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

D5.3

Initial Technical Specification of Collaboration Manager Component

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
Reviewer	Dr. Clarissa Marquezan	UDE

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

	Åsmund Tjoar	MRTK
Dissemination Level	Public	
Contractual Delivery Date	30th September 2012	
Actual Delivery Date	30th September 2012	
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 third 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 the end-to-end networks of transport and logistics partners. It 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 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.

In this document we focus on the presentation of the elaborated technical design for the BCM. This is based on the previously presented conceptual design and insights we gained from the FInest demonstrator development process. We also present a detailed concept for the modeling of Collaboration Objects and show a preliminary version of the corresponding meta-model. At the end we briefly discuss the used Generic Enablers within the technical architecture.

Consequently, we primarily address task T5.2 – Conceptual Design and Technical Specification – as defined by the Description of Work for Work Package 5. Additionally, we present our work on Task T5.3 – Technological Alignment with the FI PPP Core Platform, while we present our investigation on suitable Generic Enables of the FIWARE Core Platform for a realization of architectural work.

Document History

Version	Date	Comments
V0.1	31-07-2012	Definition of initial ToC
V0.5	14-09-2012	Initial content finished; document ready for internal review
V1.0	27-09-2012	Comments and remarks integrated; ready for submission

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

As described in the previous deliverables, the Business Collaboration Module (BCM) is the central knowledge base during the execution of transport processes. The BCM enables users to always have an overview about the current status of their currently ongoing transports. Furthermore, the BCM notifies the user about the occurrence of certain events (e.g. a deviation, a completion of a transport or specific process step) and triggers alarms so that humans operating the system can take the proper reactions if necessary. In order to enable this, the BCM uses a novel approach to model business process, the Collaboration Objects. As previously shown, instances of these objects contain a data and a process part and a whole transport process is described by a combination of multiple objects of (potentially) different types. The initialization is done by the outcome of the Transport Planning Module (TPM), the transport plan, and during the execution phase the BCM constantly integrates information from the Event Processing Module (EPM) and from different users of the Front-End in order to keep the internal model up-to-date.

In the previous deliverable we refined the conceptual design of the BCM at hands of a set of updated domain as well as technical requirements. We developed a demonstrator (the Resource Coordination Demonstrator) in order to cross-check these requirements and sketch the envisioned functionality of the BCM. Additionally, we gave a first introduction in to the modeling of Collaboration Objects. During the last working period, we built on top of these results and pushed our work towards the technical specification of the BCM. In this deliverable we describe the result of our efforts. We present a detailed concept of the modeling approach for Collaboration Objects and show a preliminary version of the CO meta-model. We also provide the initial technical specification of the BCM with a detailed description to its internal components, interfaces and the interaction with other modules of the FInest platform. In addition to this, we provide a brief description of the used Generic Enables of the FIWARE platform.

The remainder of this document is structured as follows: In a first section we describe the modeling of Collaboration Objects and present a preliminary version of the meta-model. Afterwards, we present the technical specification of the BCM and describe its interaction with other modules. A last section briefly describe the GEs used by the BCM.

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 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 to monitor, to manage, and to control 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 for each domain scenario one 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 mentioned within the chapter

introduction, 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 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 is 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 indispensable. 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.

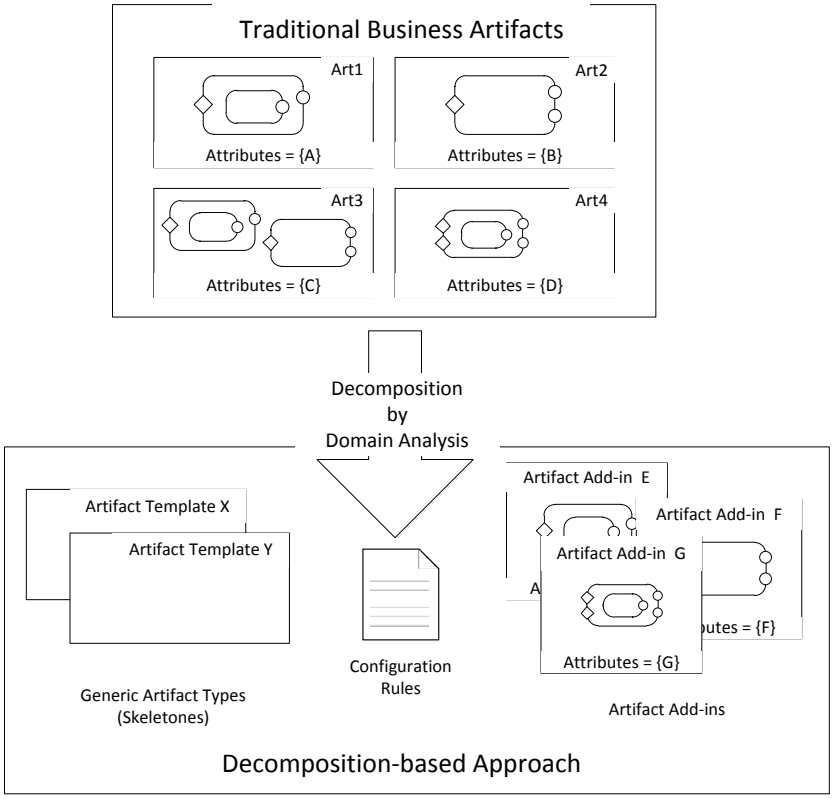

Figure 1 - Decomposition of the traditional Business Artifacts

We want to realize variability modeling to support the development and reuse of variable business artifacts. Support variants of business processes by varying a skeleton or even by extending functionality of single components. Figure 1 illustrates the 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. A domain analysis is the basis for these decisions. First, the decisions of candidate artifact types were made on basis of 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 can construct it, depending of the requirements. This means 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 a whole, monolithic systems or complex objects. This holds for testing, error handling, and security issues. This 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 increase 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:

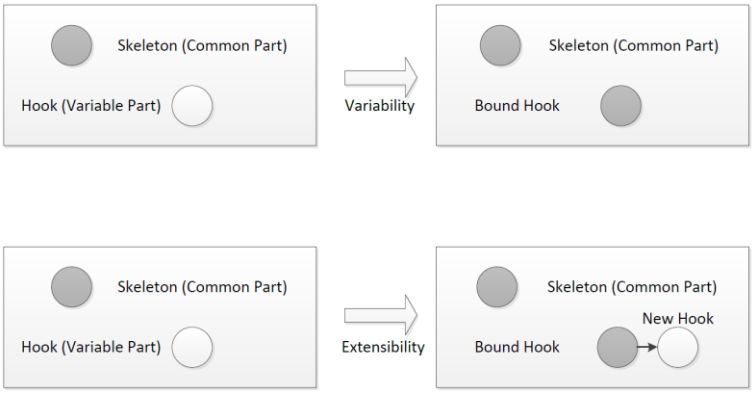

Figure 2 - Variability and Extensibility with Template-and-Hook concept

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).

