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

D5.5

Final Technical Specification and Phase 2 Implementation Plan for the Collaboration Manager

Project Acronym	FInest	
Project Title	Future Internet enabled optimisation of transport and logistics networks	
Project Number	285598	
Workpackage	WP5 Business Network Collaboration	
Lead Beneficiary	SAP	
Editor(s)	René Fleischhauer	SAP
Contributors(s)	René Fleischhauer	SAP
	Silvio Tschapke	SAP
	Michael Zahlmann	KN
	Cyril Alias	UDE
	Hande Koc	ARC
	Metin Turkay	KOC
	Jan Arve Hoseth	T&F

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

Reviewer	Clarissa C. Marquezan	UDE
Reviewer	Marianne Hagaseth	MRTK
Dissemination Level	Public	
Contractual Delivery Date	30th March 2013	
Actual Delivery Date	30th March 2013	
Version	V1.0	

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

Abstract

This document contains the fifth deliverable of Work Package 5. The work package is responsible for the Business Collaboration Module (BCM), which aims at the introduction of an infrastructure to manage and support the execution of collaborative business services among transport and logistics partners. In order to ensure the non-disclosure of confidential data, the BCM enables user and access management, which provides users with specific views on the data accordingly to their individual disclosure level. It also integrates information from different external sources as well as other modules of the FInest platform and makes this available for end-users of the system.

In this document we present the final technical specification of the BCM based on its first version presented in Deliverable D5.3. We also provide an implementation plan for the BCM which shows how the provided concepts and ideas can be turned into running software. The project plan for the FInest follow-up project cSpace establishes the foundation for this plan. Additionally, we provide a final version of the BCM's internal data model (based on Collaboration Objects), which has been updated in accordance with our experience from the proof-of-concept implementation (Deliverable D5.4).

With this document we address and complete T5.2 – Conceptual Design and Technical Specification and T5.5 - Phase 2 Implementation Plan of Collaboration Manager Component as defined by the Description of Work for Work Package 5. Task T5.1 and Task T5.3 are conclusively dealt with in Deliverable D5.1, D5.2 as well as D5.3. Together with Deliverable D5.4, which completes Task T5.4, all tasks of Work Package 5 are completed successfully.

Document History

Version	Date	Comments
V0.5	05-03-2013	initial version; ready for internal review
V0.6	12-03-2013	changed structure in accordance to results of Hamburg meeting
V0.9	20-03-2013	reviews integrated
V1.0	21-03-2013	final version

Table of Contents

List of Tables

List of Figures

Acronyms

Acronym	Explanation
ARH	Port of Alesund (FInest partner)
ATA	Actual Time of Arrival
ATD	Actual Time of Departure
BCM	Business Collaboration Module
CO	Collaboration Object
ETA	Estimated Time of Arrival
ETD	Estimated Time of Departure
IMO	International Maritime Organization
LSC	Logistics Service Client
LSP	Logistics Service Provider
MOF	Meta Object Facility
MRTK	MARINTEK (FInest partner)
NCL	North Sea Container Line (FInest partner)
T&F	Tyrholm & Farstad (FInest partner)
SOA	Service-oriented Architecture
TAM	Technical Architecture Modeling
OMG	Object Management Group
AIS	Automatic Identification System
GSM	Guard-Stage-Milestone

1. Introduction

The Business Collaboration Manager (BCM) is the central module within the FInest platform to observe and influence transportation processes during their execution. Users involved in such processes are enabled to see the current status of the processes at all times and can react immediately if critical situations occur. The BCM supports problem resolution by triggering alarms and sending notifications, based on user-specific notification rules, via multiple delivery channels. To provide this functionality, the BCM depends on the output of other FInest modules: The TPM is responsible for the transport planning process and creates a transport plan as a result of this process. The transport plan provides the basic data structure for the BCM and is imported after the planning process is finished. It contains all necessary (static) information about the shipment process. In order to integrate information about the progress of a transport, the BCM depends on detected situation events sent from the EPM. The events reflect a concrete situation in the real-world (such as cargo pick-up or a delay). Using these events the BCM is able to update the imported transport plan with information about the current status of the shipment. The end user is always able to see the current status of his/her shipment.

In Deliverable D5.2 we developed the conceptual design of the BCM. After a refinement process it served as the foundation for the first version of the BCM's technical specification. Additionally, we developed a novel data model concept based on Collaboration Objects in order to have an appropriate data representation within the BCM. Previously developed FInest demonstrators (*cf.* Deliverable D5.2) outline the applications that can be built using the BCM functionality in order to solve current logistics issues as described in the project use cases.

In this document we will refine the technical specification, introduced by the last deliverable (D5.3) and present its final version. We re-use the contents of D5.3 and update relevant parts. This enables us to provide a holistic and self-contained document. In Section 1.1 we provide a list of updated sections in this document. Besides the work on the technical specification, we also elaborated a proof-of-concept (POC) implementation of the BCM (*cf.* Deliverable D5.4). The POC allowed us to validate the previous version of the technical specification and discover possible improvements. We directly transferred our experience with the proof-of-concept development into the refined technical specification. We also updated the Collaboration Object-based data model based on the implementation experience and present concrete data types developed for the BCM. Finally, we present an implementation plan for the BCM during the execution of the follow up project cSpace in Phase II.

The remainder of this document is structured as follows: In the first section we describe the approach for modeling and implementing Collaboration Objects (Section 2). After that we present the final version of the technical specification (Section 3). In Section 4, we address the alignment to the FI PPP Core Platform and the potential usage of Generic Enablers. The implementation plan of the BCM is presented in Section 5 and, finally, in Section 6 we conclude the document with a short summary.

1.1. Differences from Deliverable D5.3

In the previous Deliverable D5.3 – Initial Technical Specification of Collaboration Manager Component, we introduced the initial version of the technical specification of the BCM based on its conceptual design. In this Deliverable D5.5 - Final Technical Specification and Phase 2

Implementation Plan for the Collaboration Manager Component, we present a refined and final version of this. Additionally we also provide an implementation plan for Phase II.

In order to provide a self-contained document, we use the same structure as for D5.3 for this document. In order to provide a holistic view on the provided information, the document also encompasses information that was not changed in comparison to the last Deliverable D5.3. In order to prevent repetition of information we show in the following list an overview about the updated parts of this document:

- Section 2:
  - Section 2.2: Updates on the modelling approach and concrete CO Types
  - Section 2.3: Updates on the meta-model incorporated; additionally implementation details are provided
  - Section 2.4: New
- Section 3:
  - Section 3.2: Technical design refined
    - Section 3.2.1: Removed since the described component is considered as obsolete
    - Section 3.2.2.: Updated interface design
    - Section 3.2.4: Updated design and purpose
- Section 5: New

1.2. Objectives of the Deliverable

In this document we present the conducted work towards the achievements of the following objectives:

- Objective 1: Provide a conceptual design and detailed specification for the 'Collaboration Manager' component and its integration within the overall envisioned technical solution (Task T5.2)
- Objective 2: Identify the Generic Enablers required from the Core Platform, and specify the technological realization of the 'Collaboration Manager' on top of this (Task T5.3)
- Objective 3: Develop a conceptual prototype for demonstrating the planned features (Task T5.4)
- Objective 4: Define a detailed Implementation Plan for the follow-up project in phase 2 (Task T5.5)

2. Modeling of Collaboration Objects

In this chapter, we describe the approach for Collaboration Object (CO) meta-model and present the meta-model afterwards. As described in the previous deliverables D5.1 and D5.2, Collaboration Objects are the central modeling entity of the Business Collaboration Module (BCM). Instances of these objects will be initialized with transport plan information originating from the Transport Planning Module (TPM) and supplemented by process information derived

from events received from the Event Processing Module (EPM). Through the combination of data and process information, the user is enabled to have a complete view on the current status of the transport process, with all its details.

During our design work in the past 6 months, we discovered that it might be hard to reflect all possible logistics process implementations in only one strict model. Feedback from our domain partners emphasized this observation and led us to a more novel approach of designing a meta-model for Collaboration Objects and the underlying entity-centric modeling approach. In a first section we give a precise overview about the background of our decision and describe our approach on a conceptual basis. In a subsequent section we will present the meta-model using a UML class diagram.

2.1. Decomposition of Collaboration Objects

A Collaboration Object (CO) encapsulates collaborative tasks and therefore includes all information required to reach an operational goal. The knowledge base of a CO evolves while receiving and producing data. A logistics process thus is driven by the interaction of stakeholders with COs. The loosely coupled nature of an entity-centric design provides a certain degree of flexibility. We introduce the concept of Configurable Collaboration Objects to provide a higher level of adaptability at design time and at runtime. First, we recapitulate main characteristics of the business artifacts introduced by Hull et al. in [1] to identify aspects we want to improve in order to meet our requirements. Second, we explain the theoretical background of variability in software design and identify the parts of Collaboration Objects that we want to be variable.

2.1.1. Motivation

As presented in the previous deliverables, Configurable Collaboration Objects are based on the approach of Business Artifacts (also called Business Entity) [1] and on the Guard-Stage-Milestone meta-model [2–4]. The main objective thus is to intuitively represent business processes by core artifacts including both data and life-cycle information [5]. A Collaboration Object, like any conventional business artifact, is a central place for all kind of information needed in completing a specific business goal. This allows monitoring, managing and controlling business operations. One example of a business artifact is a customer order. After being created, different stakeholders may execute tasks on the customer order instance, depending on certain conditions regarding the data or life-cycle status. Summarizing, the focus is not on complying with a pre-defined sequence of activities, but on the data and tasks incorporated within an artifact. Now we examine how to increase the level of adaptability with our solution design.

