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

D8.5

Final Technical Specification and Phase II Implementation Plan for Logistics Contract Manager

Project Acronym	FInest	
Project Title	Future Internet enabled optimisation of transport and logistics networks	
Project Number	285598	
Workpackage	WP8 Logistics Contract Establishment and Management	
Lead Beneficiary	University of Duisburg-Essen (UDE)	
Editor(s)	Clarissa Cassales Marquezan	UDE
Contributors(s)	Stephan Heyne	SAP
	Michael Stollberg	SAP
	Andreas Metzger	UDE
	Marianne Hagaseth	MRTK
	Asmund Tjora	MRTK

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

	Cyril Alias	UDE
	Oyvind Olsen	NCL
	Michael Zahlmann	KN
	Metin Turkay	KOC
	Evert-Jan van Harten	AFKL
Reviewer	Agathe Rialland	MRTK
	René Fleishhauer	SAP
	Rod Franklin	KN
Dissemination Level	Public	
Contractual Delivery Date	31.03.2013	
Actual Delivery Date	31.03.2013	
Version	1.0	

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 is associated with the E-Contracting Module (ECM) and is submitted as specified in the FInest Description of Work (DoW) as deliverable D8.5 – "Final Technical Specification and Phase 2 Implementation Plan for Logistics Contract Manager" – associated with Work Package 8 (WP8). The role of this module within FInest is to provide online and semi-automatic contract establishment and management. The FInest core modules using the services offered by ECM are TPM (Transport Planning Module – associated with WP7) and BCM (Business Collaboration Module – associated with WP5).

In this document, we present the work conducted to achieve the following objectives associated with deliverable D8.5:

a) provide the final technical specification of the ECM (task T8.2);

b) provide domain-specific capabilities of the ECM module using GE (Generic Enablers) offered by FI-WARE (task T8.3);

c) technical realization of ECM aligned with concrete technical realization of GEs (task T8.4);

d) implementation of the proof-of-concept of the ECM module (task T8.4);and

e) detailed implementation plan for the follow up of the ECM module in Phase II (task T8.5).

It is important to notice that D8.5 complements the information in the previous report D8.3, which introduced the initial technical specification of the ECM module. The deliverable D8.5 updates the content of the previous deliverable with the final specification and final data models.

In addition, it is important to notice that the details about the prototype implementation are reported in the deliverable D8.4.

Document History

Version	Date	Comments
V0.1	13.02.2013	First draft of the structure
V0.2	04.03.2013	Restructuring sections and add text
V0.3	05.03.2013	Add content
V0.4	06.03.2013	End-to-end document
V0.5	07.03.2013	Document ready and sent for review
V0.6	26.03.2013	Final version including comments from reviewers
V1.0	29.03.2013	Released

Table of Contents

List of Tables

List of Figures

Acronyms

Acronym	Explanation
BCM	Business Collaboration Module
Dx.x	Number of the Deliverable
DoW	Description of Work
ECM	E-Contracting Module
EPM	Event Processing Module
FInest	**F**uture **In**ternet **e**nabled optimi**s**ation of **t**ransport and logistics networks
FI PPP	Future Internet Public Private Partnership
GE	Generic Enablers
ICT	Information and Communication Technology
QoS	Quality of Service
SLA	Service Level Agreement
SLO	Service Level Object
SME	Small Medium Enterprise
Tx.x	Number of the Task
TEP	Transport Execution Plan
T&L	Transport & Logistics
TPM	Transport Planning Module
TSD	Transport Service Description
TSP	Transport Service Provider
UI	User Interface
UML	Unified Modelling Language
USDL	Unified Service Description Language
RDF	Resource Description Framework
WP	Work Package

1 Introduction

The E-Contracting Module (ECM) is a core module of the FInest collaboration service. The role of this module is to support the online and real-time establishment and management of transport and logistics contracts. There are four main characteristics that make the ECM innovative and a more robust service with capabilities related to contracting activities (establishment, negotiation, execution).

i. Explicit connection between operations in Marketplaces (either external or inside Finest platform) and the contract management. The ECM module is designed to enable the integration of short term contracts agreed in the Marketplace into the contract management operations offered by the module. In this way, all types of contracts established by a party can be managed.

ii. With the ECM, planners can utilize real, precise and agreed terms of their contracts, and not interpreted terms, to define the SLAs for execution of service bookings.

iii. The ECM is designed to explicitly connect to different marketplaces, and not to replace them. To accomplish this, the ECM is envisioned to become a central point for the visualization of transport and logistics options (service offers and demands), whether these options are defined within the FInest collaboration service or originating from external parties.

iv. The ECM has operations explicitly defined to conduct different types of consistency checks between ongoing/planned transportation services versus the SLAs established in the contract. With the ECM, these checks can be done online and in near real time.

In the previous deliverable, D8.3 – "*Initial Technical Specification of Logistics Contract Manager*", we described the initial specification of the ECM service. We provided the description of components, interfaces, and information models necessary to provide the functionalities of the ECM module within the scope of WP8. In this deliverable, *i.e.* D8.5 – "*Final Technical Specification and Phase II Implementation Plan for Logistics Contract Manager"*, we present the final technical specification of the ECM module.

There are many similarities between these two documents. The former introduced the technical specification of the ECM module. The latter, on the one hand, updates the previous content with the final specification, and on the other hand relies on content and procedures previously reported. Thus, to avoid repetition of information, and at the same time enable a consolidation of the ECM final specification, we indicate in Table 1 which content has been reported in D8.3, what is updated in D8.5, and which content appears in D8.5 in order to maintain the consolidation of the ECM specification. In addition, we also present in Table 2 a summary of the technical specification updates included in this document.

Table 1 – Relation between D8.3 and D8.5 report on technical specification

Content in D8.3	Content in D8.5
Differentiator Factors of ECM (Section 1.1)	Summarized in Section 1.
Methodology (Section 2)	Omitted because we followed the same methodology.
Technical Specification (Section 3)	The content of the technical specification of some components were slightly updated and others were copied to provide the consolidation of the technical specification of the components in one single document.
Data Model Specifications (Section 4)	The content of some data model specifications were slightly updated and others were copied to consolidate the data model specification of the ECM in one single document.
Generic Enablers Used by the ECM Module (Section 5)	Omitted because no changes were observed in the GEs, and we used the same GEs previously reported.

Table 2 – Description of technical specification updates in D8.5

Name of updated entity	Updated Sections in D8.5
Marketplace Operations Component	Section 2.2.3; Section 2.3.2
Linked-USDL Vocabulary for T&L Service Offers	Section 3.1
Linked-USDL Vocabulary for T&L Contracts	**Section 3.3 (Introduced in this report)**

The documentation of the prototype implementation associated with the ECM technical specification is detailed in deliverable D8.4 (*Prototypical Implementation of Logistics Contract Manager*).

In addition, a very close channel for discussion with the Application Chapter of FI-WARE was established during the period of this report. The bi-lateral cooperation enabled both sides to better understand the concrete capabilities of the GEs that could be used in designing the ECM,

and we could also provide feedback on expected capabilities and examples of employment of the GEs in a domain-specific context.

1.1 Objectives of this Deliverable

In this document, we present the work conducted to achieve the following objectives associated with deliverable D8.5:

- provide the final technical specification of the ECM (task T8.2);
- provide domain-specific capabilities of the ECM module using GE (Generic Enablers) offered by FI-WARE (task T8.3);
- technical design and implementation of the ECM module aligned with concrete technical realization of GEs (task T8.4);
- technical implementation of the proof-of-concept of the ECM module (task T8.4);
- and detailed implementation plan for the follow up of the ECM module in Phase II (task T8.5).

1.2 Deliverable Organization

This deliverable is organized as follows.

- Section 2 describes the final technical specification of the ECM module. An overview of the component model is first described. A brief description of each component of the ECM module is presented in this section. In addition to the component descriptions, the interfaces of the ECM are also described in detailed. The major differences between the initial and final technical specifications are the updates on the UI and marketplace components of the ECM module. In the case of the UI components, the experience with the proof-of concept implementation enabled a refinement on this component. We extensively revised the marketplace components and provide a more comprehensive organization of them.
- Section 3 provides the updated description of the data models defined in WP8. These models are related to the definition of formats for how the contract information, as well as the transport and logistics service offers and demands can be expressed. In order to keep the alignment with FI-WARE and leverage the GEs, these models are defined following the Linked USDL information representation format.
- Section 4 provides the detailed Phase II implementation plan for the ECM. We provide a mapping from the current work done for the technical specification of the ECM in Phase I and how this work will be taken up in the structure of the follow up project, called cSpace. We also present the timeline for the development of this project, milestones, and the associated deliverables.

- Finally, Section 5 comprises the final remarks about the work developed during the period of this report.

1.3 Relationship with Other Work Packages

WP1 - *'Domain Characterization and Requirements Analysis'* - is concerned with the identification of business requirements in the transport and logistics domain. Those requirements contribute to the overall design goals and rationale of the E-Contracting Module in WP8.

WP2 - *'Use Case Specification'* - provides use case scenarios for the FInest project. These scenarios serve as the basis for the refinement of the business requirements identified in WP1, and serve for designing the demonstration, test, and evaluation prototypes associated with the technical WPs, such as WP8. These scenarios are based on major business challenges, thus closely related to the demonstrators and technical specifications developed in cooperation with the work done in the technical WPs.

WP3 - *'Solution Design and Technical Architecture'* - provides the overall design and architecture of the FInest collaboration service. This means that the core technical modules, i.e., design of the solutions in WP5 – WP8, must follow the definitions and guidelines of WP3 in order to be integrated into the FInest collaboration service.

WP4 - *'Experimentation Environment'* - is concerned with defining required experimental infrastructure for demonstrating the use cases scenarios described in WP2. During the demonstration process the elements required by the WP8 module will also need to be considered by the experimental infrastructure.