In the following we will explain design decisions by means of this list.

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[1] 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 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.

Figures 3.3 and 3.4 illustrate template and hooks before, and after configuration. The idea is to bind the features to a core, and afterwards these should be transparent. The result will always be a Configurable Collaboration Object, instead of binding a template.

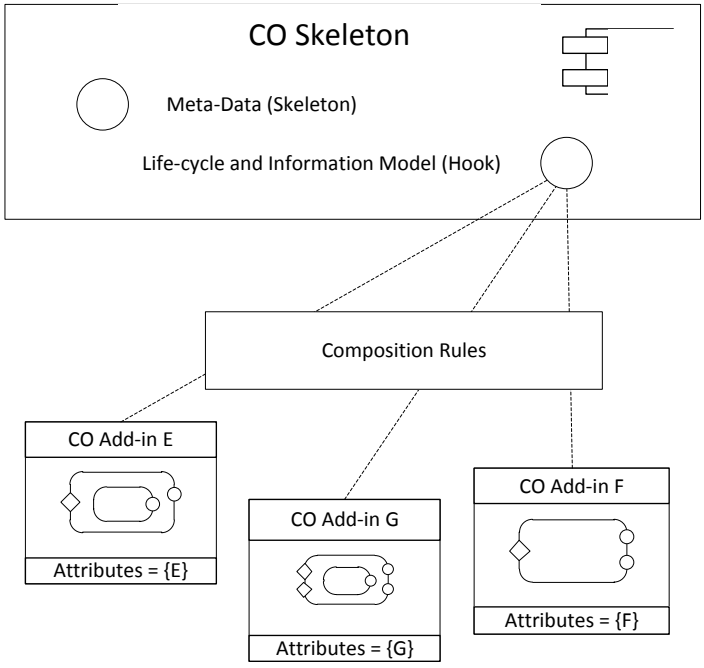

Figure 3 - Collaboration Object (before configuration)

[1] The Guard-Stage-Milestone meta-model was introduced by [7] and was already described in Deliverable D5.2

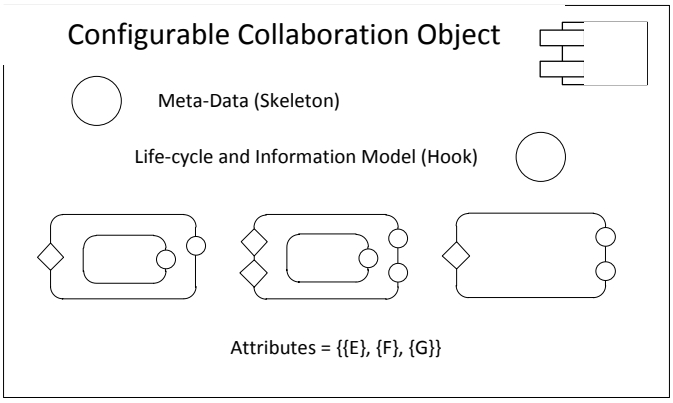

Figure 4 - Collaboration Artifact (after configuration)

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 recap of the TEP and TCP documents (refer Deliverable D7.2 and D7.3 for further details).

A Transport Execution Plan (TEP) is a model which holds all information related to the execution of a transport service, whereby at least a Logistics Service Client (LSC) and a 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[2] [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.

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

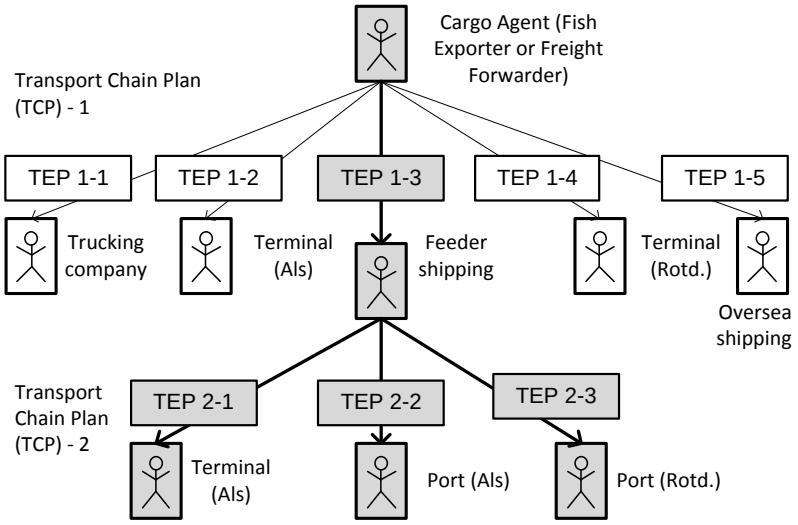

Figure 5 - Transport Plan structure for Use case 1 (Fish transport)

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 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 and stuffed into containers at the terminal. After the following sea transport, the cargo is unloaded at the terminal in Rotterdam to prepare the fish for oversea shipping. 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. 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. Example-based Description of the Collaboration Object Model Development Process

The application of the developed modeling approach to realize a concrete scenario requires an in-depth domain analysis in the first place. Domain expert feedback is helpful to define the overall business goal and to choose an appropriate granularity of artifacts and features. Based on a data-centric perspective, the identified types of Collaboration Objects are stepwise refined and extended with lifecycle information and runtime constraints. Referring to a methodology for conventional artifact-centric modeling proposed by [9], we summarize the steps for the business process design based on Collaboration Objects:

1. *Artifact Type Discovery*: discover critical collaboration artifact types and their operational goal(s)

   a. Staging: formulate artifact sub-goals to identify top-level stages. Define data attributes related to each stage
   b. Service Discovery: identify internal/external tasks for atomic stages
   c. Refinement: define Guards and Milestones with Event-Condition-Action Rules
2. *Feature Type Discovery:* Identify behavior and operational goals, which are supposed to be reused in several artifact instances
   a. Staging: Formulate stages that are part of the feature lifecycle
   b. Service Discovery: Identify internal/external tasks, which are related to the feature
   c. Refinement: define Guards and Milestones with Event-Condition-Action Rules; define points of variation
3. *Dependency Definition:* set up composition rules and dependencies between feature types

This rather abstract methodology consists of three major steps. The first step aims at a high-level specification of business operations by defining key artifacts and their most important lifecycle stages. Typically, this is done in the context of a top-down analysis by considering user stories and exceptional use-cases. Key domain entities and their top-level stages are identified to represent the entity lifecycle and the operational business goal. Subsequent to this, tasks, which are supposed to represent the artifact behavior, are associated to atomic stages. Finally, dependencies and their resulting constraints are designed at the level of guards and milestones. This lifecycle model and the corresponding data model build the skeleton for the artifact type, which is common for all instances of this type. The goal of step 2 is to identify artifact features as the reusable fragments that extend the capabilities of artifacts. Each feature thus has its own data and lifecycle model, and encapsulates functionality reused among different artifact instances. Since features are rather generic and adaptable fragments, the second step finally requires the explicit selection of feature variation points. The third step of the modeling process aims at defining the relationships among features and artifacts, which mainly consists of a dependency analysis for later consistency checking and verification procedures.