The artifact-centric modeling paradigm and event-based communication between an artifact (the notion of an *artifact* within the literature correspond to a Collaboration Object; in fact a CO is an implementation of an *artifact*) and its environment provides flexibility in form of adding, deleting, or replacing several artifacts without adapting the whole business process. In this modeling approach, however, an artifact type is fixed and cannot be varied neither at deploy nor at runtime. That means each domain scenario is limited to the specified data attributes and to the

life-cycle model of the artifact type. Even for slightly different requirements regarding data types or tasks that must be supported for a specific use-case scenario, considerable effort is required to adapt to this new situation. As previously mentioned, a domain analysis showed that there are different aspects that are likely to change across transport scenarios. Each subsequent item influences the steps of the business process as well as the data model. Below, we present a non-exhaustive list of aspects of a transport process, which can have a huge impact at the necessary process steps to implement it:

1. Transport means
    a. Air
    b. Road
    c. Sea
    d. Rail
2. Cargo type
    a. Dangerous goods
    b. Perishable goods
3. Different process implementation by different stakeholders
4. Country regulations
    a. Prohibited truck transport on Sunday in Germany
    b. Import or export procedures
5. Climate differences and weather conditions

The list illustrates that there are a lot of variable parts in transport processes. Their conduction are hugely affected by the actual mean of transport, the type of cargo, different procedures of the involved stakeholders, the diversity of country-wide regulations for particular goods or different requirements caused by climate and weather conditions. Thus, there are reasons not to model business artifacts with pre-defined information models since it is very likely that later adaptions are not possible. Variability is an essential property of logistics processes and therefore of business artifacts. The domain model components rather must be prepared to effectively deal with variability to reach a flexible design and adaptive software. Variability can be defined at different levels of design, depending on whether the entire domain model, the workflow, or single implementation aspect vary and therefore must be adapted in order to support variants. The conventional artifact-centric approach allows different alternatives to deal with this kind of uncertainty.

a) Pre-define as many artifact type variants as possible and hope that one of them fit the required capabilities, which can end up in huge amount of slightly different artifact types
b) Stop execution and construct the required artifact from scratch, which requires massive efforts for redesign
c) Build artifact types, which contain all kinds of optional attributes and life-cycle manifestations and parameterize the required ones and ignore the others, which would end up in huge artifact type definitions that are no more intuitive)

According to discussions with our domain partners, the variety of possible scenarios and configurations within logistics processes is too big. It would be very hard, if not impossible, to define an artifact-centric model (CO model) consisting of a fixed set of types with pre-defined data and behavior for this domain. Thus, none of these alternatives is an acceptable alternative for our aim and our defined requirements.

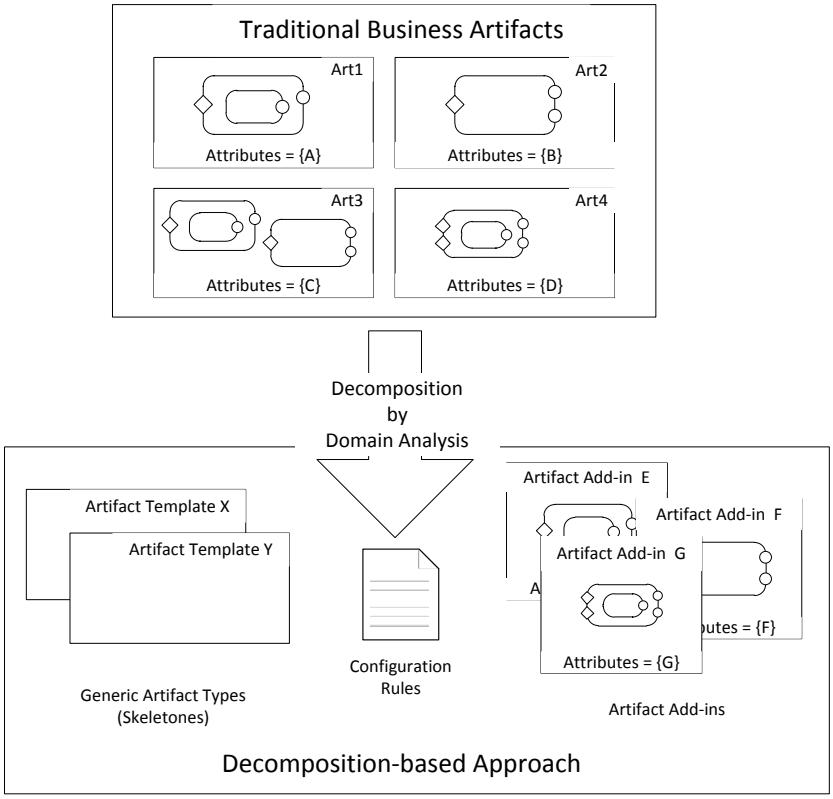


Figure 1 - Decomposition of the traditional Business Artifacts

We want to realize variability modeling to support the development and reuse of variable business artifacts. This is realized by a decomposition of traditional Business Artifacts and Figure 1 illustrates this idea. Instead of building an artifact system based on pre-defined artifact types, we want artifacts to be configurable, depending of what is needed in the actual situation. Thus, the first thing is a decomposition of artifact meta-model into pieces that are adequate. We introduced variability at different abstraction levels.

Common and variable features of a business artifact must be identified and a domain analysis is the basis for this decision. First, we identified the candidate artifact types based on use-cases. By studying different use-case scenarios of the transport and logistics domain and interviewing business experts, we identified parts and steps that are common across different use-cases and the things that do vary. Common things are designed as a skeleton. Things that vary we encapsulated into add-ins. These decisions were generalized to the artifact meta-model (*cf.* Section 2.2). Finally we wanted to configure artifacts dynamically. The user can support variants of an artifact which are living in an artifact system. With this, this workflow is not fixed to a predefined process- or data-model of an artifact type, but it can be construct, depending on the requirements, which gives freedom and flexibility for building process models.

Employing a modular design has many benefits. First of all it enables us to select and include these process steps and data which are actually needed for a specific transport. So, we do not have to deal with generic process description and can enable the user to get a precise overview of the current transport. Additionally, there are also other more technical benefits: Small chunks of code are easier to handle than whole, monolithic systems or complex objects. This holds for testing, error handling, and security issues, and has positive effects on the scalability.

Functionality can be easily added or removed without the need of adapting the whole system. As a result, extensions can even be developed independently, and can be used in different contexts, or like in our case in different Collaboration Objects. Assembling those modular components to form different variants of artifacts or whole business processes saves time and money. This allows covering a broader spectrum of potential variations, which is exactly what we need.

Although a modular design provides a number of advantages it also has its challenges. In general, the complexity and the effort for an initial implementation increases with the desired level of flexibility. During design phase, various aspects have to be considered. First, it must be decided which level of the modular design should be employed. The goal is to achieve the desired flexibility while keeping the variation space manageable. If the component granularity is chosen too low-level, one can gain flexibility, however the number of possible variations becomes too big to be handled efficiently. The other extreme is to choose the granularity too high-level and abstract. Such a decision would result in a design where the level of required flexibility cannot be reached. Second, the modular component types must be identified to get the maximum benefit and to meet the domain requirements. This requires in the first place certain knowledge on the application domain. Domain experts can support the design process in identifying common aspects and parts that change within the problem domain. Third, a modular design might suffer from inconsistencies at runtime. Thus consistency requirements must be ensured already during the design process. For complex systems, composition rules or even composition programs must be defined to ensure valid configurations. This requires a formal configuration model or even a formal language to describe components and its interfaces and a composition process. As a consequence of these challenges modularization should not be taken to the extreme, although a modular design has high potential to make a system more efficient and adaptable.

2.1.2. Template-and-Hook Concept

To provide a theoretical background and a formal foundation for the variable design model, this section will introduce a terminology to describe the envisioned concept for variability more in detail. In a first step, the terms variability and extensibility and facets that must be considered are explained.

We want to realize variability that is common in the logistics domain. In [6] the author, Pree, introduces a template-and-hook (T&H) concept for describing communality and variability knowledge. This concept helps to describe the “hot-spots” of flexibility that allow variation and extension. Pree uses the concept for describing design patterns on the level of functions and classes. We use this terminology to explain the generic concept of separating fixed from variable parts. We will describe the properties of this T&H concept.

A component ready for variation consists of parts that are fixed (frozen spots) and parts that can change (hot spots). These parts are described as role types named template and hook. Template gives the skeleton of the component. The qualified hook can be exchanged. It represents the flexible, the variable part. It grasps variability and extensibility. Figure 2 illustrates the concept:

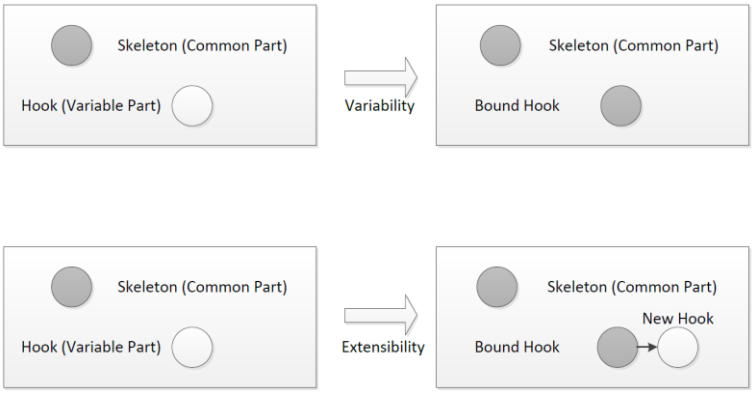


Figure 2 - Variability and Extensibility with Template-and-Hook concept

Variation means binding a hook to a component, extension means extending a hook of a component. Binding a hook with a fragment we define as a composition or configuration of this component. There are several facets of extension and variation which have to be decided during the design process:

- Granularity: It must be decided what is the subject for variation and what is the object for variation, all in all, the components for composition. This might range from methods, modules, or services; over aspects representing cross-cutting concerns or features; up to systems and frameworks. A method or a module might implement an algorithm to describe how-to handle a deviation during a transport. Services are encapsulated within an artifact to represent internal or external tasks, for example custom review or booking activities. Examples of cross-cutting aspects are monitoring or logging, that are required for all business artifacts.
- Contract between components: Describes the relation between components and describes their interfaces. They can be implicit or explicit. Are the components mandatory, alternatives, or optional dependencies? On the level of features, there are feature diagrams to express this for example.
- Binding time: Defines the time when the binding is made. It is decided which hook shall be bound to which template. The goal is to postpone the configuration as far as possible to be able to react on changes. Of course the optimal solution would be a system, which provides variability as well as extensibility at runtime. However, it always should be well considered how much flexibility you want; there is a trade-off which we have described in the previous section (*cf.* Section 2.1.1).

After presenting the different facets of extension and variation, we will now map these to our approach of Configurable Collaboration Objects.

Granularity: We want to provide dynamic extensibility on the level of Collaboration Objects. This enables us to react to new circumstances by extending a common functionality with new fragments at runtime. The idea is to provide a core artifact, which is dynamically extendable with functionality and abilities. The first step must be to identify changing parts. In our discussion with our domain partners we found out that there is a set of common steps for the most logistics processes. However not each step is required in each use-case. Regarding to the GSM - The Guard-Stage-Milestone meta-model was introduced by [7] and was already described in Deliverable D5.2 - model these are stages or groups of stages which represent a core step in the life-cycle of a Collaboration Object. We want to create a core component with

interfaces, which fits for different use-cases and a priori unknown extensions. Each extension encapsulates a specific part of a life-cycle and the relevant data model and can be handled in a uniform manner. At runtime a, theoretically, infinite number of extensions can be bound at this core, depending on the requirements a CO must fulfill.