The technical WPs - *WP5, WP6, and WP7; respectively, 'Business Network Collaboration', 'Proactive Event Driven Monitoring', 'Transport Planning and Replanning'* - are not isolated entities in the overall FInest collaboration service. In this sense, WP8 is primarily associated with WP5 and WP7, because information about contracts must flow between the e-contracting, business collaboration and planning modules. In this deliverable, sequence diagrams in Figure 10 and Figure 11 show details of the relationship among WP8 (i.e., ECM) and the other modules (i.e., TPM and BCM).

WP9 - *'FI PPP Alignment'*- is responsible for the alignment of the FInest project with the FI PPP Program. WP8 is related to WP9 in the sense that WP9 helps in the identification of GEs associated with e-contracting services, and also in the potential alignment of technical specifications to better support the use of the GEs within WP8.

2 Final ECM Technical Specification

The initial technical specification of the ECM was reported in D8.3. In this document we refine the design based on additional information on the capabilities of the GEs and our learning in developing the proof-of-concept. This chapter introduces the details of this specification by describing: (i) the component model refined from the conceptual architecture, (ii) the list of interfaces and their respective operations provided by the ECM, (iii) the list of data types associated with the methods of the offered support for the ECM interfaces, and (iv) the sequence diagram demonstrating how the ECM is able to interact with other core modules from the FInest collaboration service.

2.1 Component Model Overview

Figure 1 presents the component diagram of the ECM after the refinement of the technical specification conducted during the period of this report. This diagram shows the main components, interfaces provided and required, as well as the GEs from FI-WARE on which the ECM functionalities will be built.

The ECM module is depicted in the middle of Figure 1 and the grey area in the middle illustrates all components belonging to the ECM module architecture. The dark components on the top and the bottom of Figure 1 illustrate other FInest modules to which the ECM is associated. The white components in the bottom of Figure 1 represent the different external marketplaces whose information can be integrated into the ECM module.

Internally, the ECM module is subdivided in four main components:

i. Contract Operations;

ii. Contract Analytics;

iii. Marketplace Operations; and

iv. Connection to UI.

Each of the above listed components is organized internally into sub-components. This structure helps to identify a better clustering of functionalities and interfaces that need to be provided. The description of the components and their internal organization are discussed in the next section.

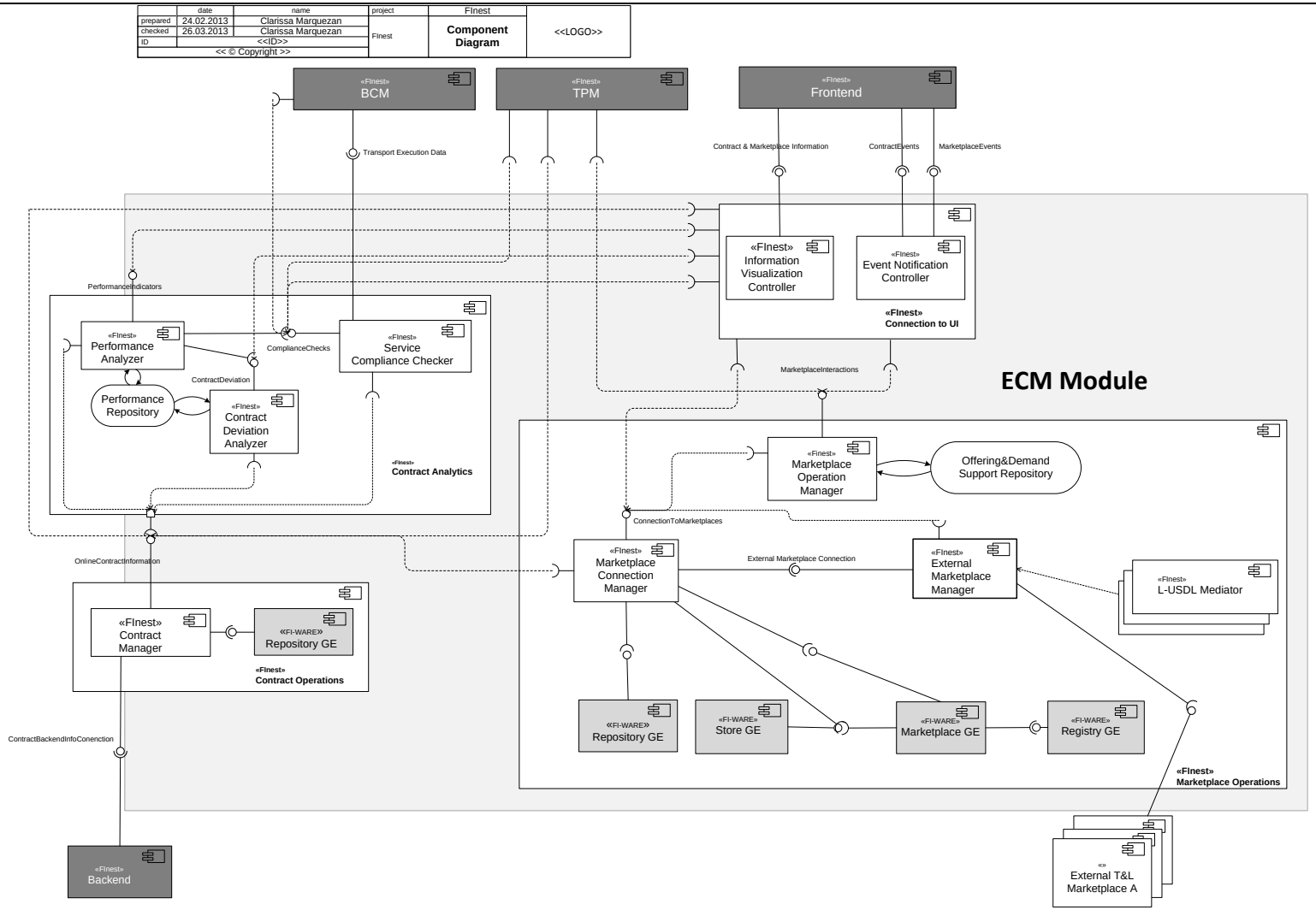


Figure 1 – Component Diagram of E-Contracting Module (ECM)

2.2 Component Description

This section presents more detailed characteristics of the four main components of the ECM module. The focus is to provide an overview of the interfaces required and provided by each of the ECM components, as well as how such sub-components are related to each other and also to other modules of the FInest collaboration service.

2.2.1 Contract Operations Component

The Contract Operations Component is primarily concerned with actions related to established contracts, such as storing, reading, deleting, updating, and searching for information concerning the contracts. The contract information stored in FInest is a subset of the legal contract. This subset is related to the service requirements and service level agreements necessary to drive the daily activities of transport and logistics service execution. The repository for storing such information is built on top of the Repository GE as depict in Figure 1. Using this GE we can leverage the searching mechanisms that is supported by this GE.

The functionalities provided in this component are, in fact, enabled by the association of two sub-components. One is the *Contract Management* sub-component that is designed and developed under the scope of the FInest project. The other is the *Repository GE* that is designed and developed under the scope of FI-WARE project.

- **Contract Management**: Provides interfaces for the online access of contract information and an interface for connecting to backend systems in order to retrieve and send information to legacy systems related to contracts. This sub-component relies on the basic CRUD (create, read, update, delete) operations from the Repository GE in order to provide domain specific methods related to the contract information in the transport and logistics domain.
- **Repository GE <<FI-WARE>>**: This GE provides an interface with methods related to storing, deleting, updating, and searching for data (collections or resources according to the FI-WARE definition). Data in the context of FI-WARE can be anything, and for this reason we defined in FInest WP8 a specific information model to represent the information of transport and logistics contracts (as detailed in Section 3.3). As mentioned before, the information related with the transport and logistics contracts is then stored in the Repository GE.

2.2.2 Contract Analytics Component

The Contract Analytics Component enables quick online access to the most relevant information on contracts for the daily activities in transport and logistics services execution. It also provides services for analysing the data of the contract versus the executed service in a near real time fashion. This component encloses a set of activities related to the analysis of situations related to contracts. For example, activities within the scope of this component are: detection of constant violations of the contract SLAs together with recommendation for changing the terms of the

contract; checking in a very early stage of the transport and logistics service planning process if the required booked capacity complies with the actual terms of the contract between the parties. No GEs are directly required by this component. Nonetheless, this component relies on the contract operations interfaces in order to get access to the contract information.

Three sub-components are part of the Contract Analytics Component. All of these sub-components rely on the interface provided by the Contract Management sub-component in order to access the stored contract information. In addition, the Contract Analytics Component has a storage entity called **Performance Repository.** It is used to store in a permanent way the metrics, KPIs, and the output of the analysis associated with the contracts.

- **Service Compliance Checker:** Responsible for performing operations related to compliance checks using the contract information as the reference value. This sub-component exposes one interface that can be accessed by the: (i) TPM module (in order to check the compliance of bookings against the actual terms of the contract); (ii) BCM (in order to request a check whether the executed transport and logistics service was conducted according to the agreed terms of the contract); and (iii) the internal sub-component Performance Analyzer.
- **Contract Deviation Analyzer:** Encloses the operations associated with the analysis of the contract document itself and what is related to it. For example, this sub-component exposes one interface where it is possible to identify the recurrence of deviations associated with a long term contract. This interface is not mandatory for the core modules of the FInest project, but it can be used by the Frontend (with the proper authentication and authorization credentials) in order to expose the contract deviation information. The outputs of the operations executed in this sub-component are also stored in the Performance Repository as illustrated in Figure 1.
- **Performance Analyzer:** Responsible for executing operations associated with the KPI analysis of partners. Through the provided interface, other modules – particularly the Frontend – can have access to the methods that calculate performance indicators. The information required and generated by these operations is stored in the Performance Repository. This sub-component might use the interfaces provided by the other two sub-components inside Contract Analytics Component in order to provide more sophisticated operations.