Now, we describe the design process on basis of collaboration artifacts for the fish export scenario. In a first development cycle, we identified different types of Transport Execution Plans (TEPs) as key collaboration artifacts. We distinguished between four artifact types according to different means of transport (rail, road, air and sea), as well as one artifact type to represent logistics services that do not represent any physical transport. A TEP is considered as a key domain entity, because it represents the explicit knowledge of one part of the process. Once instantiated, a TEP evolves and passes several stages on its way to the operational business goal. A Booking Stage for instance, includes tasks for verifying, updating, or canceling a booking. Within a Documenting Stage, required data such as way bills or load lists are requested, whereas a Reviewing Stage associates tasks for customs reviews. Beyond this, Stages for shipment, loading/unloading, or for monitoring services are defined within each TEP. In contrast to a TEP, a TCP is considered as a management element, which handles a collection of TEPs.

Our first approach for Collaboration Object model was to design a fixed set Collaboration Object types with pre-defined stages and attributes. However it turned out that this approach had some substantial drawbacks. In accordance to the feedback to the domain partners in the work package, we came to the conclusion that the model is too inflexible to represent all possible transport scenarios. The diversity within the domain makes it nearly impossible to create a complete model of Collaboration Objects types.

For this reason we revised the modeling approach and introduced the concept of Skeletons and Add-ins to provide a flexible definition of Collaboration Objects during runtime as described in Section 2.1). This abandoned the fixed type concept. TEP2-1 for example represents the loading at the terminal in Ålesund. The corresponding artifact must ensure that a load list is available and that the terminal is notified about the loading state. TEP2-2 in contrast, is supposed to handle a port call. Although, this TEP has some data types in common with other TEPs, it differs in the corresponding port call functionality. These differences in associated services and required data attributes are common among all TEPs. As a result, the model was restricted to one generic TEP artifact type only. This TEP artifact type thus serves as a skeleton including those data and lifecycle attributes that are common among possible instances. At runtime, this skeleton is extended with functionality with regard to specific scenario requirements.

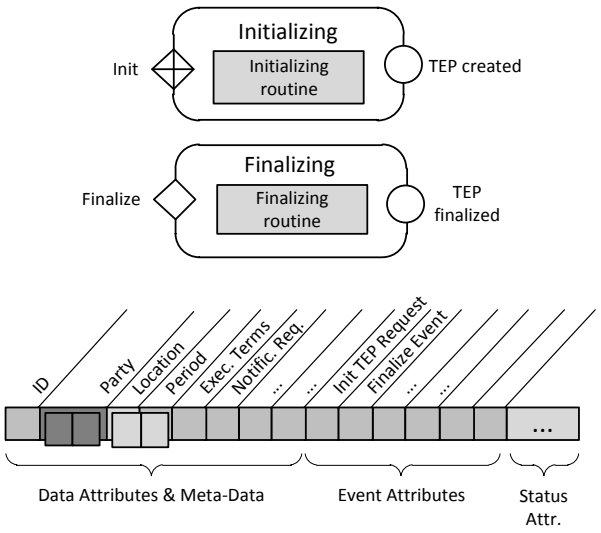

Figure 6 - TEP-CO (Skeleton)

We defined a TEP-CO skeleton which provides the basic attributes and stages of every Collaboration Object instance that represents a Transport Execution Plan. Thus, this skeleton is the same for all of the TEP-COs within the example: TEP1-3, TEP2-1 and TEP2-2. Figure 6 shows the structure of the skeleton using the GSM notation. It consists of an Initializing stage, triggered at time of instantiation, and of a Finalizing stage. Since these default stages are part of the object skeleton, they are available for all its instances. The differences between TEP instances are in the implemented services. Whereby TEP1-3 realizes the feedering transport, TEP2-1 is responsible for the loading, and TEP2-2 handles a port call. The e-Freight modeling framework therefore provides a list of service codes, identifying services such as loading/unloading, transport, stuffing, and others. Those services, as well as the identified top-level stages of the first modeling cycle are taken as a basis for feature type modeling. However, differences between instances of the same feature type must be considered. The truck loading process requires different steps and document types, than a loading of a ship.

In the following we apply these findings to the TEPs 1-3 and 2-1. The focus thereby is not on providing a detailed specification of data attributes, but it aims to illustrate the idea of Add-ins and Skeletons. TEP1-3 represents the feedering to transship goods from one point to another. The corresponding Transport Add-in, shown in Figure 7, consists of three stages: Departure, Ongoing Shipment, and Arrival.

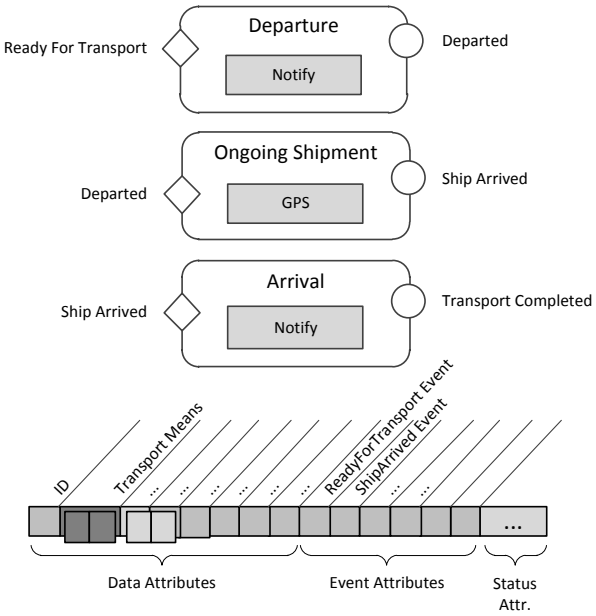

Figure 7 - Transport Add-in

To each stage a task is associated which informs registered stakeholders of the current state. If a "Ready for Transport" event is received, the Departure Stage is opened and a notification is sent to the registered users for this artifact. The task of the Ongoing Shipment Stage is to inform the stakeholders about the concrete position during the shipment. Finally, the system sends a notification when the milestone "Transport Completed" is achieved. The data attribute "Transport Means", which actually belongs to the e-Freight TEP model, is the major data attribute of the Transport Add-in. The TEP collaboration artifact type in contract only contains data attributes that are common among all possible TEP instances. Since there are TEPs which represent services without any transport means, this data attribute is removed from the TEP skeleton. TEP2-1 provides an example with two features. On the one hand, an export license must be checked by an external service or agent, and on the other hand cargo must be loaded onto the ship. The Add-ins are illustrated in 5.5 and 5.6.