Contract between components: Explicit extension points have to be provided, where extensions can be registered. However, algorithms itself have not to be marked as extensible. With this design the wiring of Collaboration Objects does not have to be done at design time in order to achieve specific behavior. Which extensions are required and bound to the core component can be determined during runtime. These hooks declare the composition interface of optional features. Since the internal behavior of GSM is event-based, there is no need to specify extension points in the core neither in the extensions itself.

Binding time: In a modular designs try to delay the binding time as late as possible. We will bind several extensions at design time. The information about the requirements must be included within a recipe.

Figure 3 and Figure 4 illustrate template and hooks before, and after configuration. The idea is to bind the features to a core, and afterwards this should be transparent. The result will always be a Configurable Collaboration Object, instead of binding a template.

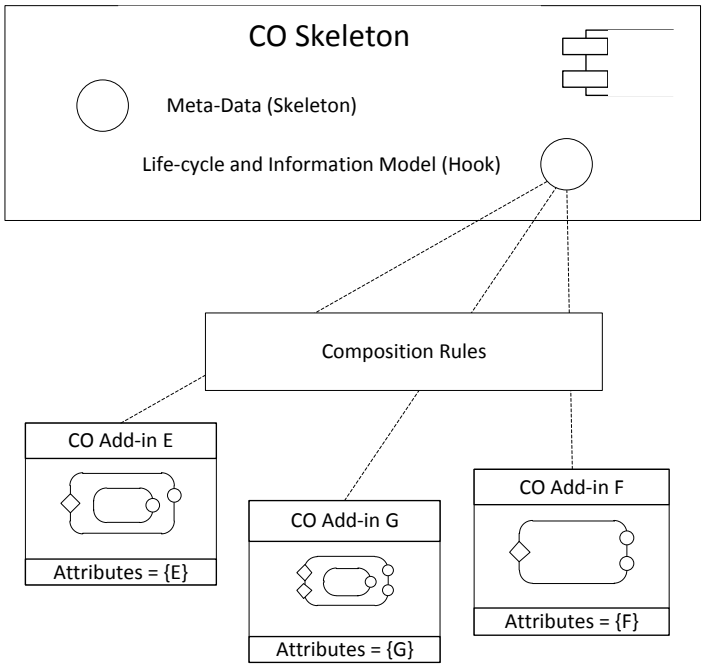


Figure 3 - Collaboration Object (before configuration)

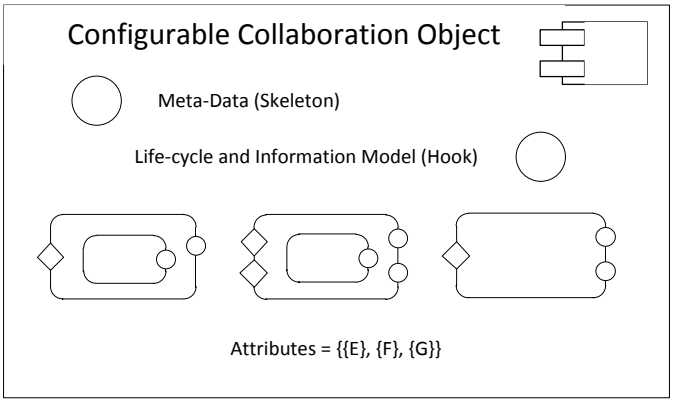


Figure 4 - Collaboration Artifact (after configuration)

2.2. Collaboration Object Modeling Example

After presenting the conceptual approach of Configurable Collaboration Objects in the previous section, we now define the Collaboration Object meta-model. This also illustrates the previously described approach. As example scenario we chose Use case 1 – Fish Transport from Norway to Brazil (for details please refer Deliverable D2.2 and D2.3), because this use case scenario was the one with richer information at the point in time that we in WP5 needed to fix the example. After a brief introduction in the scenario and a presentation of the structure of the transport plan documents, we continue with an explanation how the initial set of Collaboration Objects Skeletons and Add-ins were derived.

2.2.1. Introduction

The overall transport plan consists of a set of different TCP and TEP documents. Before we start with the further explanation we give a short description about the TEP and TCP documents (refer Deliverable D7.2, D7.3 and D7.5 for further details).

A Transport Execution Plan (TEP) is a model which holds all information related to the execution of a transport service, whereby exactly one Logistics Service Client (LSC) and a one Logistics Service Provider (LSP) are involved. A Transport Chain Plan (TCP) represents a logistics chain, and is simply a collection of TEPs. Although an implicit sequence of steps exists, the contained TEPs can be executed concurrently. The TEP information model is specified in context of the e-Freight project[1] [8]. e-Freight is a research and development project (2010-2013), which aims to support the development of a standardized framework for real-time freight information exchange covering all transport means.

[1] http://www.efreightproject.eu/

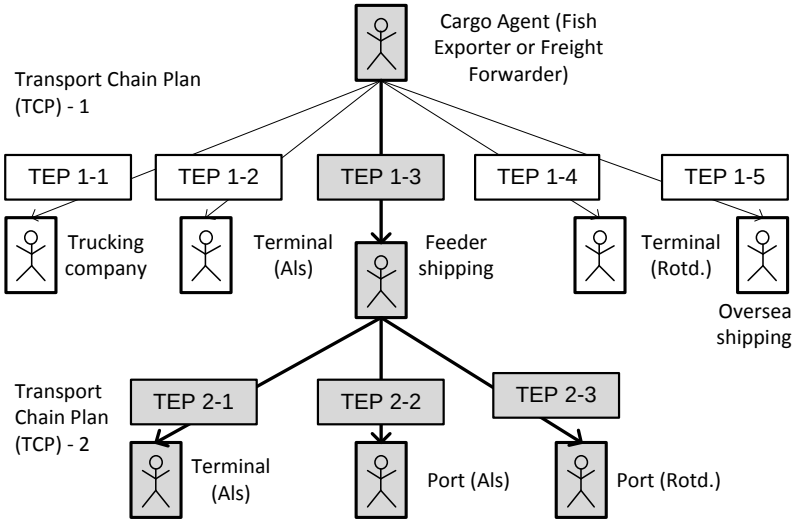


Figure 5 - Transport Plan structure for Use case 1 (Fish transport)

The subsequent scenario description is based on the corresponding transport plan for the fish export. The plan, illustrated in Figure 5, represents the hierarchical structure of TEPs and relations of stakeholders. Each TEP thereby realizes one part of the overall transport process. The top-level (TCP1) agent at the root organizes the shipments in order to get the goods from the manufacturer to the final point of distribution. This cargo agent, or freight forwarder, is an expert in supply chain management. It is in his responsibility to book carrier types for road and sea transport. The upper layer of TEPs describes the service agreements between the cargo agent and service providers. In a first step, the fish is delivered by a trucking company at the Port of Ålesund (TEP1-1) and stuffed into containers at the terminal (TEP1-2). After the following sea transport (TEP1-3), the cargo is unloaded at the terminal in Rotterdam (TEP1-4) to prepare the fish for oversea shipping (TEP1-5). However, not all transport services are under direct responsibility of the cargo agent. A service provider also can sub-contract third-party service providers. An example for this is provided by TEP1-3, which realizes the feedering by booking services realized by other TEPs (TCP-2). Feedering corresponds to the transshipment of goods or containers to an intermediate destination to combine small shipments into large oversea shipments. Thus, a feeder shipping line does the shipment from Ålesund to Rotterdam, where the cargo is transferred into bigger deep sea vessels. In the following, we focus on this feedering part, which is highlighted in the figure.

2.2.2. Collaboration Object Types

As introduced above, the Collaboration Object model is based on TEP and TCP documents, which are emitted by the TPM module after the (re-)planning process and a prerequisite of the transport execution control performed by the BCM. Alignment of the internal data model of the BCM with the data contained in these plans eases data import and alignment between the different modules. For this reason we introduced the TransportExecutionPlan Collaboration Object (short: TEP CO) with the last deliverable. Due to the fact that a TCP is a management structure and will not carry any process information during the transport execution we did not introduce a TransportChainPlan Collaboration Object.

A TEP CO contains all information provided by the TEP and adds the ability to include process information. This is in accordance with the Business Entities approach, which combines similar data and process information in a single entity [1]. Collaboration Objects are based on this idea (cf. Deliverable D5.1), and we use the same notion to represent process information: the Guard-Stage-Milestone (GSM) model [2]. The following description is based on the GSM model. Please refer Deliverables D5.1 – D5.3 for further details.

So far we only described how we came to the TEP CO as the first Collaboration Object, but omitted any further description of possible sub-types or possible process information. Through our further investigation of the use cases and our work on the proof-of-concept implementation, we discovered four different sub-types of the TEP CO:

1. TransportExecutionPlanSCTransport
2. TransportExecutionPlanSCLoading
3. TransportExecutionPlanSCAgentService
4. TransportExecutionPlanSCWarehousing

Although all four sub-types are based on the same TEP document (TPM output document), we introduced them to target very specific aspects of the transportation processes of use cases 1-3. For the sake of simplicity we used those types in a fixed manner but they can also be the result of the configuration process as described in the section above, which would enable a flexible type model. In addition to this, we discovered the necessity to tie the different TEP COs of one transport together and make them all accessible through a single entity. For that reason we introduced a completely new Collaboration Object: Transport.

In the remainder of this chapter we address all of the concrete Collaboration Object types, present their details in GSM form and describe their purpose.

2.2.2.1. TransportExecutionPlanSCTransport

This CO type is intended to be used for parts of the transportation process (legs) that form the actual transport of goods from a start location to an end location. It is agnostic towards the transport means used and, therefore, can be used for land, sea or air transportation. As we have already mentioned, the data part of the type contains all information that is available in the underlying TEP document. Process wise, the instances of this type contains two stages: one *Root Stage* and one *Transportation Stage*. Every CO type contains a Root Stage, which only indicates whether the current CO instance is active or not. The activation of this stage is the prerequisite for the activation of all other stages. The Transportation Stage represents the actual goods movement. The BCM is able to indicate deviations during execution through the guards *Pickup* and *Delayed Pickup* as well as through the milestones *Drop* and *Delayed Drop*. Note that the activation and closing of the corresponding stages only tracks the process information. The exact delay times are stored in the data part of the CO instance.