2.2.3 Marketplace Operations Component

This component is associated with multiple activities before and during the establishment of a connection between a transport and logistics service provider and the transport and logistics client within the scope of marketplaces. In addition, it is also within the scope of this component to enable the integration of information from external marketplaces into the ECM module. This component heavily exploits many GEs offered by the Application Chapter of FI-WARE. FI-WARE has already defined many of the operations that need to be executed in a marketplace, such as uploading offers, searching for offers, inserting new stores, defining roles of participants, etc. In addition, to use the API offered by the GEs (Marketplace, Store, Registry

and Repository) we defined domain specific operations that use the basic API from FI-WARE in order to provide more sophisticated services/operations for the users of the ECM module.

Building upon the basic functionalities offered by the GEs in the Application Chapter of FI-WARE, we defined in WP8 a set of sub-components that are able to handle the complexity of providing sophisticated marketplace operations for the transport and logistics domain, and the capacity of connecting to external transport and logistics marketplaces (marketplaces built using structures not developed in accordance with FI PPP architecture concepts). For external marketplace connections, we specifically designed mechanisms of regulating how information from the external marketplaces are pushed into the ECM as well as how information can be pulled back into such external marketplaces.

Of special note, during the interaction with the FI-WARE team, we identified the need of designing three different components to handle the marketplace operations. The rationale for this decision is associated with the fact that the Marketplace GE, Mediator GE, and Proxy GE in the App Chapter cannot transparently handle integration and management of information from external marketplaces.

The search operation of the Marketplace GE, that is of extreme importance for the ECM in order to find offers and demands, is not able to search for information that is not stored inside the Marketplace GE. This means that we had to add to the ECM design a set of components. They are able to pull and push information from external marketplace, manage this connectivity and expose this information inside the FInest marketplace (based on the Marketplace GE) as if it were native from FInest, i.e., as if the offers and demands were created inside the FInest marketplace. A user from the ECM module would have the impression that FInest is centralizing all the information, instead of having to manually access each one of the different marketplaces.

Indeed, the ECM is centralizing the information, through the functionalities offered by the Marketplace Connection Manager and Marketplace Operation Manager sub-components; and guaranteeing that the pulled information from external marketplaces is consistent, coherent and synchronized with the external source of this information, through the functionalities offered by the External Marketplace Manager sub-components. In summary, the first two sub-components just mentioned handle the daily operations on top of stored information inside the Marketplace GE, whereas the other two sub-components are required to feed the Marketplace GE, Store GE, Registry GE, and Repository GE with information from external marketplaces.

The sub-components of the Marketplace component as listed as follows.

- **Marketplace Connection Manager**: Responsible for being the connection point between the ECM module and FI-WARE GEs required for enabling the marketplace operations. This sub-component exposes one interface to which different entities within the ECM module (there is no restriction to external access only) connect in order to gain access to the information of marketplaces accessible by the ECM module. The decision of clustering the basic operations of the GEs in only one access point inside the ECM is based on a risk assessment. If for some reason the technology to store and search for information associated with marketplaces changes (e.g., the API of the services exposed by the Marketplace GE changes in the second release), we need to update only one

component inside the ECM module without changing the domain specific operations built up on the basic service APIs from the Marketplace GE.

- **Marketplace Operation Manager**: This sub-component is responsible for offering the sophisticated operations for the transport and logistics domain using the basic services offered by the Marketplace Connection Manager sub-component. One interface is provided by this sub-component and can be accessed by the Frontend or the TPM module. In addition, the extra information handled by this sub-component, such as bids or watch lists of offers and demands, is stored in the *Offering&Demand Support Repository* as illustrated in Figure 1.
- **External Marketplace Manager**: For each external marketplace to which the ECM module connects, it is necessary to create a proxy that enables the conversion of information between the external Marketplace and the internal data formats of the ECM. This sub-component is responsible for managing the lifecycle of the set of marketplace proxies, as well as guaranteeing the functionality of the information exchange channel between ECM and each external marketplace. Each proxy developed for integrating information from external marketplaces needs to configure and register inside the aforementioned GEs the information that will be later on accessible via functionalities offered by the Marketplace Operation Manager sub-component. These proxies need to be monitored, and the ECM module needs to guarantee their proper operation in order to really enable the flow of information from/to external marketplaces. In addition, this sub-component is responsible for managing the operations of pushing and pulling information from external marketplaces. As described before, the support offered by FI-WARE to integrate information from external marketplaces is extremely limited. It is only possible to search for information using the Marketplace GE APIs if this information is stored inside the Repository GE and it is stored according the Linked-USDL format. Thus, it is necessary to define mechanisms able to intermediate this conversion of information and also keep the consistency between the Linked-USDL version of the external information and the external information itself. This process is all supported by this sub-module. Thus, offers and demands of each proxy associated with external marketplaces have to be imported and translated into Linked-USDL and then stored inside the Marketplace GE, so that the search operations of the ECM module can find these external offers. Because the search operation cannot be done directly in the external marketplace system, the ECM module, and specifically this sub-component, has to be responsible for keeping the synchronization actions such as forwarding the bid information created for a transport and logistics demand stored inside the Marketplace GE if this offer is associated with an external marketplace.
- **L-USDL Mediator**: This is a sub-component that needs to be developed for each external marketplace. It is responsible for enabling the data mediation between the Linked-USDL format used inside the ECM and whatever data format is used by an external marketplace. The marketplace proxy developed for a specific external marketplace (e.g., External T&L Marketplace A, as illustrated in Figure 1) will invoke the specific developed L-USDL Mediator (e.g., L-USDL Mediator Marketplace A) for the data type associated with the specific marketplace in question (e.g., Marketplace A).

- **Store GE, Marketplace GE, Registry GE, Repository GE <<FI-WARE>>**: These GEs are the basic enablers needed in order to enable the basic marketplace functionalities. The Application Chapter from FI-WARE, in which the aforementioned GEs are defined, contains further GEs also associated with marketplaces. However, the ones used by the ECM are limited to these core GEs.

2.2.4 Connection to UI Component

The information passed to the UI can be of two different natures: event notifications and ordinary information associated with visualization of the supported methods offered by the ECM interfaces. This component is responsible for coordinating the interactions between the ECM methods and the visualization of the results processed by such methods.

The mechanisms to provide the visualization of the operations and information handled by the ECM is performed by the two sub-components of the Connection UI component as described below.

- **Information Visualization Controller**: This sub-component is responsible for providing the interface between the ECM module and the Frontend of the FInest collaboration service. The focus of this sub-component is to control all kinds of interactions not related to event notifications. For example, this sub-module provides methods so that the information from a long term contract can be visualized, or it enables the results from a compliance check to be visualized in the UI, etc.
- **Event Notification Controller**: Responsible for receiving and forwarding UI event notifications to the Frontend component of the FInest collaboration service. It is important to notice that the event notifications addressed here are not related to the type of events handled by the EPM (Event Processing Module) module from FInest. Examples of events handled in the Event Notification Controller are the ones generated by the Contract Analytics Component informing that contracts are near the expiration date and need to be extended or that a long term contract should be created with a spot market partner (because this partner is constantly used).

2.3 Interface Definitions

This section describes the details of the interfaces provided by the ECM components. For each interface, the core set of methods (or operations) is also presented. By using these methods, other modules from the Finest collaboration service can get access to the functionalities offered by the ECM module.

2.3.1 Interfaces related to Contract Operation and Contract Analytics Components

In this section, the interfaces discussed in Section 2.3.1.1 and 2.3.1.2 are related to the Contract Operation component and the others are associated with the Contract Analytics component. Figure 2 introduces the class diagram characterizing each one of the interfaces and their respective methods.

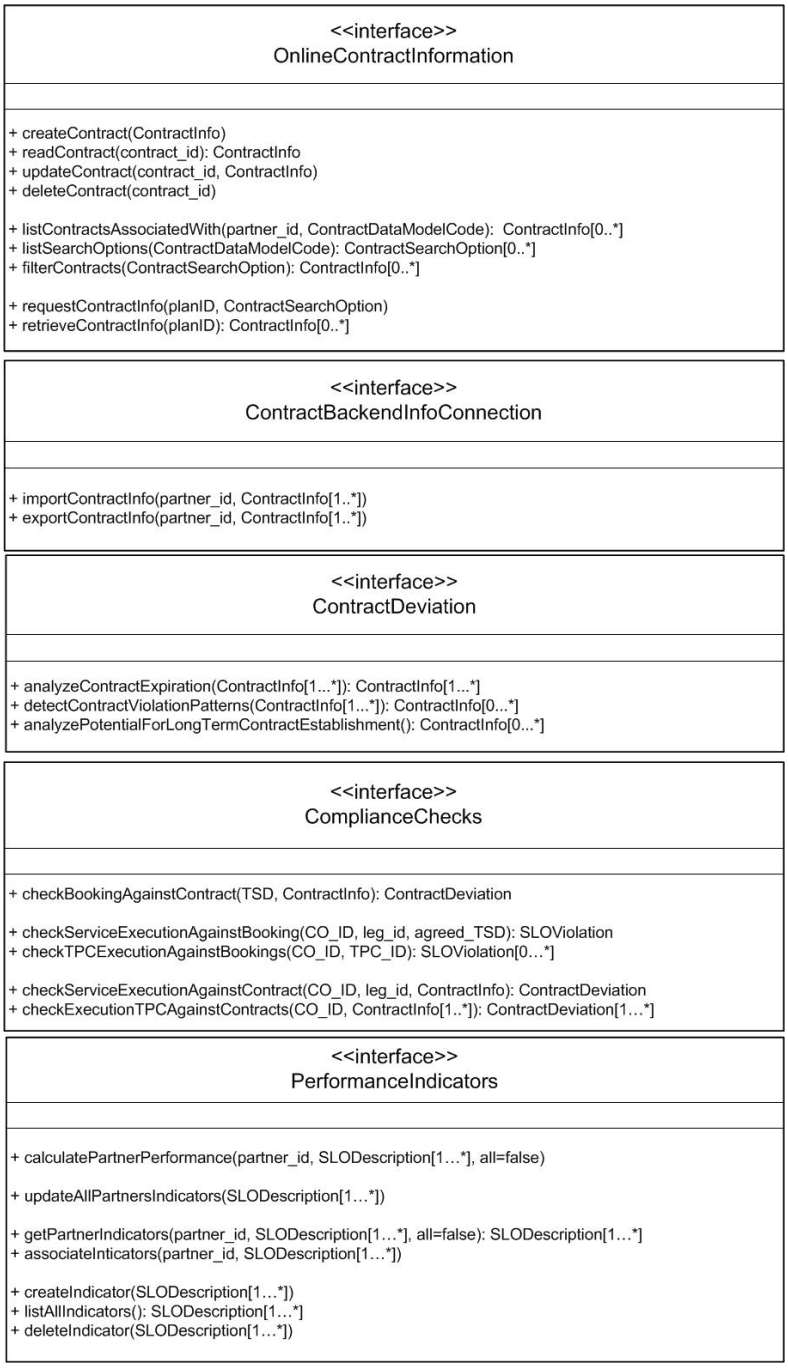


Figure 2 – Provided Interfaces related to Contract Operations and Contract Analytics Components