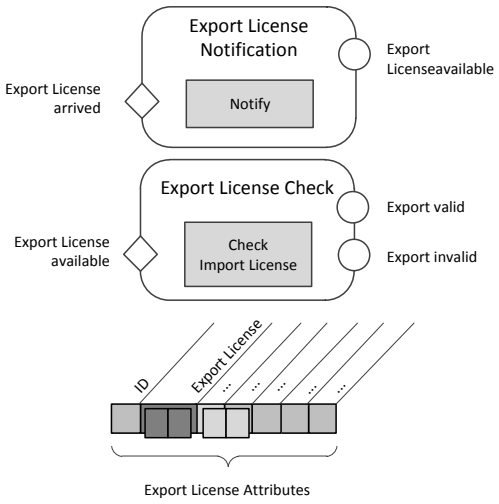

Figure 8 - Export License Add-in

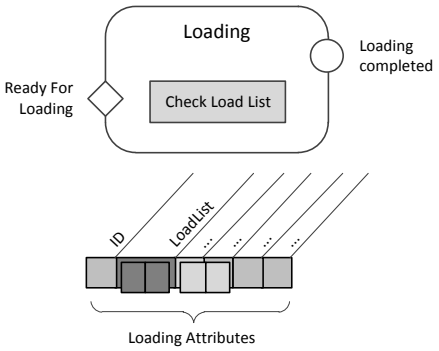

Figure 9 - Loading Add-in

The Export License Add-in contains two stages. One stage informs all registered stakeholders of an inserted Export License and another stage for license analysis. The license checking itself is an external service, accomplished by external Web services or humans, who assess the license for validity. One main data attribute hereby is the Export License document, which is specific to this Add-in. The second Add-in related to TEP2-1 is the Loading Add-in. It contains all information related to the loading of the ship. Loading events are handled, and corresponding data such as a load list is checked for inconsistencies or missing information, once the Loading Stage is opened. The related milestone finally is triggered by the external service. In a configuration step, both Add-ins are bound to the TEP skeleton. The instantiated Collaboration Object for TEP2-1 is presented in Figure 10. The remaining TEPs are modeled in the same way, until the whole transport plan is modeled by add-in configured collaboration artifacts. The transport plan (*cf.* Figure 5) thus is the input for principal Collaboration Object configuration. The plan is analyzed; related artifact types and Add-ins are instantiated and finally composed to deployable Collaboration Objects. During the execution phase of a transport process, each CO evolves as messages are received, data is inserted, or tasks are executed, until the final milestone is achieved.

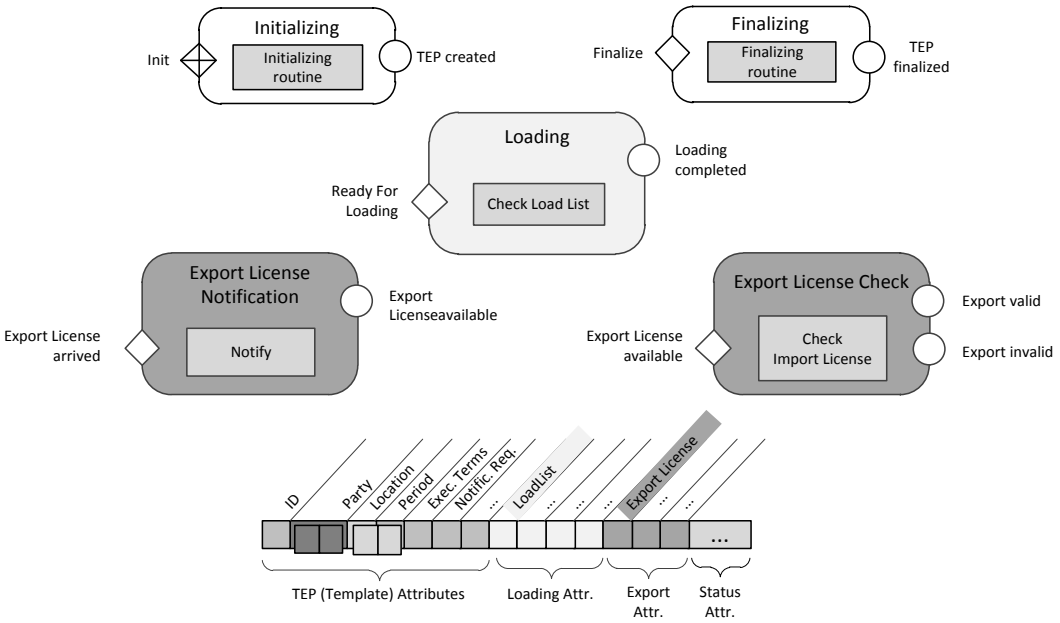

Figure 10 - Completely assembled Collaboration Object for TEP2.1 (TEP-CO)

2.3. Collaboration Object Meta-model

In this section we want to present the preliminary meta-model for Collaboration Objects. It is in accordance to our conceptual description in Section 2.1 and is derived from our first modeling efforts based on Use case 1 (*cf.* Section 2.2). The presented meta-model reflects all the developed concepts and the initially discovered Skeletons and Add-ins. We will continue our work on the model during the next work period and present an extended version with the final technical specification in M24.