In Figure 6 you can find a graphical representation of the described CO type in GSM.

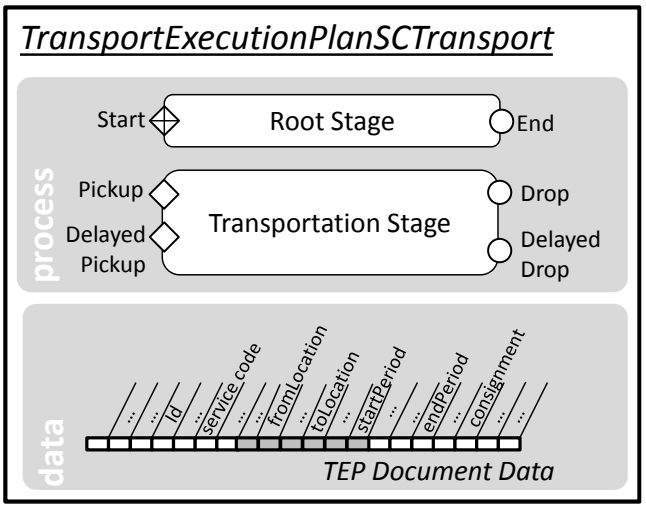


Figure 6 - GSM representation of the TransportExecutionPlanSCTransport CO Type

2.2.2.2. TransportExecutionPlanSCLoading

This CO type was introduced to represent concrete goods loading activities for transportation processes. It also encompasses the Root Stage in order to indicate the activation and end of the transportation step modeled by an instance of this type. Furthermore, it includes a *Loading List Processing Stage* and *Import License Processing Stage* in order to be able to illustrate the document processing necessary to perform a loading. The *Loading Stage* represents the actual loading of goods to all kinds of vehicles. The defined guards and milestones can be used to indicate deviations and errors during execution. A graphical overview of this CO type can be found in Figure 7.

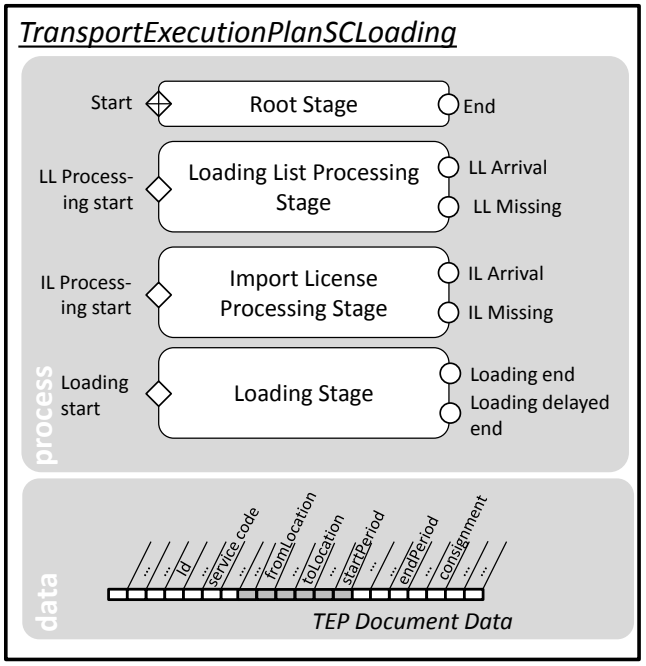


Figure 7 - GSM representation of the TransportExecutionPlanSCLoadingCO Type

2.2.2.3. TransportExecutionPlanSCAgentService

Transport plans created by the TPM can have a hierarchical (tree) structure. This is necessary due to the common approach of service providers to use sub-contractors to implement their services. In order to also have an appropriate representation in the BCM's internal data model CO type *TransportExecutionPlanSCAgentService* was introduced. It currently contains only the common Root Stage and an Agent stage to indicate the start and stop of the provided agent service. An agent's service starts with the start of the first "child CO" and ends with the last "child CO". Figure 8 shows a graphical overview of this CO type.

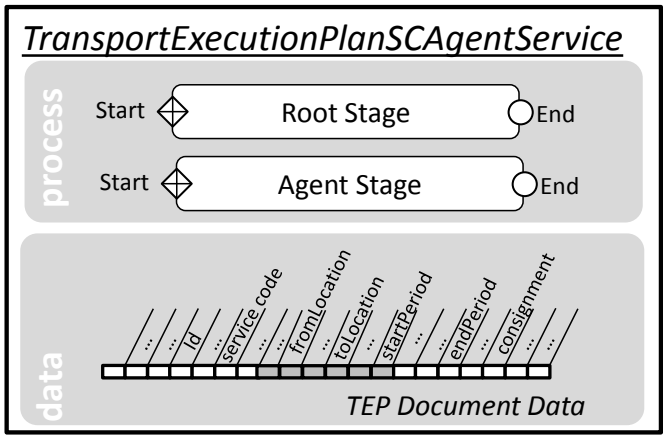


Figure 8 - GSM representation of the TransportExecutionPlanSCAgentService Type

2.2.2.4. TransportExecutionPlanSCWarehousing

This CO type was introduced to represent warehousing as an extended logistics service in the BCM data model. For example, this is necessary for use case 1 to represent the intermediate storing of fish in the terminal. Instances of the type provide, besides the standard Root Stage, one additional stage to model the actual goods storage process (when does it start and end). This stage is called the *Warehousing Stage*. In Figure 9 a graphical representation of the CO type can be found.

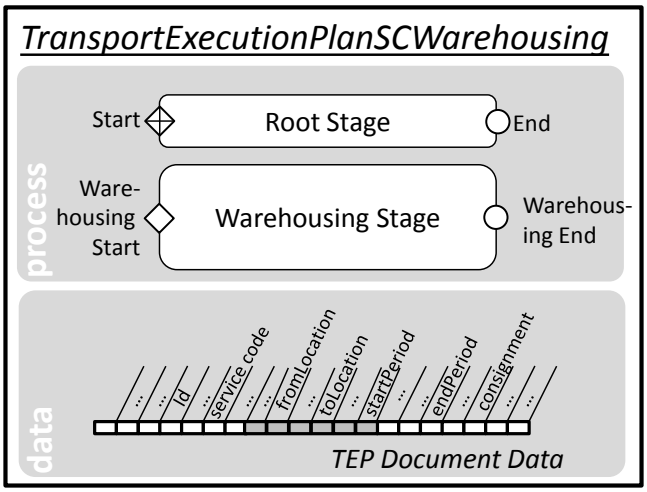


Figure 9 - GSM representation of the TransportExecutionPlanSCWarehousing Type

2.2.2.5. Transport

We introduced this CO type to have a central entity for the complete transport. Instances of this type contain some meta-information about the transport and link to all other Collaboration Object instances related to the transport (TEP CO instances). This type is not based on the data from either the TEP or TCP and is defined by the BCM. The main task of the object, besides bundling all involved Collaboration Object instances together, is to provide an overall status of the transport. The type holds the *Transport Running* stage for this reason. This stage tracks different milestones to reflect the overall transport status. Also present is the common Root Stage.

In Figure 10 we provide a graphical overview about this CO type.

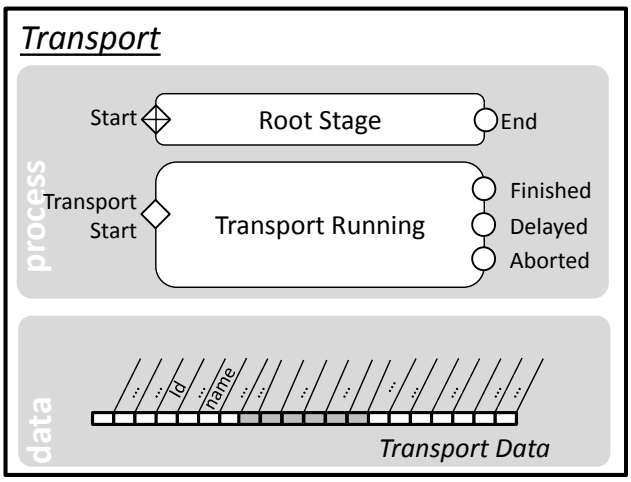


Figure 10 - GSM representation of the Transport CO Type

2.3. Collaboration Object Meta-model

In this section we want to present the used meta-model for Collaboration Objects. It is in accordance to our conceptual description in Section 2.1 and is derived from our modeling efforts based on Use case 1 (*cf.* Section 2.2). The presented meta-model reflects all the developed concepts and the proposed Skeletons and Add-ins.

In Figure 11 we present the meta-model within a UML class diagram. The package *eu.finest.bcm* is used to distinguish the classes used for the meta-model from the documents originating from the TPM (*eu.finest.tpm*) and show their dependencies.

The central class of the meta-model is *CollaborationObject*. It builds the abstract class for all derived Collaboration Objects. In accordance to the presented concept of Configurable Collaboration Objects, the subclasses have to be considered as basis for deployable COs. They only consist of the basics Add-ins valid for all use cases. Each Collaboration Object contains multiple COAddins, which add specialized chunks of data and process information.

Within the diagram the relationship between *CollaborationObject* and *COAddin* is depicted as association. Seen from a software technical perspective, the assembly of a complete Collaboration Object is thus done by object aggregation. Although, this one simple way to achieve the intended behavior, this is not meant as an exclusion criterion. The actual software

technical concept we had in mind for adding stages to COs were Mixins. Due to the fact that these constructs are not available in every programming language, object aggregation would be one possible workaround. However, there are other possibilities, such as aspect-oriented or generic programming. Within the presented meta-model we want to focus on the result: the addition of functionality to a CO.

The *TransportExecutionPlan* is a concrete subclass instance of a CollaborationObject. It represents a TransportExecutionPlan (TEP) document, which describes the bilateral relationship of a transport service provider and transport service client (see Deliverable D7.3/5 for further details). By the combination of different TEP documents a whole transport process is described, starting from point A and ending at point B. During the execution of a transport every service declared in the TEP documents is executed. Hence, the transport execution directly corresponds to the TEP structure and, therefore, the TEP document was the first candidate for a Collaboration Object.

During the proof-of-concept development it was discovered that the current model lacked a root element for a transport. Although the first TransportChainPlan (TCP) can be used as a root element, there are usually multiple such documents and the documents only serve as a management structure that defines the TEP order. For that reason, we introduced a second sub-class of CollaborationObject, the Transport class. For every transport that is managed an instance of this class is maintained by the BCM. It serves as the root element for a transport, and all related information (such as contained TransportExecutionPlan instances) is accessible through this object. Additionally, the Transport instances are also CollaborationObjects, which makes it possible to include process information in the Transport object. This allows the object to reflect the current status of the overall transport.