2.3.1.1 OnlineContractInformation

This interface is provided by the Contract Manager sub-component inside the Contract Operation component. Through the methods of this interface any module can have access to the information of established contracts and manage such information (i.e, create, read, update, delete). Table 3 depicts the methods provided by this interface.

Table 3 – Methods of OnlineContractInformation Interface

Method	Description
`Public createContract(ContractInfo)`	Create and store a new contract in the repository. *Parameter:* The contract information (`ContractInfo`) *Return value:* None
`Public readContract(contract_id): ContractInfo`	Return the contract information for a specific stored instance. *Parameter:* Contract identification `(contract_id)` *Return value:* The contract information `(ContractInfo)`
`Public updateContract(contract_id, ContractInfo)`	Update the contract information for a specific stored instance. *Parameter:* Contract identification `(contract_id)` and the information to be updated `(ContractInfo)` *Return value:* None
`Public deleteContract(contract_id)`	Delete the contract information for a specific stored instance. *Parameter:* Contract identification `(contract_id)` *Return value:* None
`Public listContractsAssociatedWith(partner_id, ContractDataModelCode): ContractInfo[0..*]`	Return the list of contract information for a given partner. *Parameter:* Partner identification `(partner_id)` and the data model type of the contract `(ContractDataModelCode)`

	Return value: List of contract information **(ContractInfo[0..*])**
Public listSearchOptions(ContractDataModelCode): ContractSearchOption[0..*]	Return the list of search options search options for filtering contract information associated with a contract data model type. *Parameter:* Data model type of the contract **(ContractDataModelCode)** *Return value:* List of search options **(ContractSearchOption[0..*])**
Public filterContracts(ContractSearchOption): ContractInfo[0..*]	Provide a list of contracts matching a given search condition. *Parameter:* Search condition information **(ContractSearchOption)** *Return value:* List of contracts **(ContractInfo[0..*])**
Public requestContractInfo(planID, ContractSearchOption)	Indicate to the ECM the need to retrieve contract information associated with a specific T&L plan tentative. *Parameter:* Identification of plan requesting the information and search condition information **(ContractSearchOption)** *Return value:* None
Public retrieveContractInfo(planID): ContractInfo[0..*]	Return the contract information associated with a specific T&L plan tentative. *Parameter:* Identification of plan requesting the information **(planID)** *Return value:* List of contract information **(ContractInfo[0..*])**

2.3.1.2 Interface: ContractBackendInfoConnection

This interface is provided by the Contract Manager sub-component inside the Contract Operation component. Through the methods of this interface any module can have access to operations related to legacy systems related to contract. Table 4 depicts the methods provided by this interface.

Table 4 – Methods of ContractBackendInfoConnection Interface

Method	Description
`Public importContractInfo(partner_id, ContractInfo[1..*])`	Import contract information from a specific partner from legacy systems. *Parameter:* Identification of partner (`planID`) and list of the contract information (`ContractInfo[1..*]`) *Return value:* None
`Public exportContractInfo(partner_id, ContractInfo[1..*])`	Export contract information from a specific partner to legacy systems. *Parameter:* Identification of partner (`planID`) and list of contract information (`ContractInfo[1..*]`) *Return value:* None

2.3.1.3 Interface: ContractDeviation

This interface is provided by the Contract Deviation Analyser sub-component inside the Contract Analytics component. Table 5 depicts the methods provided by this interface.

Table 5 – Methods of ContractDeviation Interface

Method	Description
`Public analyzeContractExpiration(ContractInfo[1...*]): ContractInfo[0...*]`	Identify contract expiration. *Parameter:* List of contract information to be checked (`ContractInfo[1..*]`) *Return value:* List of contract expired (`ContractInfo[0...*]`)
`Public detectContractViolationPatterns(ContractInfo[1...*]): ContractInfo[0...*]`	Identify contracts that have recurrence of violations. *Parameter:* List of contract information to be checked (`ContractInfo[1..*]`) *Return value:* List of contract with violations (`ContractInfo[0...*]`)

`Public analyzePotentialForLongTermContractEstablishment():ContractInfo[0...*]`	Identify short contracts between the same partners that have the potential to become long term partners. *Parameter:* None *Return value:* List of contract with potential to become long term contracts (`ContractInfo[0...*]`)

2.3.1.4 Interface: ComplianceChecks

This interface is provided by the Service Compliance Checker sub-component inside the Contract Analytics component. Table 6 depicts the methods provided by this interface.

Table 6 – Methods of ComplianceChecks Interface

Method	Description
`Public checkBookingAgainstContract(TSD, ContractInfo): ContractDeviation`	Check if the booking information is compliant with the terms of the contract associated with the partners of the booking. *Parameter:* The description of the service to be booked (`TSD`) and the contract information associated with the specific partners involved in the booking (`ContractInfo`). *Return value:* Contract deviation information (`ContractDeviation`)
`Public checkServiceExecutionAgainstBooking(CO_ID, leg_id, agreed_TSD): SLOViolation`	Check if the service execution associated with a booking is compliant with the terms of the booking. *Parameter:* The identification of the Collaboration Object associated with the service execution (`CO_ID`), the specific leg indicating the partners in that leg (`leg_id`), and the identification of the description of the booked service (`agreed_TSD`). *Return value:* SLO deviation information (`SLOViolation`)

Public checkTPCExecutionAgainstBookings(CO_ID, TPC_ID): SLOViolation[0…*]	Check if all executions of all the legs of the transport chain plan are compliant with the booked terms of the booking. *Parameter:* The identification of the Collaboration Object associated with the service execution **(CO_ID)**, the transport chain plan identification **(TPC_ID)**. *Return value:* SLO deviation information (**SLOViolation**)
Public checkServiceExecutionAgainstContract(CO_ID, leg_id, ContractInfo): ContractDeviation	Check if the service execution associated with a contract is compliant with the terms of the contract. *Parameter:* The identification of the Collaboration Object associated with the service execution **(CO_ID)**, the specific leg indicating the partners in that leg **(leg_id)**, and the identification of the contract associated with the leg **(ContractInfo)**. *Return value:* Contract deviation information (**ContractDeviation**)
Public checkExecutionTCPAgainstContracts(CO_ID, ContractInfo[1..*]): ContractDeviation[1…*]	Check if the whole service execution associated with contracts in a transport chain plan is compliant with the terms of the associated contracts. *Parameter:* The identification of the Collaboration Object associated with the service execution **(CO_ID)** and the identification of contracts associated with the chain **(ContractInfo[1..*])**. *Return value:* Contract deviation information (**ContractDeviation[1…*]**)

2.3.1.5 Interface: PerformanceIndicators

This interface is provided by the Service Compliance Checker sub-component inside the Contract Analytics component. Table 7 depicts the methods provided by this interface.

Table 7 – Methods of PerformanceIndicators Interface

Method	Description
Public calculatePartnerPerformance(partner_id, SLODescription[1…*], all=false)	Calculate performance indicator values of a specific partner and store the information in the Performance Repository. *Parameter:* The identification of the partner **(partner_id)** and the list of performance indicators **(SLODescription[1…*])**. *Return value:* None
Public updateAllPartnersIndicators(SLODescription [1…*])	Calculate for all partners stored in the Performance Repository the required performance indicators and store each value in the Performance Repository. *Parameter:* The list of performance indicators **(SLODescription[1…*])**. *Return value:* None
Public getPartnerIndicators(partner_id, SLODescription[1…*], all=false): SLODescription[1…*]	Return the list of performance indicators associated with a specific partner. *Parameter:* The identification of the partner **(partner_id)**, list of performance indicators **(SLODescription[1…*])**, and a Boolean value indicating if all performance indicators should be considered or not **(all)**. *Return value:* List of performance indicators **(SLODescription[1…*])**.
Public associateInticators(partner_id, SLODescription[1…*])	Associate a set of performance indicators with a specific partner and store the information in the Performance Repository. *Parameter:* The identification of the partner **(partner_id)** and the list of performance indicators **(SLODescription[1…*])**. *Return value:* None
Public createIndicator(SLODescription[1…*])	Insert new performance indicator(s) in the Performance Repository. *Parameter:* The list of performance indicators **(SLODescription[1…*])**.

	Return value: None
Public listAllIndicators(): SLODescription[1…*]	Return a list of all performance indicators stored in the Performance Repository. *Parameter:* None. *Return value:* The list of performance indicators **(SLODescription[1…*])**
Public deleteIndicator(SLODescription[1…*])	Remove one or more performance indicator(s) from the Performance Repository. *Parameter:* The list of performance indicators **(SLODescription[1…*]).** *Return value:* None.