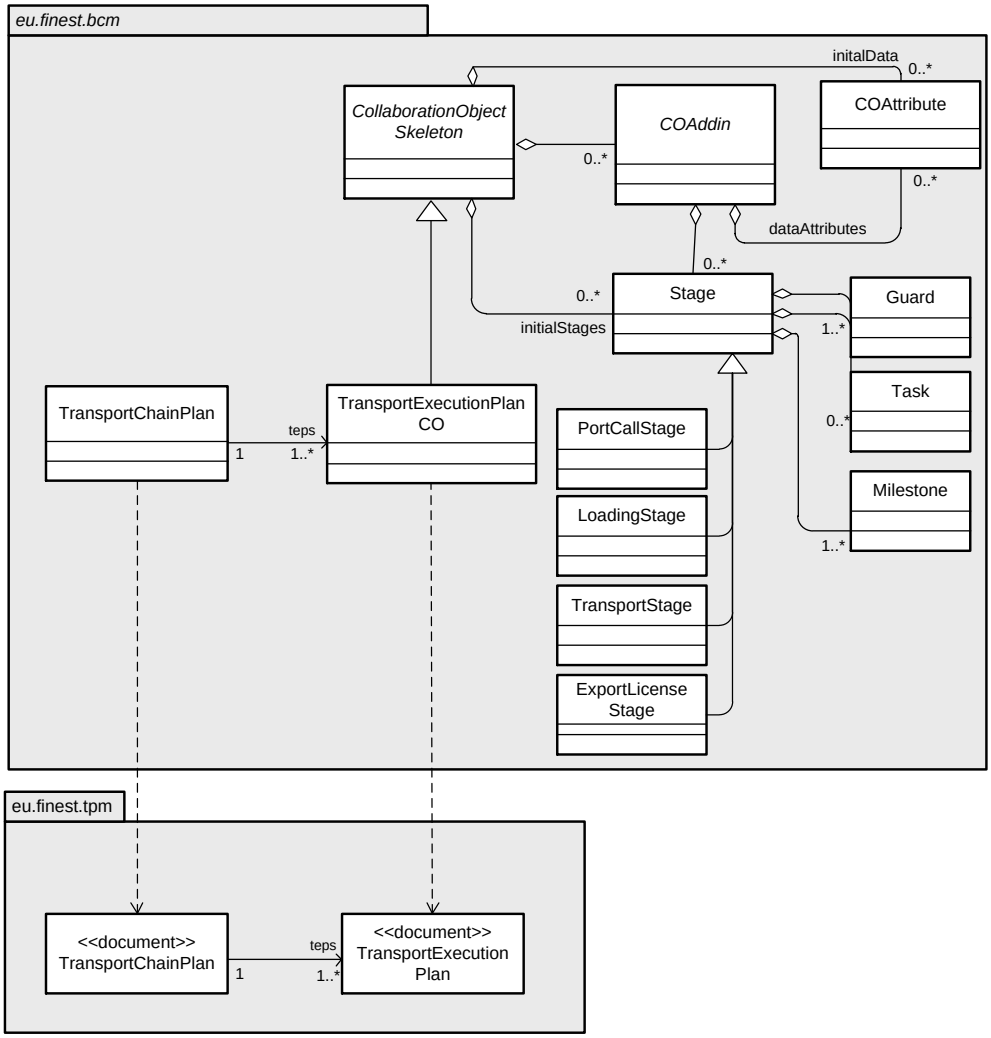

Figure 11 - Preliminary Collaboration Object Meta-model

In Figure 11 we present a preliminary version of 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 *CollaborationObjectSkeleton*. It builds the abstract class for all derived Collaboration Objects skeletons. In accordance to our presented concept of Configurable Collaboration Objects, the subclasses have to be considered as the skeletons for final the final deployable COs. They only consist of the basics Add-ins valid for all use cases. Currently there is only TransportExecutionPlanCO the only subclass, because it is our first

discovered Collaboration Object skeleton. Each Collaboration Object contains multiple COAddins, which add specialized chunks of data and process information. Currently we discovered four different stages (PortCallStage, LoadingStage, TransportStage and ExportLicenseStage), but we expect that we extend this list in the very near future.

Within the diagram the relationship between *CollaborationObjectSkeleton* 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 CO skeletons 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 skeleton object.

We also depicted within the diagram the relationship to the transport documents TCP and TEP originating from the TPM in order to depict their connection.

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 initial technical specification, we want to give a recap of the conceptual design work for the BCM. Figure 12 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)

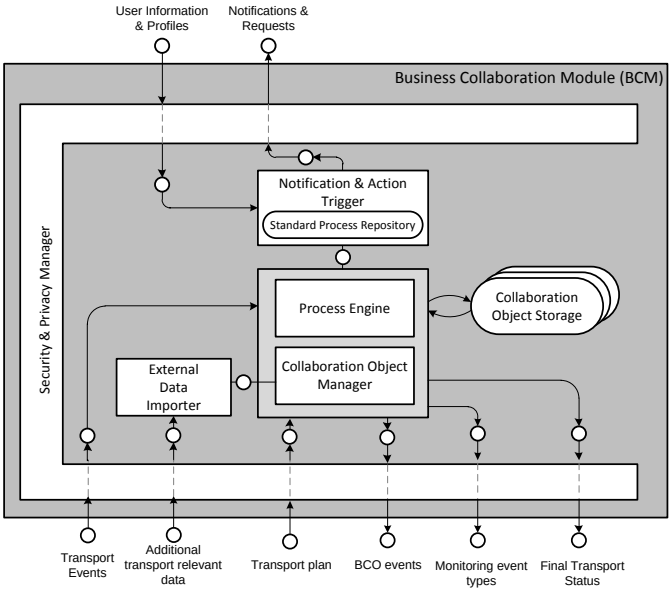

Figure 12 - BCM Conceptual Design from Deliverable D5.2

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 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 13 shows the current 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 if 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. Collaboration Object Manager

The Collaboration Object Manager is a conceptual component and will be without any technical manifestation. It was kept by us in the diagram to enable the reader to make the connection between the conceptual and technical design. In fact this component directly corresponds to the Collaboration Object Manager of the conceptual design. In order to achieve a clearer and better design we decided to distribute its envisioned functionalities in the conceptual design among different subcomponents. But with all its subcomponents it has exactly the same functionalities.

3.2.1.1. Provided Interfaces

No specific interfaces are provided.

3.2.1.2. Used Data Types

No specific data types are used.