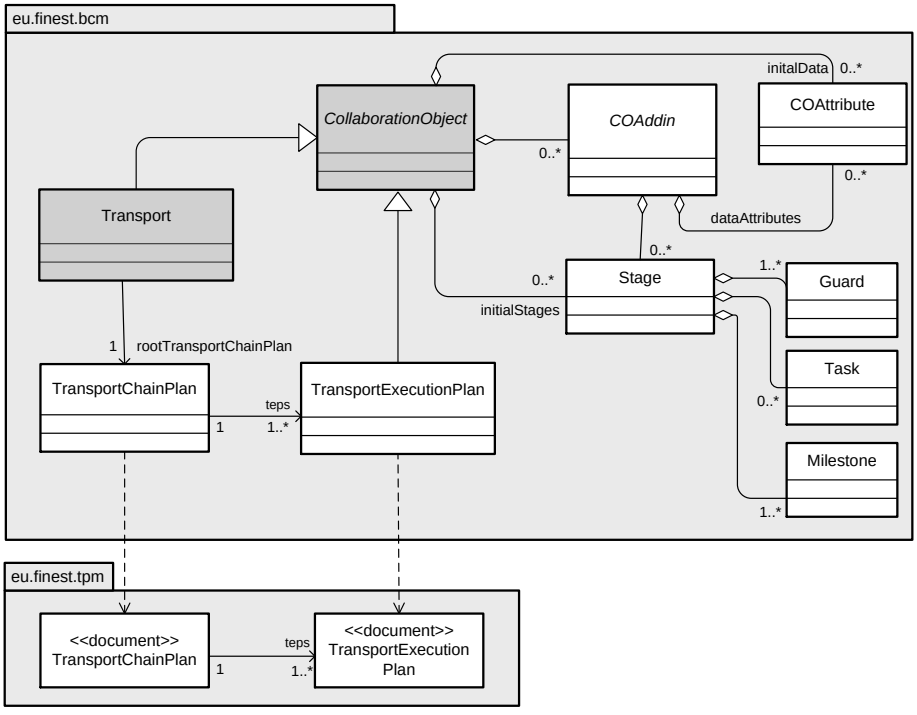


Figure 11 - UML Class Diagram of Collaboration Object Meta-Model

2.4. Data Model Implementation for Proof-of-Concept

In the previous section we discuss the meta-model for the Collaboration Object on a technical, but abstract level. The description is independent from a concrete implementation technique (design pattern, programming language or execution environment). In this section we want to give an overview of the implementation approach used for the proof-of-concept (POC).

The Collaboration Object approach is based on Business Entities introduced by IBM. This novel approach combines static data and process information in one entity (see Deliverable D5.1). Traditional execution engines for flow-based processes cannot be (directly) applied to execute Collaboration Object instances (see Deliverable D5.2). Although a suitable engine is currently under development by the ACSI project [6], it was not available at the time of development of our POCs. Due to this lack of an appropriate execution engine, we decided to develop a simplified approach to execute Collaboration Objects. For the sake of simplicity we decided to use pure Java and introduced a set of sub-classes of the TransportExecutionPlan. Each sub-class targets a particular aspect of the use cases, although these referenced TEP documents prototypical and are not fully specified. After the import of a TEP document the Composition Engine (*cf.* Section 3.2.2 of this deliverable) analyses the service code[2] of the document and decides which sub-class has to be instantiated. For example, the service code for warehousing is '20' and if the Composition Engine would instantiate TransportExecutionPlanSCWarehousing (this is the case for one of the TEPs used in the POC fish use case).

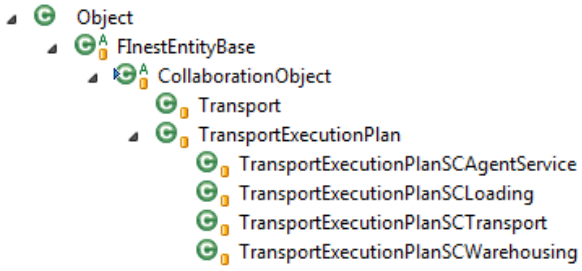


Figure 12 - Class Hierarchy of CollaborationObjects in the Proof-of-Concept Implementation

In Section 2 of Deliverable D5.3, we describe an approach to varying Collaboration Objects by adding COAddins to the instances. This approach allows us to tackle the high degree of diversity in the transport and logistics domain without having a fixed and hard-coded data model. In the proof-of-concept implementation we use a similar approach, but on simplified level. We reduced the COAddin so that it only contains a stage and introduce a set of concrete sub-classes for Stages. In Figure 13 you can find the list of those sub-classes.

In order to address the example of Transport-ExecutionPlanSCWarehousing, an instance of this class is equipped with the following stage instances during the initialization done by the Composition Engine: RootStage, WarehousingStage. The RootStage indicates the start and end of the overall process represented by TEP-CO. Each TEP-CO instance has this stage. The WarehousingStage represents the warehousing part of the TEP and can be activated right after

[2] The service code defines the nature of a transportation service, such as Transportation, Warehousing, Stuffing, Storing, etc. For further details please refer Deliverable D7.3/5.

the RootStage has become active. As we already described in Deliverables D5.1, D5.2 and D5.3, the user can observe the progress of his or her transportation process by observing the active stages.

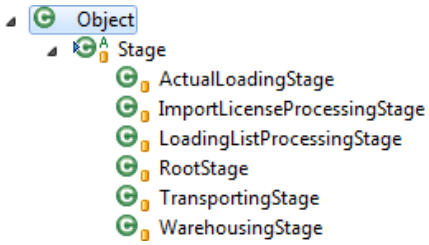


Figure 13 - Class Hierarchy of Stages

3. BCM Technical Specification

After presenting the foundations for the technical design of the BCM in the previous section, we present here the results of the design work as one of the major achievements for milestone M18. At first we describe the actual design in detail based on its graphical representation. In the following subsections, we outline the interaction of the BCM with other modules and describe how the technical design was elaborated from the conceptual design presented in the previous deliverables.

3.1. Conceptual Design of the BCM

Before we present the final technical specification, we want to give a recap of the conceptual design work for the BCM. Figure 14 shows the conceptual design from Deliverable D5.2 and in the remainder of this section we want to briefly describe the differences to the technical. We especially denote the deviations from our conceptual work and explain their reasons.

In general the technical specification differs in four major aspects from the conceptual work:

1. *Notification & Action Trigger was moved to the Front-End*

   The Notification & Action Trigger was concerned with the transparent delivery of notifications (or events, or messages) to end-users in order to trigger specific actions or decisions. During the collaboration with other technical partners, it became obvious that not only the BCM has to notify end-users about specific events. Also the EPM, TPM and ECM have to distribute their events to users. Thus, we externalized the responsible component and put it to the Front-End. By that it is now available for each module.

2. *Processing Engine was removed*

   The Processing Engine was concerned with the execution phase of a transport process. Created Collaboration Object instances shall be updated by the integration of external events. Our continued work on the concept of Collaboration Objects (*cf.* Section 2) showed that an execution engine is very tightly wired to the object instances. Thus, we decided to remove this component since all functions are now realized by the Collaboration Manager.

3. *External Data Importer was moved inside the Collaboration Object Manager*

   The External Data Importer is concerned with the integration of backend data within Collaboration Object instances. It was moved due to the fact that this data integration is done during the instantiation and configuration of Collaboration Object, which is a major task of the Collaboration Object Manager.

4. *Security & Privacy Manager was moved to the Front-End*

   The protection of Collaboration Object data was the primary purpose of the The Security & Privacy Manager. Authorization mechanisms shall be introduced so that only authorized users gain access to the data. In coordination with the FInest core module partners, we decided that this functionality is also required by the other modules and we decided that it will be centralized within the Front-end. For this reason also the BCM will use the commonly provided concepts (*cf.* Deliverable D3.3 for details)

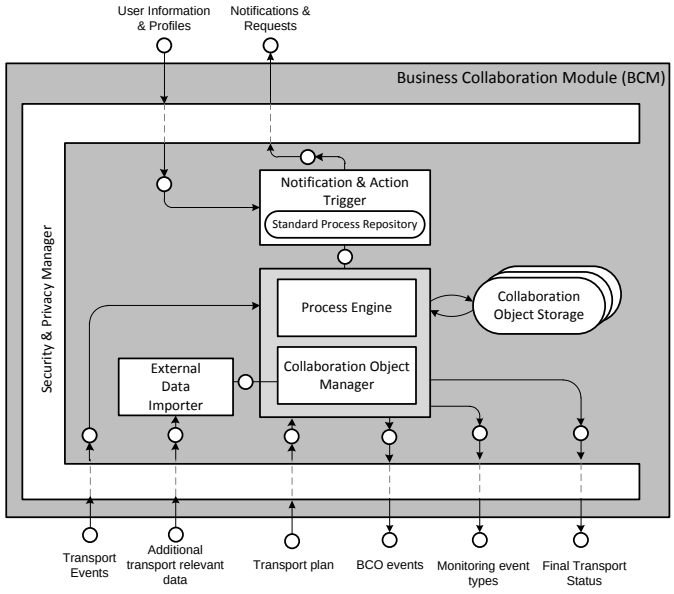


Figure 14 - BCM Conceptual Design from Deliverable D5.2

3.2. Technical Design

In order to represent the technical design of the BCM we use the Component diagram type of the Technical Architecture Modeling[3] (TAM) specification. This diagram type consists of hierarchical composed components, whereas each component can declare the provided or required functionality by defining interfaces. The diagram offers a provided and a required

[3] http://www.fmc-modeling.org/fmc-and-tam
Note: The TAM specification was also used in the Deliverables D5.1, D5.2 and D5.3.

interface notation in order to distinguished between needed and offered functionality. Besides components also artifacts are allowed, which represent static entities (e.g., documents). For more details please refer the TAM specification.

Figure 16 shows the final technical design of the Business Collaboration Manager. It shows the internal component structure of the BCM and beyond that the demanded and provided functionality to the other core modules in the FInest platform, represented as interfaces. Stereotypes are used to indicate where the component originates:

- <<FInest>>: Indicates that the component will be provided by the FInest project team
- <<FIWARE>>: Indicates that the component is provided by a FIWARE GE or a component of it

In the subsequent sections we provide a detailed description of each component, its purpose and the specification of its interfaces. This also encompasses a specification of the used data types, whereas common data types such as String or Object are omitted. The definition of interfaces and data types uses UML class diagrams and a Java-based syntax for the definition of method parameters.

