process Mining of RFID-based Supply Chains – IEEE Conference on Commerce and Enterprise Computing, Vienna 2009

[17] Popplewell, K. Harding, J. Ricardo, C. Poler, R - Enterprise Interoperability IV - Making the Internet of the Future for the Future of Enterprise, Springer, 2010

[18] Combi C, Pozzi G – Task Scheduling for a Temporal Workflow Management System - Proceedings of the Thirteenth International Symposium on Temporal Representation and Reasoning (TIME 2006)

[19] B. Leban, D. McDonald, and D. Forster. A representation for collections of temporal intervals. InAAAI86, pages 367–371, 1986GITO GMBH. 2013. *agito BPMO Community Center* [Online]. Available: https://service.agito-it.com/bpmo/community/ [Accessed 2014-01-16.

[20] OMG. Business process model and notation (BPMN) version 2.0, OMG document number dtc/2010-05-03., 2010

[21] OMG. Unified Modeling Language (UML) version 2.4.1, OMG document number formal 2011

[22] G. Keller, M. Nüttgens, A.-W. Scheer: Semantische Prozeßmodellierung auf der Grundlag Ereignisgesteuerter Prozeßketten (EPK). In Collection: "Veröffentlichungen des Instituts für Wirtschaftsinformatik". A.-W. Scheer (Hrsg.). Number 89, Saarbrücken 1992

[23] OASIS. Web Services Business Process Execution Language Version 2.0, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[24] W.M.P. van der Aalst and A.H.M. ter Hofstede: YAWL: Yet Another Workflow Language.*Information Systems*, 30(4):245-275, 2005

[25] W.M.P van der Aalst: The application of Petri nets to workflow management. Journal of circuits, systems, and computers 8.01 (1998): 21-66.

[26] A. Lanz, B. Weber, M. Reichert: Time patterns for process aware information systems Requirements Engineering June 2014, Volume 19, Issue 2, pp 113-141

Appendix A

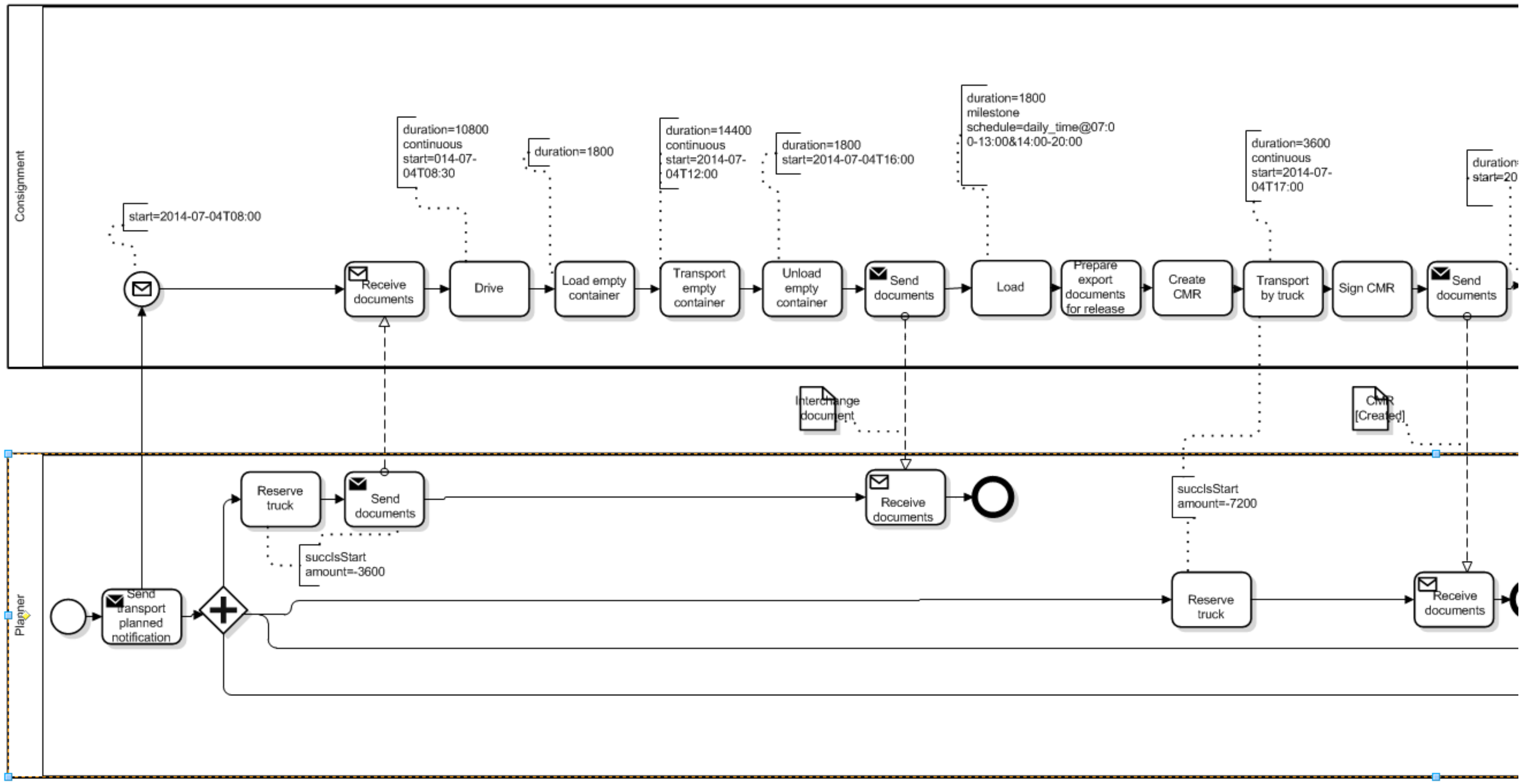
This annex collects time patterns for process models found in the literature.