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

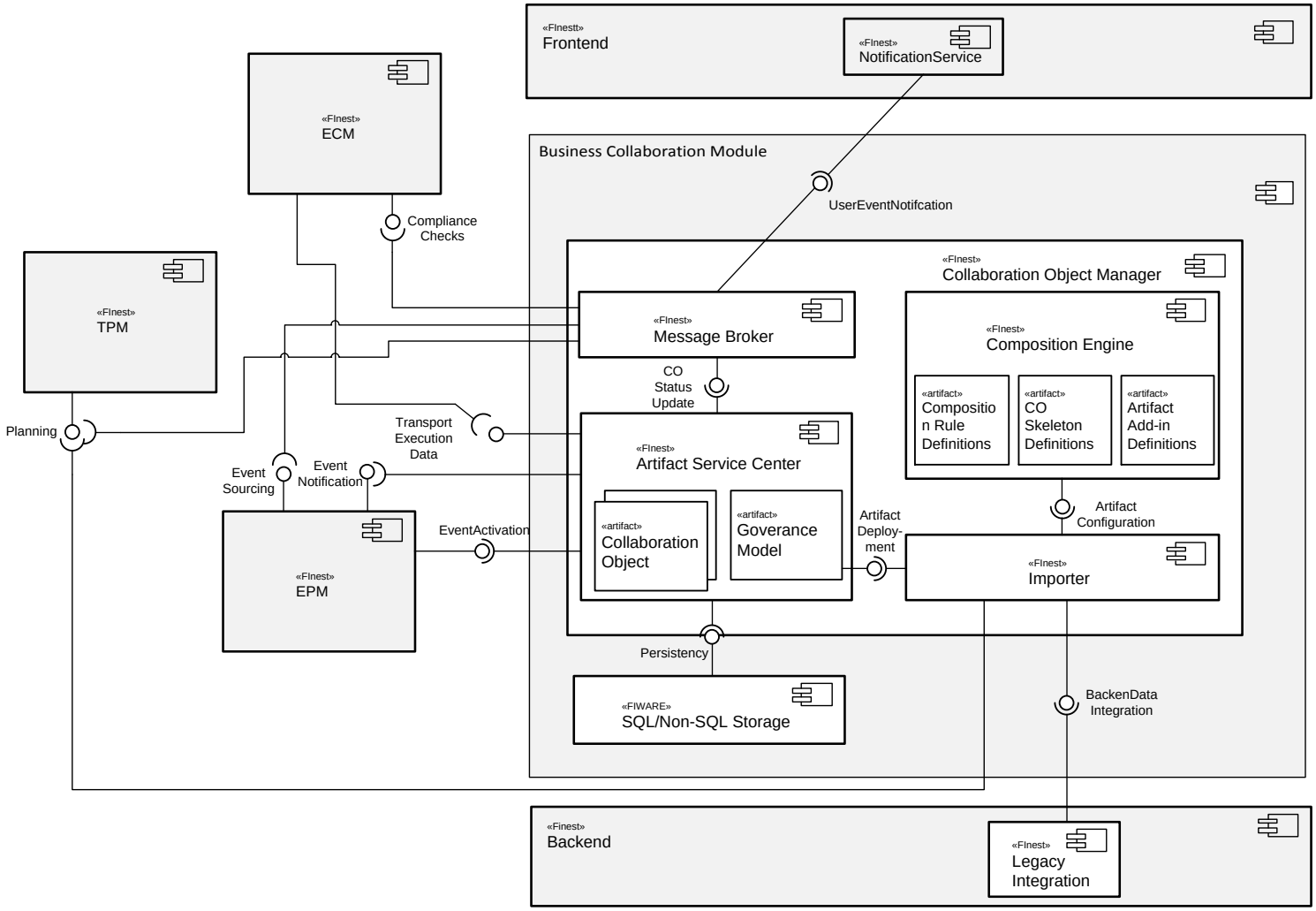

Figure 13 - Preliminary Technical BCM Design

3.2.2. 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. 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.

3.2.2.1. Provided Interfaces

We summarize all provided interfaces of the Importer component in Figure 14. In the subsequent Table 1 we give a detailed explanation of each provided interface method.

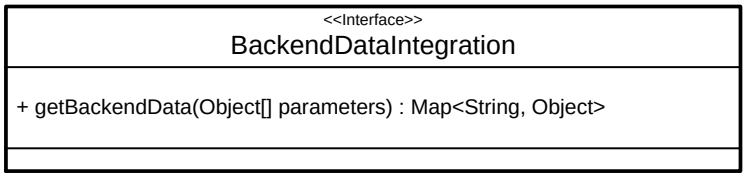

Figure 14 - Provided Interfaces of the Importer component

Table 1 - Interface Methods of the BackendIntegration Interface

Method Name	Visibility	Parameters	Return Value	Description
getBackendData	Public	Parameters : Object[]	Map<String, Object>	Used to gather backend data. Currently the exact way to access backend data is unknown. Because of that the parameters and return values are using general data types.

3.2.2.2. Used Data Types

No specific data types are used.

3.2.3. 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 analyzing 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.

3.2.3.1. Provided Interfaces

We summarize all provided interfaces of the Composition Engine in Figure 22. In the subsequent Table 2 we give a detailed explanation of each provided interface method.

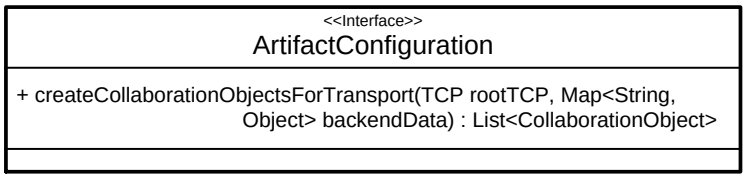

Figure 15 - Provided Interfaces of the Composition Engine

Table 2 - Interface Methods of the Artifact Configuration Interface

Method Name	Visibility	Parameters	Return Value	Description
create Collaboration ObjectsFor Transport	Public	rootTCP : TCP, backendData : Map<String, Object>	List<CollaborationObject>	Method to transmit the collected information from the transport plan and backend data to the Composition Engine in order to create Collaboration Object instances. Returns the created instances as a list

3.2.3.2. Used Data Types

Figure 16 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.

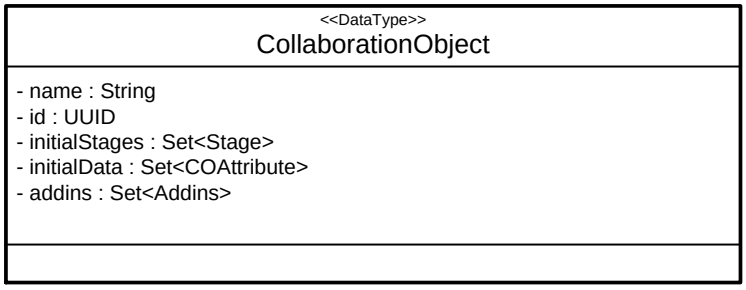

Figure 16 - Used Data Types of Composition Engine

3.2.4. 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 13 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 is 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 parameterized 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.

3.2.4.1. Provided Interfaces

We summarize all provided interfaces of the Artifact Service Center in Figure 17. In the subsequent Table 3 and Table 4 we give a detailed explanation of each provided interface method.