3.2.1. Importer

The Importer is responsible to bring big amount of data in to the BCM. Currently there are two different use cases where this is necessary:

1. Import of an new or re-planned transport plan
2. Import of backend data

These use cases also can be seen in a temporal sequence. If the Importer receives a new transport plan, its primary aim is to invoke the creation of Collaboration Objects for this transport. For this the Importer implements the initial data preparation. Due to the service nature of each FInest component, each message from other modules - as the reception of transport plans – is realized as text-based messages. Such messages have to be converted into an internal object model in order to provide subsequent components with an easier access to the data. This process is usually denoted as ‘Deserialization’ and it is one of the first steps the Importer has to implement. Depending on the used serialization format the Importer have to use different parsers or object mappers in order to convert the received information (examples for object mappers are: JAXB for XML or Jackson for JSON). This procedure usually also include an initial consistency check, since only (syntactically) valid documents can be handled. After the Importer has successfully finished this first step it is responsible to hand the result to the Composition Engine, where the actual Collaboration Objects are created. The created Collaboration Objects are handed back to the Importer and it will deploy them within the Artifact Service Center for the execution of the transport plan.

As we have already indicated, the second use case for the Importer comes after the initial data preparation and considers the retrieval of backend data. After the Collaboration Objects are created, they can be enriched with backend data. An example for this is the import of existing documents from backend systems to the corresponding CO object. That can be done directly after the composition or later during the execution of the transport process. In any case the Importer has to do a semantically mapping between the received backend data and the corresponding Collaboration Object. With this, the Importer implements the functionality ‘*External Data Importer*’ proposed in the conceptual design of the BCM in Deliverable D5.2.

Of course, before the same Deserialization techniques as for the import of the transport plan have to be applied.

3.2.1.1. Provided Interfaces

In Figure 15 we give a detailed explanation of each method provided by the aforementioned interface.

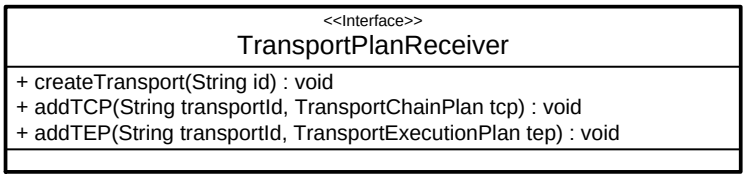


Figure 15 - Provided Interface TransportPlanReceiver of the Importer

Table 1 - Interface Methods of the Importer Interface

Method Name	Visibility	Parameters	Return Value	Description
createTransport	Public	transportId : String	Void	Indicates that the BCM shall create a new managed transport. Naturally, this would cause the creation of the corresponding Collaboration Object after the Importer has forwarded the information to the Composition Engine.
addTCP	Public	transportId : String tcp : TransportChainPlan	Void	Adds the information contained in the field tcp to a transport with the given id. The TCP document contains information about the order of different TEP documents, which represent a bilateral relationship between a service provider and a service client.
addTEP	Public	transportId : String tep : TransportExecutionPlan	Void	Adds a TEP document to the transport. This document encompasses all information about a relationship of a service client and service provider and can represent a transportation leg for example.

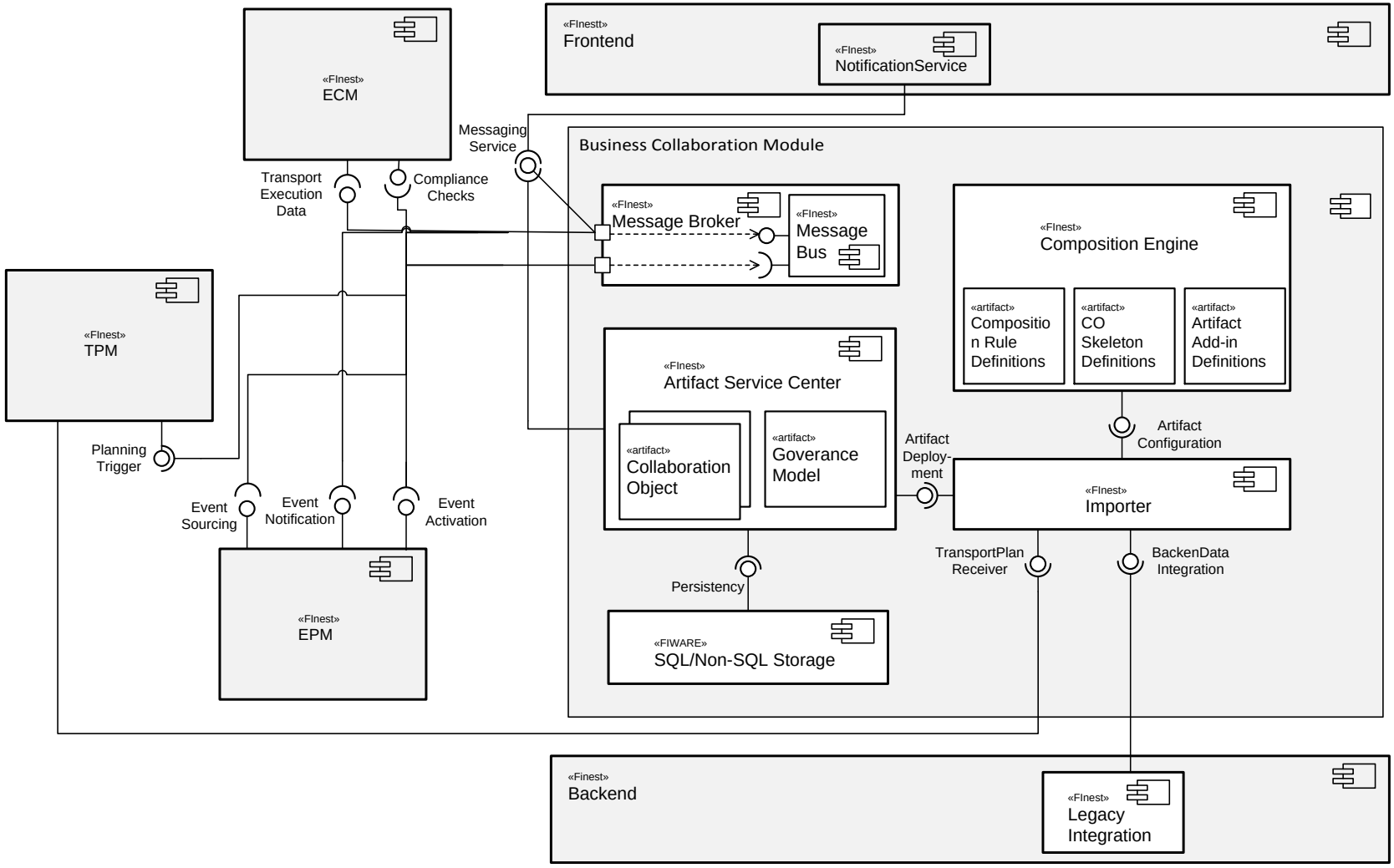


Figure 16 - Final Technical BCM Design

3.2.1.2. Used Data Types

No specific data types are used.

3.2.2. Composition Engine

The purpose of the Composition Engine is to create Collaboration Object instances for the received transport plan. After the initial data preparation is done by the Importer (*cf.* Section 3.2.1), the Composition Engine starts the analysis of the received information. As described in Section 2.2, there is no fixed type model for Collaboration Objects, from which the Composition Engine can instantiate Collaboration Object instances. Moreover, each instance is composed individually depending on the requirements defined within the transport plan. The Composition Engine uses predefined composition rules (*cf. Composition Rule Definitions* in the diagram) to map transport requirements to Collaboration Object skeletons and required Add-ins. The result is a Collaboration Object instance for a specific aspect of the transport process (e.g. a transport segment, previously represented by a TEP document). Multiple of these instances describe the entire transport process and the Composition Engine transmits them to the Importer, where they might be enriched with backend data and are deployed to the Artifact Service Center.

3.2.2.1. Provided Interfaces

In Figure 17 we give a detailed explanation of each method provided by the aforementioned interface.

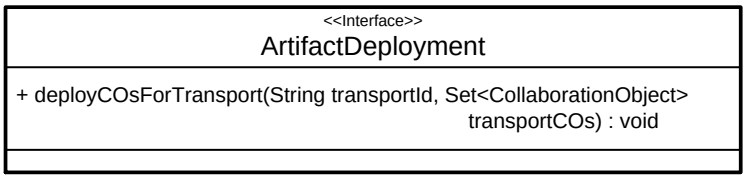


Figure 17 - Provided Interfaces of the Composition Engine

3.2.2.2. Used Data Types

Figure 18 shows the used data types of the Composition Engine component. The data type 'TCP' is omitted because it is governed by the TPM and described in Deliverable D7.3.

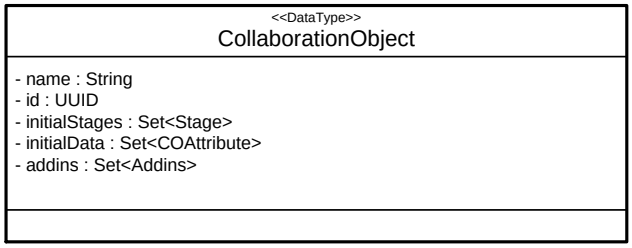


Figure 18 - Used Data Types of Composition Engine

3.2.3. Artifact Service Center

The Artifact Service Center is the central execution environment for Collaboration Objects. It receives a set of Collaboration Objects for each transport that should be tracked by the BCM. Thus, for every transport process the Artifact Service Center has to manage multiple Collaboration Objects and the Artifact Service Center usually handles multiple transport processes simultaneously. The component diagram in Figure 16 implies this by the artifacts '*Collaboration Objects*'. All information of a single transport process is encapsulated with the set of Collaboration Objects and the main task of the Artifact Service Center is to keep this information up-to-date, so that the current status of the transport process is always reflected within the data model. Governance information (e.g. the structure of the underlying transport plan) can be used to facilitate the management of changes within the Collaboration Objects. For example, a change of one Collaboration Object can also affect another. If the structure of the transport plan and therewith the structure of the Collaboration Object-based transport execution model are known, the related Collaboration Object can also be updated without an explicit event is necessary. In the diagram this governance information is represented by the artifact '*Governance Model*'

In order to update the internal model the Artifact Service Center is using events from the EPM and also from the Frontend. Both events are received over the same interface (Event Receiver), but in case of the EPM a previous configuration step is necessary. The EPM tracks events by the means for rules which are provided by the EPM. A rule is not dynamically created during runtime, but selected and started from a predefined pool. For this reason, the Artifact Service Center has to select as well as parameterize the set of relevant rules for every transport process that shall be executed. This has to be done directly after the reception of a new set of Collaboration Objects from the Importer and the Artifact Service Center component uses an interface of the EPM to transmit the information.