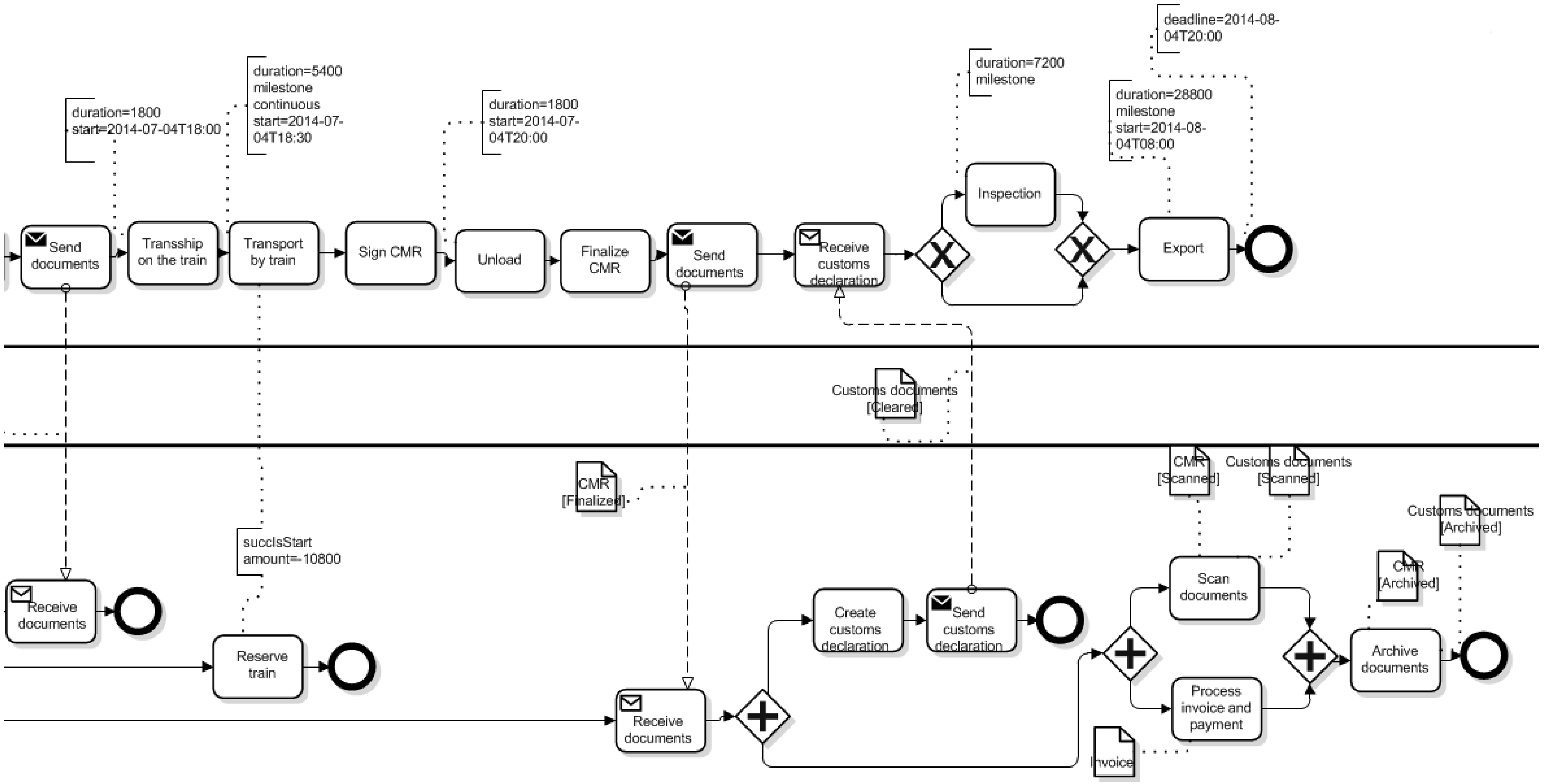
Time annotation	Definition	Example
Task duration	Stores for each task, an expected duration of execution.	The driving task "*t*" lasts 6.5 hours.
Deadline	Represents the deadline for completing all the tasks in the process.	The delivery process of the goods has to end before 01.08.2014, 12pm
Repetitive time constraint with calendar support	Represents a time constraint (that can be associated either with the tasks either with the resources) that repeats with a certain periodicity.	All drivers have a break each working day between 12pm-2pm.
Negative time constraint	Time annotation that supports the specification of blackout times.	No transports are executed on the 25th and 26th of December. Transport execution cannot start between 1am and 6am.
Schedule restricted	Time annotation that limits the execution of tasks within the time interval specified by a schedule.	Transport orders are received between 8am – 6pm. Arrival at the warehouse are permitted between 10am-5pm.
Time based restriction	Time annotation that expresses the variability of the cost incurred in performing an activity in time.	Deadline transgression costs x$ per hour.
Validity period	Time annotation that specifies the "expiration date" for the process model.	The current flow of tasks is valid only until the 1st of December 2014 when new regulations will be introduced in the enterprise.
Time dependent variability	Time annotation to specify the execution flow based on time related conditions.	If an order is received before 12pm the products will be delivered the same day otherwise the next day.
Time constraints between activities	Time annotation that specifies a duration between the end of an activity and the start of another activity.	The maximum duration between the driver receives the documents and it starts driving is of 1hr. Note: the activities don't necessarily have to be consecutive.
Time constraints between events	Time annotation that specifies a duration between the end of an event and the start of another event.	Analogous – the maximum duration between an order is received and an answer is given is of maximum 1hr.
Required delay	Time annotation that is suited for specifying inherent delays.	A planner from the European office of a logistics company needs to send some documents for approval to colleagues