2.3.2 Interfaces related to Marketplace Operations Component

In this section, all described interfaces are related to the Marketplace Operation component. Figure 3 - Figure 5 introduce the class diagram characterizing each one of the interfaces and their respective methods.

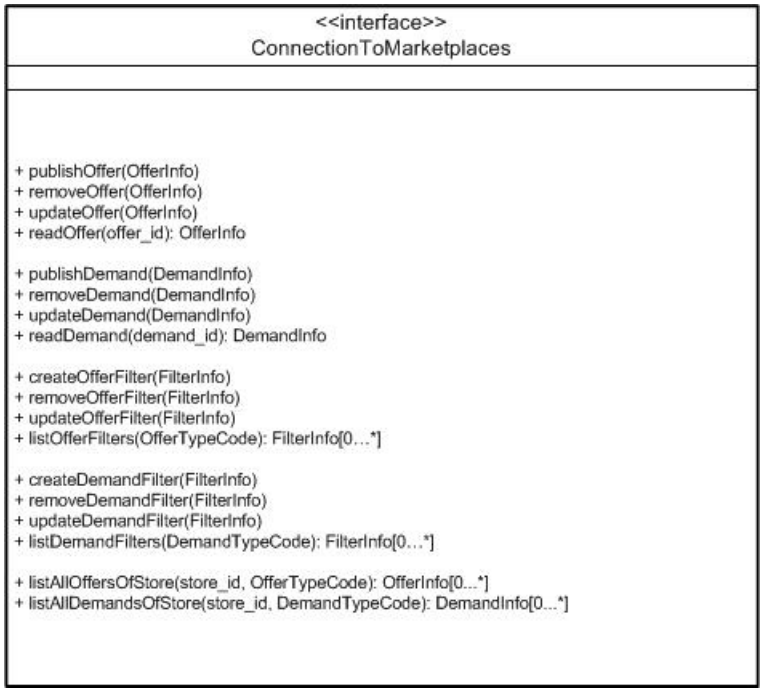


Figure 3 – Part I - Provided Interfaces related to Marketplace Operations Component

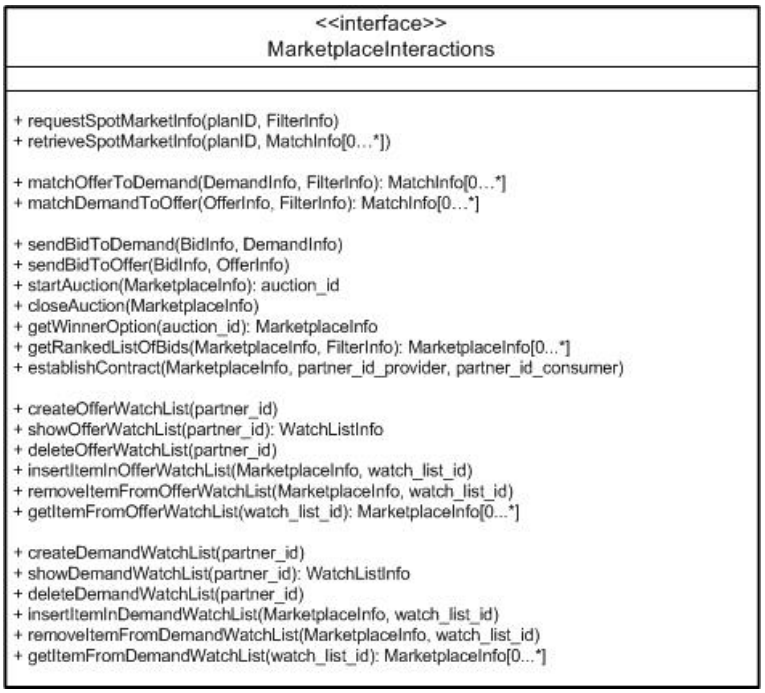


Figure 4 – Part II - Provided Interfaces related to Marketplace Operations Component

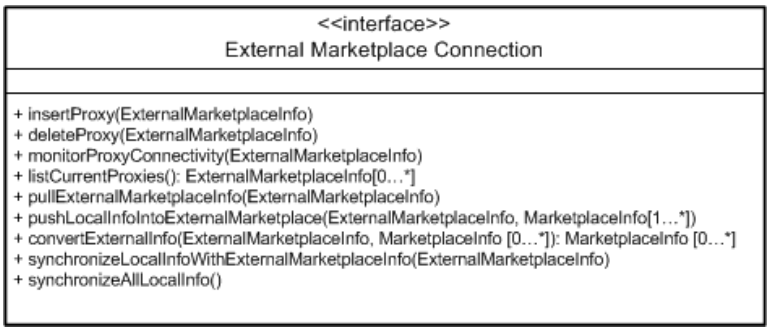


Figure 5 – Part III - Provided Interfaces related to Marketplace Operations Component

2.3.2.1 Interface: MarketplaceConnection

This interface is provided by the Marketplace Connection Manager and Table 8 depicts the methods provided by this interface. These methods are the very basic ones needed to connect to the offers and demands of a marketplace. It is out of the scope of this interface to configure the roles of the participants of the marketplace.

Table 8 – Methods of MarketplaceConnection Interface

Method	Description
`Public publishOffer(OfferInfo)`	Publish in the marketplace an offer. *Parameter:* The offer information

	(OfferInfo). *Return value:* None.
Public removeOffer(OfferInfo)	Remove an offer from the marketplace. *Parameter:* The offer information **(OfferInfo).** *Return value:* None.
Public updateOffer(OfferInfo)	Update the offer information on the marketplace. *Parameter:* The offer information **(OfferInfo).** *Return value:* None.
Public readOffer(offer_id): OfferInfo	Retrieve the offer information from the marketplace. *Parameter:* The offer information identification **(offer_id).** *Return value:* The offer information identification **(OfferInfo).**
Public publishDemand(DemandInfo)	Publish in the marketplace a demand. *Parameter:* The demand information **(DemandInfo).** *Return value:* None.
Public removeDemand(DemandInfo)	Remove a demand from the marketplace. *Parameter:* The demand information **(DemandInfo).** *Return value:* None.
Public updateDemand(DemandInfo)	Update the demand information on the marketplace. *Parameter:* The demand information **(OfferInfo).** *Return value:* None.
Public readDemand(demand_id): DemandInfo	Retrieve the demand information from the marketplace.

	Parameter: The demand information identification **(demand _id)**. *Return value:* The demand information identification **(DemandInfo)**.
Public createOfferFilter(FilterInfo)	Create a filter related to an offer type. *Parameter:* The filter information **(FilterInfo)**. *Return value:* None.
Public removeOfferFilter(FilterInfo)	Remove a filter related to an offer type. *Parameter:* The filter information **(FilterInfo)**. *Return value:* None.
Public updateOfferFilter(FilterInfo)	Update filter information related to an offer type. *Parameter:* The filter information **(FilterInfo)**. *Return value:* None.
Public listOfferFilters(OfferTypeCode): FilterInfo[0…*]	List filters related to an offer type. *Parameter:* The offer type **(OfferTypeCode)**. *Return value:* The list of filters **(FilterInfo[0…*])**.
Public createDemandFilter(FilterInfo)	Create a filter related to a demand type. *Parameter:* The filter information **(FilterInfo)**. *Return value:* None.
Public removeDemandFilter(FilterInfo)	Remove a filter related to a demand type. *Parameter:* The filter information **(FilterInfo)**. *Return value:* None.
Public updateDemandFilter(FilterInfo)	Update filter information related to a demand type. *Parameter:* The filter information

	(FilterInfo). *Return value:* None.
Public listDemandFilters(DemandTypeCode): FilterInfo[0…*]	List filters related to a demand type. *Parameter:* The demand type **(DemandTypeCode).** *Return value:* The list of filters **(FilterInfo[0…*]).**
Public listAllOffersOfStore(store_id, OfferTypeCode): OfferInfo[0...*]	List all offers of a specific store related to a specific offer type. *Parameter:* The store identification **(store_id)** and the offer type **(OfferTypeCode).** *Return value:* The list of offers **(OfferInfo [0…*]).**
Public listAllDemandsOfStore(store_id, DemandTypeCode): DemandInfo[0...*]	List all demands of a specific store related to a specific demand type. *Parameter:* The store identification **(store_id)** and the demand type **(DemandTypeCode).** *Return value:* The list of demands **(DemandInfo [0…*]).**

2.3.2.2 Interface: MarketplaceInteractions

This interface is provided by the Marketplace Operation Manager and Table 9 depicts the methods provided by this interface. These methods are fairly sophisticated, and they can use methods provided by the MaketplaceConnection interface in order to expose their sophisticated functionalities.