Due to the fact that the EPM is tracking the transport process only on the basis of event rules, it is not aware of the current business context. This means the EPM cannot decide if a certain rule has to be dis- or enabled. The Artifact Service Center has to deliver internal events – such as the status change of a Collaboration Object – to the EPM so that it can start or stop the corresponding set of rules. But also other modules and the user behind the Frontend require information about status changes within the Collaboration Objects. In order to abstract from the delivery process the Artifact Service Center uses the Message Broker component, which handles the concrete delivery of status changes.

Another important duty of the Artifact Service Center is to provide the external modules and the Frontend access to the data and status information contained within the Collaboration Objects. All information necessary for the execution of the transport is contained within a Collaboration Object which corresponds to the process. The Artifact Service Center provides an interface to enable external modules the access to Collaboration Objects by the transport process and transmits the information in a serialized format.

Transport processes are long-term business processes and processed information during these also has to be available after the finalization of the process. Hence, there has to be a suitable solution for the persistency of Collaboration Objects. The Artifact Service Center is also taking care of this and uses the SQL/Non-SQL Storage component of the Big Data GE for this.

3.2.3.1. Provided Interfaces

We summarize all provided interfaces of the Artifact Service Center in Figure 19. In the following Table 2 and Table 3 we give a detailed explanation of each method provided by the interfaces.

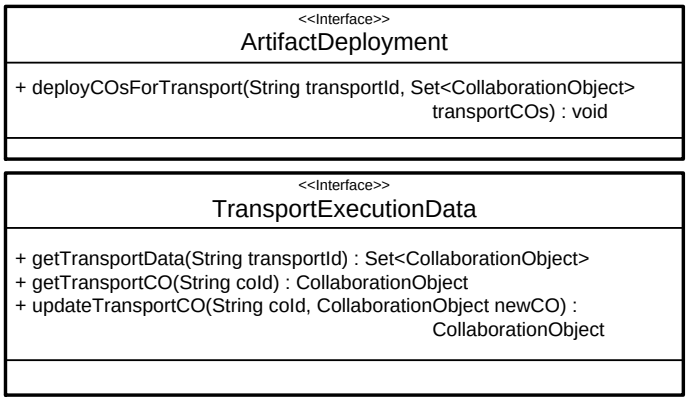


Figure 19 - Provided Interfaces of the Artifact Service Center

Table 2 - Interface Methods of the TransportExecutionData Interface

Method Name	Visibility	Parameters	Return Value	Description
deployCOsForTransport	Public	transportId: String transportCOs : Set<CollaborationObject>	List<CollaborationObject>	Used to deploy the created instances within the Artifact Service for the execution of the transport

Table 3 - Interface Methods of the ArtifactDeployment Interface

Method Name	Visibility	Parameters	Return Value	Description
getTransportData	Public	transportId : String	Set<CollaborationObject>	Gets the set of CollaborationObjects for a given transport ID
getTransportCO	Public	coId : String	CollaborationObject	Gets a specific CollaborationObject for a given Id
updateTranportCO	Public	coId : String newCO : Collaborati	CollaborationObject	Updates a specific CollaborationObject. The CO is identified by an Id and the client transmits a new CO to replace the

		onObject		old one. This can be used to add data to an existing COs

3.2.3.2. Used Data Types

Figure 20 shows the used data types of the ArtifactServiceCenter component. The CollaborationObject data type is the same as used by CompositionEngine (*cf.* Section 3.2.2.2)

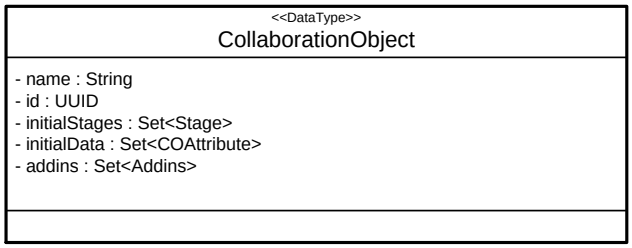


Figure 20 - Used Data Types of Composition Engine

3.2.4. Message Broker

We designed the Message Broker as a central message bus that is responsible for the distribution of incoming and outgoing messages internal to external components or vice versa. Additionally, the direct request for information is supported.

The work on the proof-of-concept implementations of the BCM (*cf.* Deliverable D5.4) showed that also the routing of inbound communication through the *Message Broker* would streamline the design and implementation of the other components. Those components can completely abstract from the communication with external clients by using the Messages Broker as a service. The rationale behind this design decision was the principal of *'separation of concerns'*, where similar tasks should be gathered in separate components.

3.2.4.1. Provided Interfaces

The Message Broker component provides interfaces to internal and external clients and directs messages via its interfaces to the internally used Message Bus. By this we illustrate that the Message Broker is responsible for transforming requests into the BCM to the internally used message format. Figure 21 below provides an overview on the externally provided interfaces.

The *TransportExecutionData* interface was formerly provided by the Artifact Service Center and is now implemented by the Message Broker. It provides methods to get access to transport process data. A client can use the interface to get all relevant information of a transport process, or parts of it. The interface also provides a method to update or delete specific transport data. A detailed description of the particular interface methods is provided in Table 4.

The MessagingService interface provides methods for clients to subscribe and publish specific message topics. By this a client is able to specify in which type of message it is interested in and receive messages if messages will get published by another client. A detailed description of the particular interface methods is provided in Table 5.

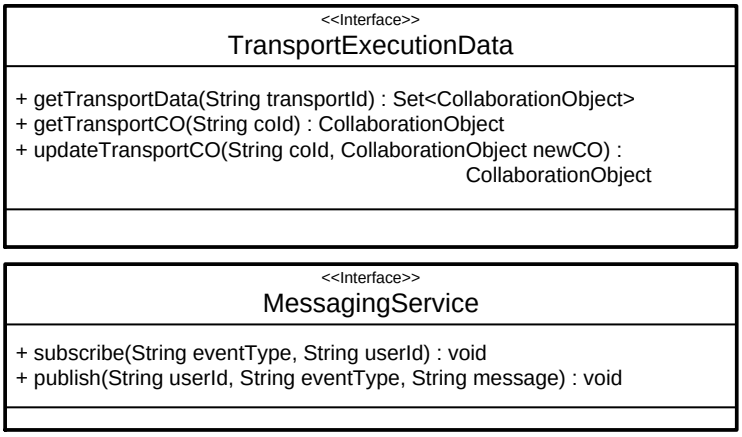


Figure 21 - Provided Interfaces by the MessageBroker Component

Table 4 - Interface Methods of the TransportExecutionData Interface

Method Name	Visibility	Parameters	Return Value	Description
getTransport Data	Public	transportId : String	Set<CollaborationObject>	Gets the set of CollaborationObjects for a given transport ID
getTransportCO	Public	coId : String	CollaborationObject	Gets a specific CollaborationObject for a given Id
updateTranport CO	Public	coId : String newCO : CollaborationObject	CollaborationObject	Updates a specific CollaborationObject. The CO is identified by an Id and the client transmits a new CO to replace the old one. This can be used to add data to an existing COs

Table 5 - Interface Methods of the MessagingService Interface

Method Name	Visibility	Parameters	Return Value	Description
subscribe	Public	eventType : String userId : String	Void	Enables user to subscribe for specific event types in order to get notified if changes occur
publish	Public	userId : String eventType :	Void	Publishes an event to a specific user.

		String, message : String		

3.2.4.2. Used Data Types

No specific data types are used.

3.2.5. SQL/Non-SQL Storage

The SQL/Non-SQL Storage is a component of the FIWARE Big Data GE. In accordance to the documentation, this component shall be used for smaller amounts of data up to 500 GB[4]. During our estimations about which amounts of data the BCM will have to handle we came to the conclusion that this boundary will fit to the very most of every use case. We took into considerations that ERP installations in midsized companies can easily exceed a database size of multiple gigabyte. That is for a single company and it can be argued that within a cloud-based scenario where hundreds or thousands of companies are managed by one platform provider, 500 GBs are exceeded. However, also the high-performance distributed file system is part of the Big Data GE and can be used if data beyond 500 GB has to be handled. For now, the usage of the SQL/Non-SQL storage requires fewer resources since no dedicated server infrastructure has to be installed.

3.2.5.1. Provided Interfaces

We summarize all provided interfaces of the MessageBroker in Figure 22. In the subsequent Table 6 we give a detailed explanation of each provided interface method.

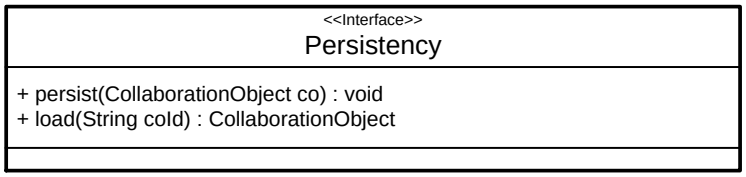


Figure 22 - Interfaces of the SQL/Non-SQL Storage Component

Table 6 - Interface Methods of the Persistency Interface

Method Name	Visibility	Parameters	Return Value	Description
Persist	Public	co : Collaborati	Void	Persists a CollaborationObject within the connected data base

[4] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FIWARE.ArchitectureDescription.Data.BigData#NoSQL_document-orientated_storage

		onObject		
Load	Public	coId : String	Void	Loads a specific CollaborationObject by a given Id

3.2.5.2. Used Data Types

Figure 23 shows the used data types of the SQL/Non-SQL Storage component. The CollaborationObject data type is the same as used by CompositionEngine (*cf.* Section 3.2.2.2)

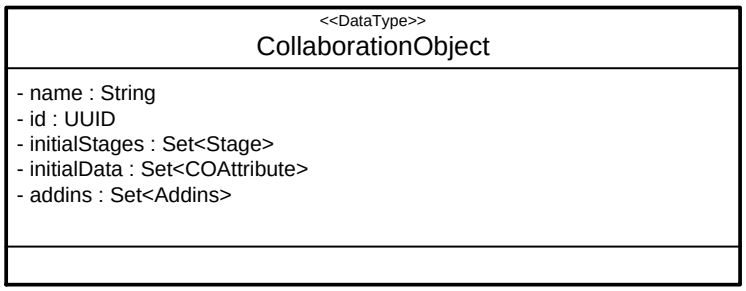


Figure 23 - Used Data Types of Composition Engine

4. Generic Enabler Usage

The evolution of the BCM design (from conceptual to technical) and the further work on the Collaboration Objects, enabled us to more precisely decide about the usage of GE from the FIWARE core platform. In Deliverable D3.1 (*cf.* Section 5) the initial set of the potentially applicable GEs for the BCM were announced. In this section we give a more detailed description of the GE usage by the BCM based on its technical specification. We denote the reasons to apply or not apply GEs which were taken into account previously.