		from the same company situated in Asia; in such scenario, the system has to wait for a time equivalent with the time zone gap between the Asian and the European locations before allowing the case execution to continue.
Dynamic annotation	Time annotation that is specified dynamically, based on allocated time windows for the execution of certain tasks.	Arrival pre-notification of the driver at the port set between 1pm-2pm 1.05.2014.
Resource utilization constraints	A time annotation to specify the maximum duration a resource is allowed to perform a task.	A truck driver is not allowed to drive for more than 4.5 hours continuously.

Table 1 – Time constraints for process models found in the literature

Appendix B: To-be process model for intermodal case




duration=1800
start=2014-07-04T18:00
duration=5400
milestone
continuous
start=2014-07-04T18:30
duration=1800
start=2014-07-04T20:00
duration=7200
milestone
duration=28800
milestone
start=2014-08-04T08:00
deadline=2014-08-04T20:00
Send documents
Transship on the train
Transport by train
Sign CMR
Unload
Finalize CMR
Send documents
Receive customs declaration
Inspection
Export
Customs documents [Cleared]
CMR [Finalized]
succIsStart
amount=-10800
Receive documents
Reserve train
Receive documents
Create customs declaration
Send customs declaration
CMR [Scanned]
Customs documents [Scanned]
Scan documents
Process invoice and payment
Invoice
CMR [Archived]
Customs documents [Archived]
Archive documents

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.