Table 9 – Methods of MarketplaceInteractions Interface

Method	Description
Public requestSpotMarketInfo(planID, FilterInfo)	Request the ECM module to start the process of retrieving information from the marketplace so that can be used during the planning phase. *Parameter:* The plan identification **(planID)** and filter information

	(FilterInfo). *Return value:* None.
Public retrieveSpotMarketInfo(planID): MatchInfo[0…*]	Return the information requested from the marketplace for a specific filter and plan identification. *Parameter:* The plan identification **(planID).** *Return value:* The list of matches **(MatchInfo[0…*]).**
Public matchOfferToDemand(DemandInfo, FilterInfo): MatchInfo[0…*]	Provide the list of matched offers for a given demand. *Parameter:* The demand information **(DemandInfo)** and filter information **(FilterInfo).** *Return value:* The list of matches **(MatchInfo[0…*]).**
Public matchDemandToOffer(OfferInfo, FilterInfo): MatchInfo[0…*]	Provide the list of matched demands for a given offer. *Parameter:* The demand information **(OfferInfo)** and filter information **(FilterInfo).** *Return value:* The list of matches **(MatchInfo[0…*]).**
Public sendBidToDemand(BidInfo, DemandInfo)	Send a bid for a demand in the marketplace. *Parameter:* The bid information **(BidInfo)** and demand information **(DemandInfo).** *Return value:* None.
Public sendBidToOffer(BidInfo, OfferInfo)	Send a bid for an offer in the marketplace. *Parameter:* The bid information **(BidInfo)** and offer information **(OfferInfo).** *Return value:* None.
Public startAuction(MarketplaceInfo): auction_id	Indicate that an auction for a marketplace item started (either demand or offer). *Parameter:* The marketplace information **(MarketplaceInfo).**

	Return value: The auction identification **(auction_id)**.
Public closeAuction(MarketplaceInfo)	Indicate that an auction for a marketplace item is finished (either demand or offer). *Parameter:* The marketplace information **(MarketplaceInfo)**. *Return value:* None.
Public getWinnerOption(auction_id): MarketplaceInfo	Return the information of the winner of an auction process. *Parameter:* The auction identification **(auction_id)**. *Return value:* The information of the winner item **(MarketplaceInfo)**.
Public getRankedListOfBids(MarketplaceInfo, FilterInfo): MarketplaceInfo[0...*]	Return the list of bids ordered according to the requested filter. *Parameter:* The marketplace item receiving the bids **(MarketplaceInfo)** and the filter information to rank the bids **(FilterInfo)**. *Return value:* The ranked list of bids **(MarketplaceInfo[0...*])**.
Public establishContract(MarketplaceInfo, partner_id_provider, partner_id_consumer)	Create a short term contract based on the marketplace item information and the partners agreeing with the information. *Parameter:* The marketplace item **(MarketplaceInfo)**, the partner identification of the party providing the service **(partner_id_provider)** and the partner identification of the party consuming the service **(partner_id_consumer)**. *Return value:* None.
Public createOfferWatchList(partner_id)	Create a watch list for offers in which a given party is interested in. *Parameter:* The partner identification **(partner_id)**. *Return value:* None.
Public showOfferWatchList(partner_id): WatchListInfo	Provide the content of a watch list for offers

	in which a given party is interested in. *Parameter:* The partner identification **(partner_id)**. *Return value:* The watch list information **(WatchListInfo)**.
Public deleteOfferWatchList(partner_id)	Delete a watch list for offers in which a given party is interested in. *Parameter:* The partner identification **(partner_id)**. *Return value:* None.
Public insertItemInOfferWatchList(MarketplaceInfo, watch_list_id)	Insert in the offer watch list one item from marketplace in which the partner is interested in. *Parameter:* The interested marketplace information **(MarketplaceInfo)** and the watch list identification **(watch_list_id)**. *Return value:* None.
Public removeItemFromOfferWatchList(MarketplaceInfo, watch_list_id)	Remove from the offer watch list one item from marketplace in which the partner is interested in. *Parameter:* The marketplace information to be removed **(MarketplaceInfo)** and the watch list identification **(watch_list_id)**. *Return value:* None.
Public getItemFromOfferWatchList(watch_list_id): MarketplaceInfo[0…*]	Return the items inside the offer watch list. *Parameter:* The watch list identification **(watch_list_id)**. *Return value:* The marketplace information to be removed **(MarketplaceInfo)**.
Public createDemandWatchList(partner_id)	Create a watch list for demands in which a given party is interested in. *Parameter:* The partner identification **(partner_id)**. *Return value:* None.

Public showDemandWatchList(partner_id) : WatchListInfo	Provide the content of a watch list for demands in which a given party is interested in. *Parameter:* The partner identification **(partner_id).** *Return value:* The watch list information **(WatchListInfo).**
Public deleteDemandWatchList(partner_id)	Delete a watch list for demands in which a given party is interested in. *Parameter:* The partner identification **(partner_id).** *Return value:* None.
Public insertItemInDemandWatchList(MarketplaceInfo, watch_list_id)	Insert in the demand watch list one item from marketplace in which the partner is interested in. *Parameter:* The interested marketplace information **(MarketplaceInfo)** and the watch list identification **(watch_list_id).** *Return value:* None.
Public removeItemFromDemandWatchList(MarketplaceInfo, watch_list_id)	Remove from the demand watch list one item from marketplace in which the partner is interested in. *Parameter:* The marketplace information to be removed **(MarketplaceInfo)** and the watch list identification **(watch_list_id).** *Return value:* None.
Public getItemFromDemandWatchList(watch_list_id): MarketplaceInfo[0…*]	Return the items inside the demand watch list. *Parameter:* The watch list identification **(watch_list_id).** *Return value:* The marketplace information to be removed **(MarketplaceInfo).**

2.3.2.3 Interface: ManageExternalMarketplaceConnection

This interface is provided by the External Marketplace Manager and Table 10 depicts the methods provided by this interface. There are two major purposes of this interface. The first is to enable the management of the proxies connecting external marketplaces to the marketplace exposed by the ECM module. The second is to enable the management of the offers and demands associated with the external marketplace. Due to the fact that the Marketplace GE is only able to search for information in Linked-USDL format, we need to provide basic mechanisms to integrate the external information. Below you can find the most important activities that need to be supported in order to successfully integrate external marketplace information:

- translate the external offers and demands into Linked-USDL, and store them inside Finest Platform;
- maintain consistency between the translated information and the external information;
- import external information and export information created inside the Finest platform into the external marketplace (e.g., an offer is created inside FInest but needs to be published in an external T&L marketplace); and
- synchronize the local translations inside the FInest platform and the information from the external marketplaces (e.g., keep the same view of the offers in both places).

Table 10 – Methods of External Marketplace Connection Interface

Method	Description
Public insertProxy(ExternalMarketplaceInfo)	Insert a new proxy to the list of active proxies to be managed. *Parameter:* The external marketplace information **(ExternalMarketplaceInfo)**. *Return value:* None.
Public deleteProxy(ExternalMarketplaceInfo)	Remove a proxy from the list of active proxies to be managed *Parameter:* The external marketplace information **(ExternalMarketplaceInfo)**. *Return value:* None.
Public monitorProxyConnectivity(ExternalMarketplaceInfo)	Monitor if the proxy connectivity is active. *Parameter:* The external marketplace to be monitored **(ExternalMarketplaceInfo)**.

	Return value: None.
Public listCurrentProxies(): ExternalMarketplaceInfo[0…*]	List all the active proxies. *Parameter:* The external marketplace information **(ExternalMarketplaceInfo).** *Return value:* None.
Public pullExternalMarketplaceInfo(ExternalMarketplaceInfo)	Import information from external marketplace into FInest internal marketplace. *Parameter:* The external marketplace information **(ExternalMarketplaceInfo).** *Return value:* None.
Public pushLocalInfoIntoExternalMarketplace(ExternalMarketplaceInfo, MarketplaceInfo[1…*])	Export information from FInest marketplace into the external marketplace. *Parameter:* The external marketplace information **(ExternalMarketplaceInfo)** and the list of marketplace information to be exported **(MarketplaceInfo[1…*]).** *Return value:* None.
Public convertExternalInfo(ExternalMarketplaceInfo, MarketplaceInfo [0…*]): MarketplaceInfo [0…*]	Convert information from the external marketplace into Linked-USDL format. *Parameter:* The external marketplace information **(ExternalMarketplaceInfo)** and the list of marketplace information to be converted **(MarketplaceInfo[1…*]).** *Return value:* The list of converted marketplace information **(MarketplaceInfo[1…*]).**
Public synchronizeLocalInfoWithExternalMarketplaceInfo(ExternalMarketplaceInfo)	Synchronize local and external information associated with a specific marketplace. *Parameter:* The external marketplace information **(ExternalMarketplaceInfo).** *Return value:* None.

Public synchronizeAllLocalInfo()	Synchronize local and external information associated for all external marketplaces integrated into FInest platform. *Parameter:* None . *Return value:* None.

2.3.3 Interfaces related to Connection to UI Component

In this section, all described interfaces are related to the Connection to UI component. Figure 6 introduces the class diagram characterizing each of the interfaces and their respective methods. It is important to notice that the methods defined by these interfaces still need to be aligned with the requirements for presenting the information according to the specifications of the UI in the FInest Frontend module. Due to this fact, we do not present in this stage of the specification all the parameters and return values required for each method. The FInest Frontend module, while not being fully specified in phase I of the project, is a key module in the phase II follow on project cSpace.