In Figure 16 (*cf.* Section 3.2) we presented the technical design of the BCM. The figure encompasses the SQL/Non-SQL Storage component as the currently only element that relies on a FIWARE GE. The SQL/Non-SQL Storage is a part of the Big Data GE and enables the storage up 500GB, which is considered as sufficient for the BCM (*cf.* Section 3.2.5). The storage builds on top of MongoDB[5], an open source document-oriented data base. This means that the storage concept is not based on tables and their relations as in relational data bases (e.g. MySQL) but on documents. A document can be anything that is considered as a closed chunk of data. In most of the cases XML, JSON or in other way formatted data is stored in such kinds of data bases. Collaboration Objects are closed entities which encapsulate a very specific aspect of a transport process. Thus, they can be easily transformed into a serialized format (a document). In fact, this also has to be done to externalize Collaboration Objects over the service interface of the BCM. The use of a document-oriented data base is beneficial for the BCM since it has not to implement a mapping from Collaboration Object attributes to data base tables (or use object-relational mapping technology). The serialization format for Collaboration Objects can be reused and the abandonment of data mapping can support the storage performance. Furthermore, the usage of the SQL/Non-SQL storage allows using the data analyzing features

[5] http://www.mongodb.org/

for the Big Data GE (capabilities of the SAMSON Platform). This is be beneficial, if big amounts of data have to be analyzed in order to run business intelligence routines.

Deliverable D3.1 states that the issued request for a Big Data Analysis at Runtime[6] GE. The FIWARE team confirmed usefulness of this request and agreed to integrate this approach. As we have already written, we do not focus on the enablement of data analysis for the current BCM release.

Another feature request we issued was the Workflow Execution Engine GE[7]. For the time we issued the request we planned to realize the process part of a GE by the means of traditional workflow engines. However, our work on a concept to realize Collaboration Objects showed that this approach might not be applicable due to high organizational overhead. The concept behind Collaboration Objects, the entity-centric modeling is too different from traditional workflow modeling so that a realization might not be applicable.

As we discussed, the SQL/Non-SQL Storage of the Big Data GE is currently the only component that will be used by the BCM.

[6] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FInest.Epic.Data.BigDataAnalysisAtRuntime

[7] https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FInest.Epic.IoS.WorkflowExecutionEngine

5. Implementation Plan for Phase II

We give a detailed implementation plan for the BCM module. This plan is aligned to the project plan (Description of Work) of the cSpace project, which is the FInest follow-up project. In this section, we explain how the BCM component is represented in the cSpace project plan, give a list of tasks associated with the BCM, describe the expected outcomes (deliverables) and present the gantt chart with the implementation plan for the BCM in cSpace project. In addition, we outline what has to be added to the BCM components in order to fulfill the cSpace vision.

5.1. cSpace Mapping & Related Tasks

The Business Collaboration Manager is represented by the **Task 240 - Real-time B2B Collaboration** in the cSpace project plan and will be implemented within this task. This task combines the FInest modules EPM and BCM together because of their close relationship during transport execution. Sub-Task 241 - *cSpace Real-time B2B Collaboration Technical Specification* - elaborates an updated technical specification of both components due to their tight interaction. Hence this subtask is also (at least partially) associated with the BCM, whereas Sub-Task 242 - *cSpace Real-time B2B Collaboration Technical Specification* - targets only at the implementation of the EPM module. Other subtasks that are directly considered with the BCM implementation are:

- *Sub-Task 243 – B2B Collaboration Support*
  In this sub-task the B2B Collaboration component is expected to be developed based on the technical design from Phase 1 project results. This will include:
  - Collaboration process engine
  - Flexible generation of Collaboration Objects and adaptation at runtime
- *Sub-Task 244 - Collaboration and Event Management Components Interoperability*
  This sub-task is concerned with the development of interoperability components between the collaboration and the event processing components based on the technical design from Phase 1 project results. This will include:
  - Event-based rule generation from collaboration definitions
  - Impact of events on collaboration objects
  - Generating appropriate events by the collaboration component for tracking
- *Sub-Task 245: Instantiation and Execution Support*
  This sub-task provides support of instantiation and execution of collaboration objects by the real-time B2B collaboration component based on the technical design from Phase 1 project results. This is expected to include the mechanisms and tools to support interpretation and instantiation of collaboration objects

Compared to the final technical specification of the BCM (described in this deliverable) the following aspects have to be added in order to achieve the vision of the cSpace project:

- Enhance the BCM data model, to extend its capability to represent all necessary information during the execution of supply chain processes
- Extend the execution engine to execute the enhanced model instances
- Update of Collaboration Objects based on real-world events

- Fault-tolerant, scalable and robust implementation

5.2. Expected Outcome

During the implementation of Task 240 - *cSpace Real-time B2B Collaboration* - contributions to several deliverables of the cSpace project are expected. Hence, the delivery list presented in Table 7 can be considered as the expected outcome of the BCM implementation.

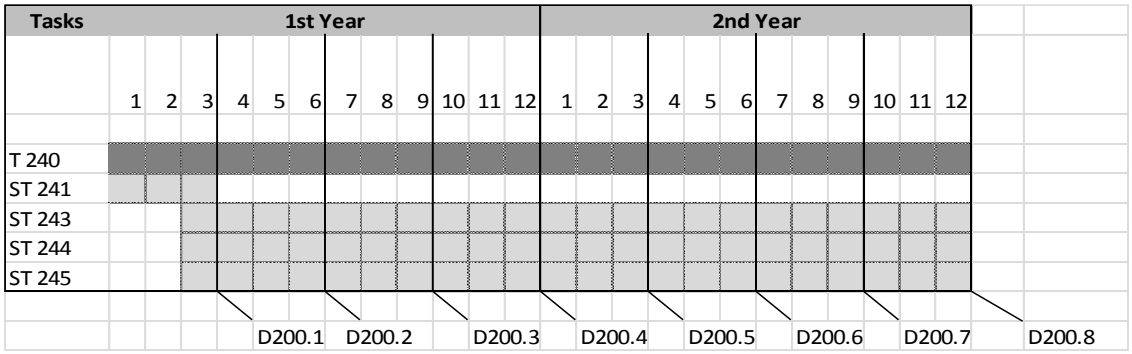
Table 7 - Expected Deliverables for Work Package 200 (incl. contributions of Task 240)

#	Name & Deliverable Number	Nature	Dissemination Level	Delivery Date
1	**D200.1** cSpace Design and Release Plan	R	PU	Month 3
2	**D200.2** cSpace Technical Architecture and Specification	R	PU	Month 6
3	**D200.3** cSpace Integrated Release V1	P	PU/PP	Month 9
4	**D200.4** cSpace Development Progress Report and V1 Updates	R/P	PU/PP	Month 12
5	**D200.5** cSpace Integrated Release V2	P	PU/PP	Month 15
6	**D200.6** cSpace Development Progress Report and V2 Updates	R/P	PU/PP	Month 18
7	**D200.7** cSpace Integrated Release V3	P	PU/PP	Month 21
8	**D200.8** cSpace Development Final Report and V3 Updates	R/P	PU/PP	Month 24

5.3. Implementation Plan Overview

In Table 8 we give an overview about the chronological sequence of the BCM implementation in a Gantt-chart. It also denotes the milestones at which contributions to deliverables are to be made.

Table 8 - Implementation Plan (Gantt chart)



6. Conclusion

This document represents the last deliverable of Work Package 5 concerned with the design and specification of the BCM within the FInest project. In it we provide the final technical specification of the BCM component. We also present an implementation plan to realize the elaborated specification, in-line with the proposal of the FInest follow-up project cSpace. The realization of the BCM is addressed by several sub-tasks of the cSpace project and all major parts are reflected within the project proposal.

After the introduction to this document (Section 1), we presented a final version of the Collaboration Object-based data model for the BCM (Section 2). We presented in the subsequent section (Section 3) the final version of the technical specification for the BCM. In Section 4, we described potential Generic Enablers to be used in the BCM implementation and in the last section (Section 5) we gave an implementation plan for the BCM based on the cSpace project plan.

7. References

[1] R. Hull, "Artifact-centric business process models: Brief survey of research results and challenges," *On the Move to Meaningful Internet Systems: OTM 2008*, vol. 5332, pp. 1152–1163, 2008.

[2] R. Hull, E. Damaggio, F. Fournier, M. Gupta, A. Nigam, P. Sukaviriya, and R. Vaculin, "Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles," *Business Entities*, no. 257593, pp. 1–22, 2010.

[3] R. Hull, E. Damaggio, R. D. Masellis, F. Fournier, M. Gupta, F. Terry, H. Iii, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. N. Sukaviriya, and R. Vaculin, "A Formal Introduction to Business Artifacts with Guard-Stage-Milestone Lifecycles," 2011.

[4] R. Hull, E. Damaggio, R. D. Masellis, F. Fournier, M. Gupta, F. Terry, H. Iii, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. N. Sukaviriya, and R. Vaculín, "Business Artifacts with Guard-Stage-Milestone Lifecycles : Managing Artifact Interactions with Conditions and Events," *Order A Journal On The Theory Of Ordered Sets And Its Applications*, 2011.

[5] D. Cohn and R. Hull, "Business Artifacts : A Data-centric Approach to Modeling Business Operations and Processes," *IEEE Data Eng. Bull.*, pp. 3–9, 2009.

[6] W. Pree, *Design Patterns for Object-Oriented Software Development*. Addison Wesley Longman, 1995.

[7] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. Heath, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and others, "Introducing the guard-stage-milestone approach for specifying business entity lifecycles," *Web Services and Formal Methods*, no. 257593, pp. 1–24, 2010.

[8] T. Cane, "Reference Solutions for Next Generation National Single Windows (e-Freight Deliverable D3.2)," 2011.

[9] K. Bhattacharya and R. Hull, "A data-centric design methodology for business processes," *Handbook of Research on Business*, pp. 1–28, 2009.