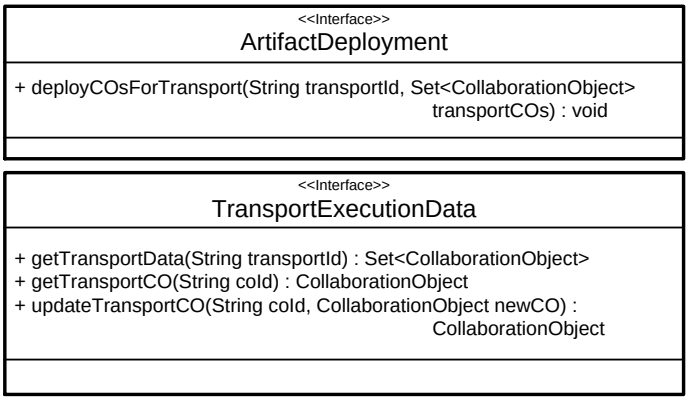

Figure 17 - Provided Interfaces of the Artifact Service Center

Table 3 - 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 4 - 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 : 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

3.2.4.2. Used Data Types

Figure 18 shows the used data types of the ArtifactServiceCenter component. The CollaborationObject data type is the same as used by CompositionEngine (*cf.* Section 3.2.3.2)

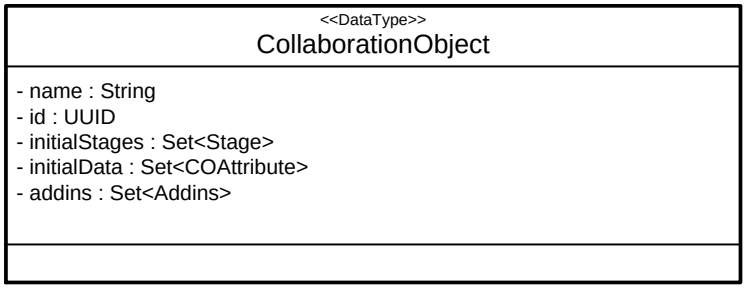

Figure 18 - Used Data Types of Composition Engine

3.2.5. Message Broker

The Message Broker is used to handle to distribution of internal CO status changes to other FInest core modules or the Front-End. As we describe in the previous section, Collaboration Objects can change their status in order to represent a changed situation of the (real-world) transport. About this status changes other modules and users have to be informed in order to enable them to react accordingly. The Message Broker emits messages for every occurred status change and in general we distinguish two different types:

- Messages to the Front-End users
- Messages to other core modules

Both message types are not mutually exclusive, which means that an internal status will often result in the emission of messages of both types. The former type is specific for a user or a group of users. Due to this the Message Broker has to use the User Management component of the Front-End in order to determine the right recipient. Messages will be delivered to the Notification Service of the Frontend, which deals with the final delivery of the message over predefined commination channels (*cf.* Deliverable D3.3). Messages to other core modules (the latter message type) have a more technical nature. They are used to inform about certain status changes within the Collaboration Objects. Examples for that could be the creation of a new object, the start or finalization of a transport as well as the reception of a specific document. As already described, the EPM uses these messages to start or stop rules. Additionally, the ECM updates its contracts and the TPM uses this execution data to update its planning routines.

3.2.5.1. Provided Interfaces

We summarize all provided interfaces of the MessageBroker in Figure 19. In the subsequent Table 5 and Table 6 we give a detailed explanation of each provided interface method.

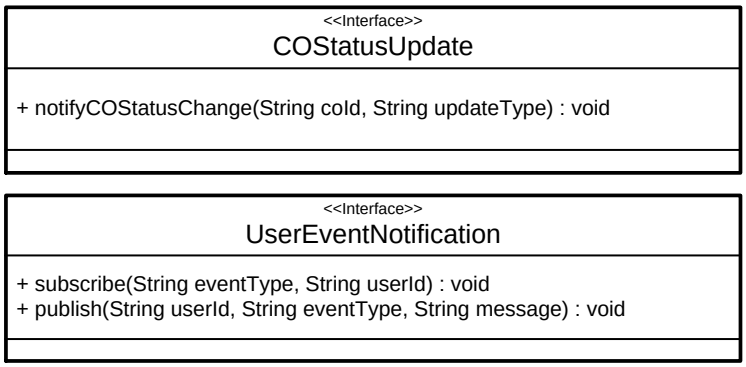

Figure 19 - Provided Interfaces by the MessageBroker Component

Table 5 - Interface Methods of the COStatusUpdate Interface

Method Name	Visibility	Parameters	Return Value	Description
notifyCOStatusChange	Public	coId: String	Void	Sends notifications about updates/changes within a managed CO to the

		updateType : String		MessageBroker to forward this information to the FrontEnd or other code modules

Table 6 - Interface Methods of the UserEventNotification 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 : String, message : String	Void	Publishes an event to a specific user.

3.2.5.2. Used Data Types

No specific data types are used.

3.2.6. 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.

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

3.2.6.1. Provided Interfaces

We summarize all provided interfaces of the MessageBroker in Figure 20. In the subsequent Table 5 we give a detailed explanation of each provided interface method.

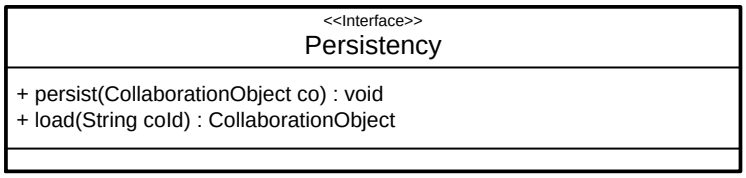

Figure 20 - Interfaces of the SQL/Non-SQL Storage Component

Table 7 - Interface Methods of the Persistency Interface

Method Name	Visibility	Parameters	Return Value	Description
Persist	Public	co : CollaborationObject	Void	Persists a CollaborationObject within the connected data base
Load	Public	coId : String	Void	Loads a specific CollaborationObject by a given Id

3.2.6.2. Used Data Types

Figure 21 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 3.2.3.2)

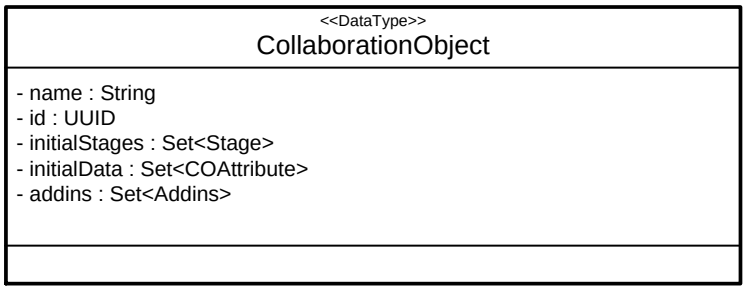

Figure 21 - Used Data Types of Composition Engine

3.3. Inter-Module Interaction

The BCM is responsible for the execution of a transport plan and enables user to always have an up-to-date view on their processes. The BCM serves as the central knowledge base for users, where they can access information with regard to transport processes. Consequently, the BCM has to integrate information from other core modules, the Frontend and the backend. Furthermore, the BCM has to inform its clients about the current execution status of a transport process. This results in a lot of interfaces (as shown in Section 3.2) between the components of

the FInest platform. In the remainder of this section we briefly describe this interaction and provide a sequence diagram as an overview at the end.