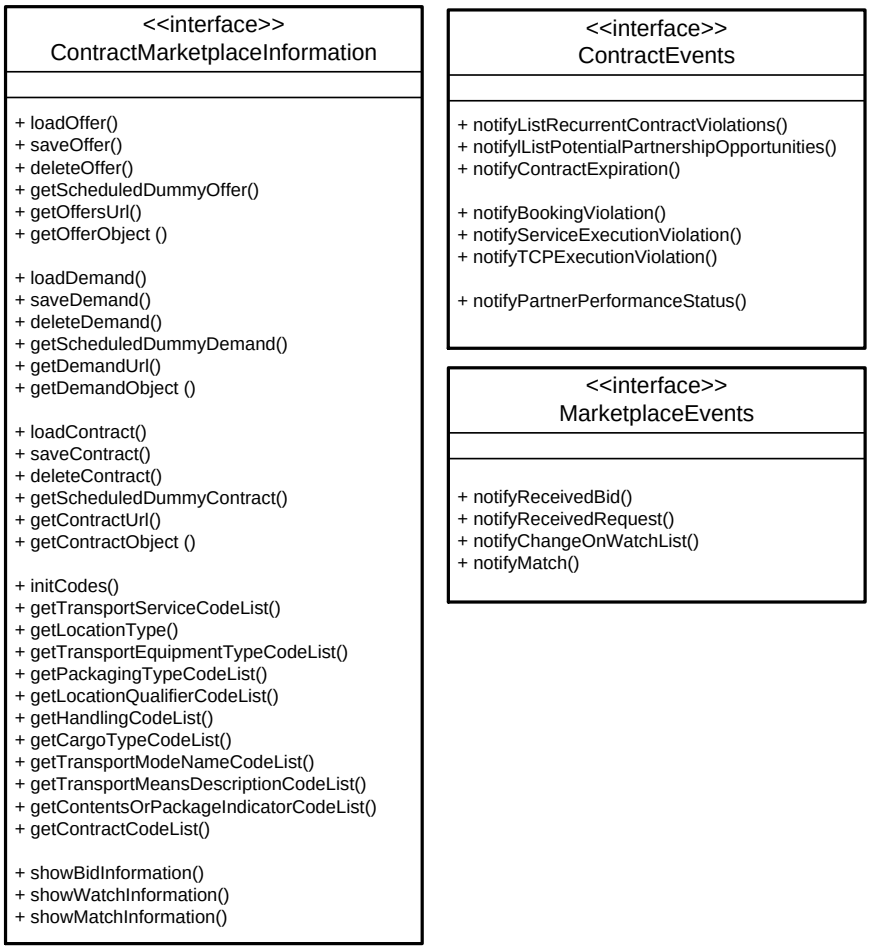


Figure 6 – Provided Interfaces related to Connection to UI Component

2.3.3.1 Interface: ContractMarketplaceInformation

This interface is provided by the Information Visualization Controller and Table 11 depicts the methods provided by this interface. The major purpose of this interface is to enable the flow of information from the ECM module into the UI components of the FInest Frontend module. All types of visualization of content are addressed by the methods of this interface.

Table 11 – Methods of ContractMarketplaceInformation Interface

Method	Description
`Public loadOffer()`	Trigger download and parsing of a service offer. The result will be an object (available through method getOfferObject) which allows simple access to all the data of the offer – no RDF knowledge is necessary at all.
`Public saveOffer()`	Takes the simplified service offer object, creates an RDF file form it and triggers the upload of that file to the given URL.
`Public deleteOffer()`	Delete the service offer with the given URL
`Public getScheduledDummyOffer()`	Loading a dummy service offer for simplified or offline testing.
`Public getOffersUrl()`	Loading a collection of offers from the repository.
`Public getOfferObject ()`	Easy way to get just the offer object after it has been downloaded.
`Public loadDemand()`	Trigger download and parsing of a service demand.
`Public saveDemand()`	Takes the simplified service demand object, creates an RDF file form it and triggers the upload of that file to the given URL.
`Public deleteDemand()`	Delete the service demand with the given URL
`Public getScheduledDummyDemand()`	Loading a dummy service demand for simplified or offline testing.
`Public getDemandUrl()`	Loading a collection of demands from the repository.
`Public getDemandObject ()`	Easy way to get just the demand object after it has been downloaded

Public loadContract()	Trigger download and parsing of a contract.
Public saveContract()	Takes the simplified contract object, creates an RDF file form it and triggers the upload of that file to the given URL.
Public deleteContract()	Delete the contract with the given URL
Public getScheduledDummyContract()	Loading a dummy contract for simplified or offline testing.
Public getContractUrl()	Loading a collection of contracts from the repository.
Public getContractObject ()	Easy way to get just the contract object after it has been downloaded
Public initCodes()	Initializes (download + parsing) of all related code vocabularies. They are afterwards available using the corresponding methods.
Public getTransportServiceCodeList()	Loading Transport Service Codes as Array.
Public getLocationType()	Loading Location Type Codes as Array.
Public getTransportEquipmentTypeCodeList()	Loading Transport Equipment Codes as Array.
Public getPackagingTypeCodeList()	Loading Packaging Codes as Array.
Public getLocationQualifierCodeList()	Loading Location Codes as Array.
Public getHandlingCodeList()	Loading Handling Codes as Array.
Public getCargoTypeCodeList()	Loading Cargo Type Codes as Array.
Public getTransportModeNameCodeList()	Loading Transport Mode Codes as Array.
Public getTransportMeansDescriptionCodeList()	Loading Transport Means Codes as Array.
Public getContentsOrPackageIndicatorCodeList()	Loading Package or Content Codes as Array.
Public getContractCodeList()	Loading contract codes as Array.
Public showBidInformation()	Enable the bid information from the marketplace to be presented in the UI.
Public showWatchInformation()	Enable the watch list information from the marketplace to be presented in the UI.

`Public showMatchInformation()`	Enable the match information from the marketplace to be presented in the UI.

2.3.3.2 Interface: ContractEvents

This interface is provided by the Event Notification Controller and Table 12 depicts the methods provided by this interface. The major purpose of this interface is to enable the flow of notifications related to contract information into the UI components of the FInest Frontend module.

Table 12 – Methods of ContractEvents Interface

Method	Description
`Public notifyListRecurrentContractViolations()`	Enable the information related to recurrent contract violation to be notified in the UI.
`Public notifyContractExpiration()`	Enable the information related to contract expiration to be notified in the UI.
`Public notifyBookingViolation()`	Enable the information related to booking violation in regards to the contract to be notified in the UI.
`Public notifyServiceExecutionViolation()`	Enable the information related to contract violations during a service execution to be notified in the UI.
`Public notifyPartnerPerformanceStatus()`	Enable the information related to partner performance to be notified in the UI.

2.3.3.3 Interface: MarketplaceEvents

This interface is provided by the Event Notification Controller and Table 13 depicts the methods provided by this interface. The major purpose of this interface is to enable the flow of notifications related to marketplace information into the UI components of the FInest Frontend module.

Table 13 – Methods of MarketplaceEvents Interface

Method	Description
`Public notifyReceivedBid()`	Enable the information related to received bids to be notified in the UI.
`Public notifyReceivedRequest()`	Enable the information related to received requests to be notified in the UI.

`Public notifyChangeOnWatchList()`	Enable the information related to changes in the watch list to be notified in the UI.
`Public notifyMatch()`	Enable the information related to matches to be notified in the UI.

2.4 Data Types Definitions

This section presents the definition of the complex data types used by the specified interfaces of the ECM module. In some of the cases, a more complex information type is required in order to encapsulate the whole content associated with the target data. In the ECM module, three main groups of data types are needed. First, we need data types that define specific codes for the possible types of contracts, representation of data structures and types of marketplace items. The details about this group are presented in Section 2.4.1. Second, we need specific data types to handle the information of contracts and this is described in Section 2.4.2. Finally, the ECM also needs more complex information to handle the functionalities related to marketplace operations and Section 2.4.3 describes such data types.

The ECM module is designed to handle contract and marketplace operations. These types of operations are not only required by the transport and logistics domain, but they can also be used by many different types of industries. Having this focus in mind, we designed the ECM module to support the transport and logistics domain needs, and at the same time to be able to be extended for any other type of industry. For this reason, many of the data type definitions already consider the flexibility for extending such types in order to include different data models, representation, etc.

2.4.1 Codes of Data Types

A set of initial codes is defined in the ECM module and can be extended and refined. Figure 7 shows the current list of codes types and their respective initial set of content. A class diagram representation has been used to illustrate these codes.

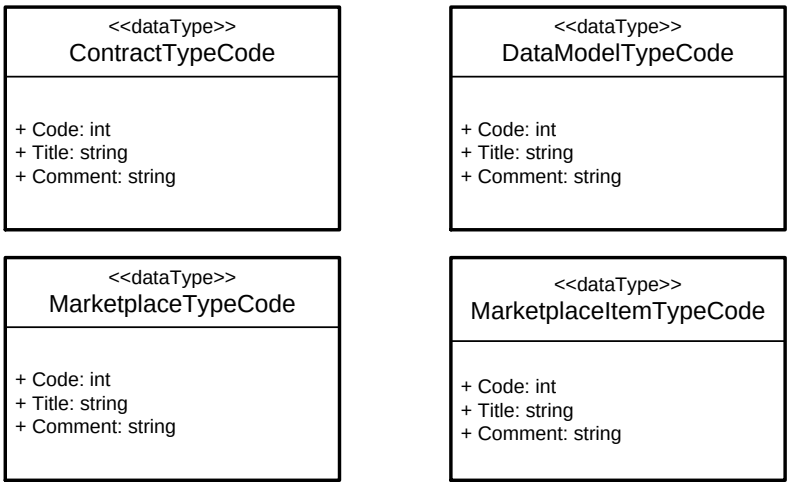


Figure 7 – Class diagrams of useful codes for the ECM module

All the code data types defined gave the same attributes. The use pseudo-codes "Code", "Title", and "Comment" enables a very high degree of extensibility on the different types of codes. Using the pseudo-codes there is no need to hard code all types in advance, but we define which will be the characteristics of the codes so that they can be easy expended.

- **ContractTypeCode** – Defines the supported types of contracts managed by the ECM module. Currently, four types of contracts are defined and all of them are related to transport and logistics operations: blanket, tariff based, capacity based, and spot contract. This list is not closed and can be expanded. For example, it there are other types of contracts, even within the transport and logistics domain (such as performance based contracts), they can be added to the scope of the ECM.
- **DataModelCode** – For the current scope of the FInest project, we consider only two types of data to represent the contract information and also interact with the TPM module: Linked-USDL and the TSD representations. However, external marketplaces will have different data formats and the components of the ECM module will have to deal with a diversity of data models. The DataModelCode will be able to represent this diversity.
- **MarketplaceTypeCode** – Within the FInest project, we will consider only transport and logistics marketplaces, but there is no restriction on the ECM specification to extend its scope to other types of marketplaces. This data type code is defined in order to identify which kind of marketplaces is supported by the ECM.
- **MarketplaceItemTypeCode** – Currently in the ECM specification, transport and logistics offers and demands are modelled according to the Linked-USDL data model. However, during the definition of concrete proxies to integrate different marketplaces there will be the need to understand different data formats and also different types of marketplace items. This code is one of the elements that needs to be considered to support such diversity in the ECM module.