Already in the previous deliverable for M12, the BCM was described as the central knowledge base for the execution phase of a transport process. The interaction with the other modules was handled in at a conceptual level (cf. Deliverable D3.2), however there were some issues to be resolved:

- Structure and granularity of the transport plan?
- What information is already available within the TCP-TEP-based transport model and which has to be added in order to enable a monitoring by the EPM?
- Who will register/activate the required rules to monitor a transport process? Where origins the necessary information?
- How to transmit the current status of the transport process to the EPM in order to start or stop event processing rules?

All these questions were addressed during the collaborative work between the work packages. We elaborated the following interactions of the BCM with the other components of the FInest platform:

- The user initiates the transport planning by an invocation if the TPM. The TPM creates the transport plan in combination with the ECM and indicates the completion of the planning process to the BCM. The BCM fetches the transport plan, represented by the TCP-TEP document structure (XML). These documents contain all information to create Collaboration Objects for the execution phase of the transport as well as the information to activate the necessary event rules and parameters by the EPM.
- The BCM set ups up the Collaboration Objects based on the information contained in the TCP-TEP documents. In parallel, the BCM forwards the information necessary for the event rules activation (contained within the TCP-TEP) documents to the EPM.
- Users are allowed to add data to Collaboration Objects during the execution of the transport process. This could be used to indicate that specific process steps are successfully performed (e.g., the customs control is completed) or the user adds required documents. In both cases the BCM informs the EPM about this so that the event processing rules can be updated.
- During the whole process the EPM is sending the detected situation to the BCM so that the Collaboration Object-based transport model can be kept up-to-date
- After the process is finished the BCM sends the final status – such as resource consumption and performance quality of the involved parties – to the ECM and TPM to update their internal data.

In Figure 22 we summarize the interaction between the BCM and other FInest components in a sequence diagram.

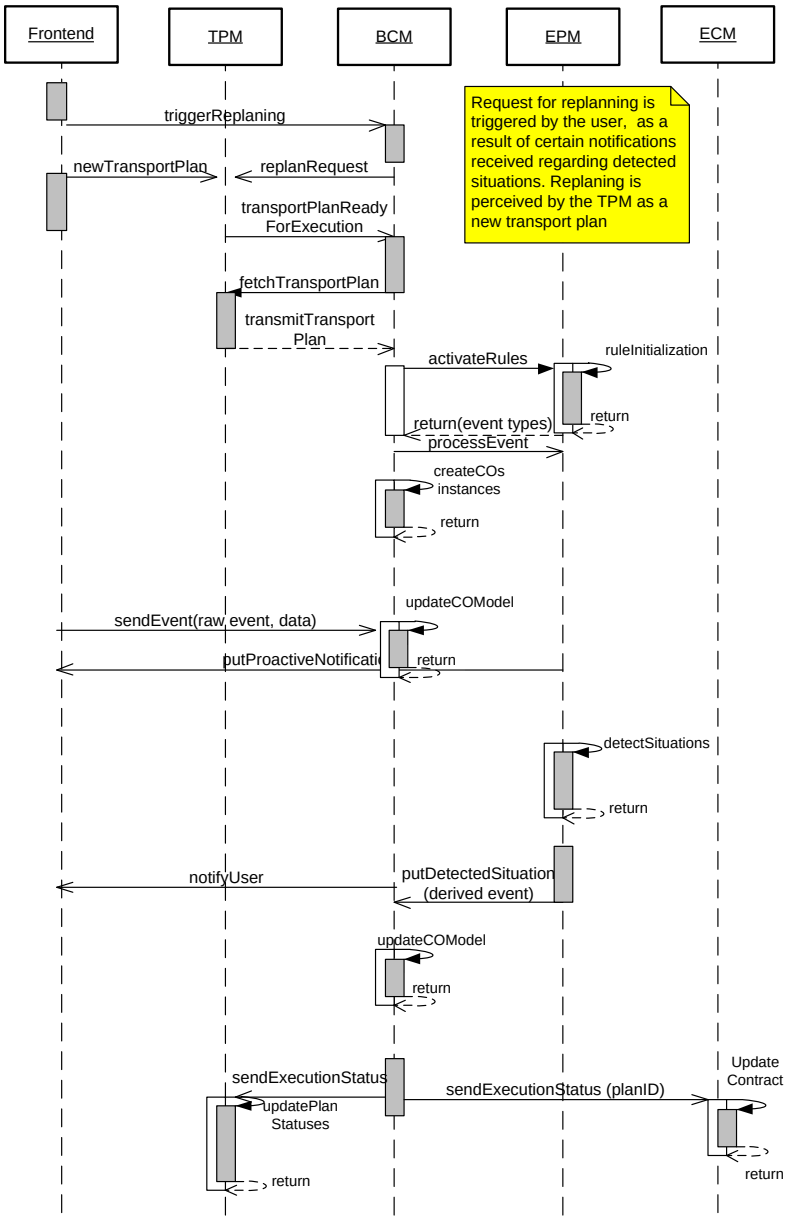

Figure 22 - BCM interaction with other FInest components

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 in to account previously. Furthermore, we outline the usage of additional GEs for the subsequent releases (or versions) of the BCM.

In Figure 12 (*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 3.2.6). 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 for the Big Data GE (capabilities of the SAMSON Platform). This might be beneficial for future releases of the BCM, when 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 to different to traditional workflow modeling so that a realization might not be applicable. But we do not made a decision about a suitable execution engine for Collaboration Objects. It might be that we find an approach to realize the process part of Collaboration Object based on traditional workflow models and then we will use the Workflow Execution Engine to within the BCM.

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. However, the technical specification of the BCM is not final and in future versions, when the implementation approaches are more sophisticated, the requirement for additional GEs might become visible.

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

[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. Conclusion

In this document we provide an evolved consideration of the Collaboration Object Model. We discuss in detail the modeling approach and denote the reasons for taking it. We also present a preliminary version of a meta-model for Collaboration Objects. Based on the conceptual design and requirement analysis presented in the previous deliverables, we elaborated the first version of a technical specification for the BCM. Additionally, we provide a short description of the Generic Enables used within the architecture.

After the introduction, in which we summarized the results of the last deliverable and described the activities since the last milestone, in Section 2, we comprehensively discussed the modeling approach for Collaboration Objects and presented a preliminary meta-model. In Section 3 we presented the technical specification of the BCM and its interaction with other FInest modules. In a last section (Section 4) we briefly describe the used Generic Enablers used within the technical specification.

Within the working period from M18 – M24 we will continue to work on the technical specification of the BCM. We will refine the provided interfaces as well as data types and refine the internal component structure. In parallel to that we will extend the meta-model for Collaboration Objects in order to meet the requirements defined in the Use case 1-3. A proof-of-concept implementation of the most vital parts of the technical architecture will show the technical feasibility of the elaborated design.

6. 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.