2.4.2 Contract Related Data Types

This section focuses only on the data types used for the interfaces associated with the Contract Operations and Contract Analytics components. Figure 8 illustrates these data types.

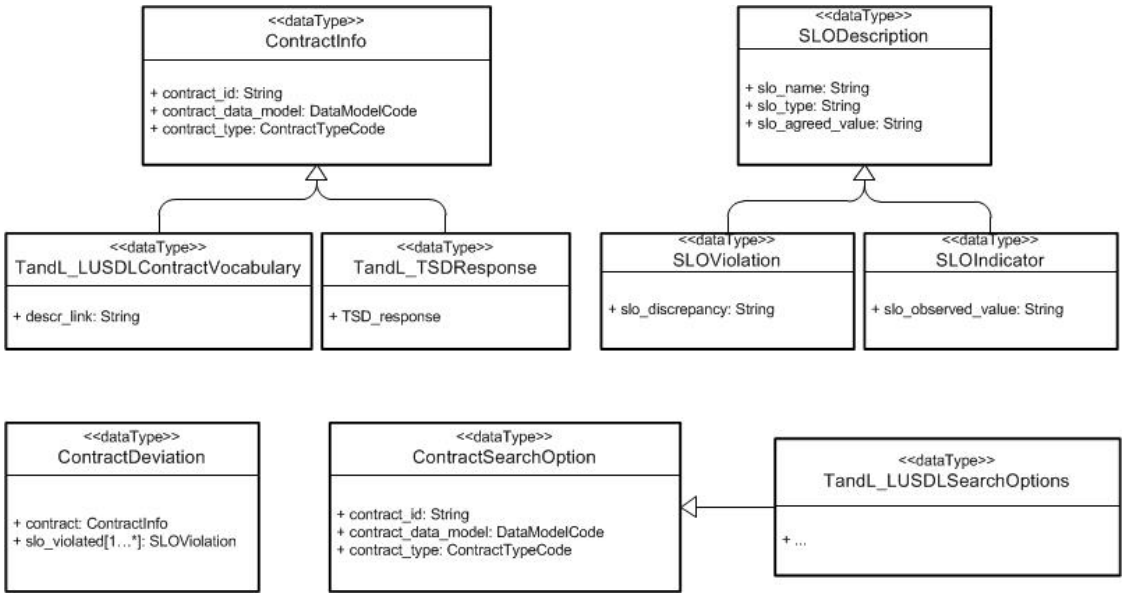


Figure 8 – Class Diagram of Data Types related to the Contract Interfaces

- **ContractInfo** – This data type encloses the basic information to characterize the contract being considered by a method. As presented in Figure 8, ContractInfo is the super class and it has currently two classes that inherent its attributes. Using the ContractInfo data type, the methods signature is flexible enough to handle different types of contracts.
- **TandL_LUSDLContractVocabulary and TandL_TSDResponse –** These are data types that share the same attributes of ContractInfo and each one includes its own set of attributes. For example, the TandL_LUSDLContractVocabulary data type has attributes that are only relevant for transport and logistics contracts represented in a Linked-USDL model.
- **SLODescription** – This data type contains the basic information that characterizes a service level object, and it can represent any information.
- **SLOViolation and SLOIndicator –** These are subclasses from SLADescription, and they inherit all the attributes of the super class and specify the attributes that are particular to each one of them.
- **ContractDeviation –** Encloses the basic information that characterizes the contract and the types of deviations associated with such a contract.
- **ContractSearchOption** – Different types of contracts will require different types of filtering fields. This super class defines the very basic information to define a search option and the child data types specify the details according to each contract model type. In Figure 8 we indicate the existence of specific search options for Linked-USDL representation of contract information.

2.4.3 Marketplace Related Data Types

This section focuses only on the data types used by the interfaces associated with the Marketplace Operations component. Figure 9 illustrates these data types.

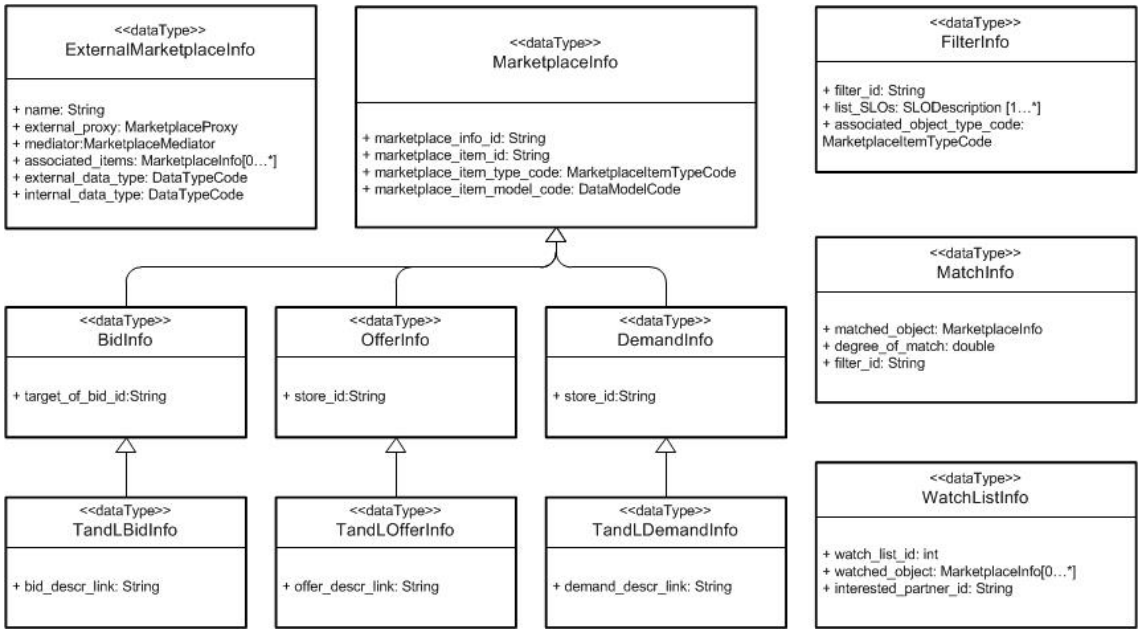


Figure 9 – Class Diagram of Data Types related to the Marketplace Interfaces

- **ExternalMarketplaceInfo** – This data type is used to define the basic information of external marketplaces. Important attributes of this data type are the marketplace proxy to which the external market place is associated and the mediator, which converts the information from the external marketplace format to the format used within the ECM module.
- **FilterInfo, MatchInfo, and WatchListInfo** – These are data types to support sophisticated operations on the marketplace. They can be seen as auxiliary data types to host the information filters to query the marketplace, information about matches between offer and demand, and the lists of marketplace items in which a party is interested.
- **MarketplaceInfo, BidInfo, OfferInfo, DemandInfo** – These data types are very important for supporting flexibility in the marketplace operations. The ECM module is designed to support not only transport and logistics marketplace operations, but also other types of industries and interactions with their marketplaces. The MarketplaceInfo data type is the parent class of specific marketplace items such as bid (represented by BidInfo data type), offer (represented by OfferInfo data type) and demands (represented by DemandInfo data type). The list of currently represented marketplace items can be extended. In addition, depending on the industry, each one of the marketplace items will have a different set of attributes to be considered. For this, we defined specific data types for TandLBidInfo, TandLOfferInfo and TandLDemandInfo that inherit all information from the parent classes.

2.5 Sequence Diagram: ECM Interactions with other Core Modules

In the Finest project, four core technical modules (TPM, BCM, ECM and EPM) are designed to interact with one another to leverage their combined functionalities. A first alignment of the interactions among the core technical modules was described in the deliverables D3.2 and D8.3. In the last one, we introduced the initial sequence diagram to show the interactions of the ECM. Now, in this report we advance the refinement of these interactions. The updated version of this sequence diagram is illustrated in both Figures Figure 10 and Figure 11.

This section focuses on the illustration of the interactions first between the ECM and TPM as illustrated in Figure 10 and among the ECM, TPM, and BCM as illustrated in Figure 11. These two core technical modules, i.e., TPM and BCM, are the ones with which the ECM is directly related. The overall interactions among the core modules could be found in deliverable D3.4.

2.5.1 Interactions before the Execution of Transport and Logistics Services

As depicted in Figure 10, there is a block of interactions that can happen **between the ECM and the TPM** core modules in order to exchange the necessary information and execute the proper actions **before the execution of a transport and logistics service**. This block of interactions is represented by the loop block. For example, in order to define a plan for executing a transport and logistics service the TPM module can interact with the ECM module in order to retrieve information from:

- established contracts as represented in the first optional block, i.e., the “opt” block in Figure 10;
- offers inside marketplaces matching the needs of the transport and logistics plan as represented in the second optional block, i.e., the “opt” block in Figure 10; and
- can also retrieve information from both optional blocks of interactions.

Furthermore, the ECM and the TPM can also interact after the plan is consolidated. In this case, before going for the execution of the consolidated plan, the TPM module can optionally check if the SLAs of the planned service are compliant with the contracts associated with the plan. This interaction is illustrated in the third option block, i.e., the “opt” block in Figure 10. The compliance check of booked SLAs of a plan is required because during the booking process many negotiations happen until the client and provider have an agreement on how they can really execute the transportation. Thus, this booking negotiation process is prone to have the SLA values of a transportation service deviate from the actual values in the contract.

Checking the booked plan can avoid, or at least create awareness, that the execution of the booked transport and logistics service will have deviations with respect to the contract associated with it. This could mean, for example, extra financial charges for some of the partners. With the interactions between the ECM and TPM modules, the parties can be aware of this situation even before the actual execution of the service.

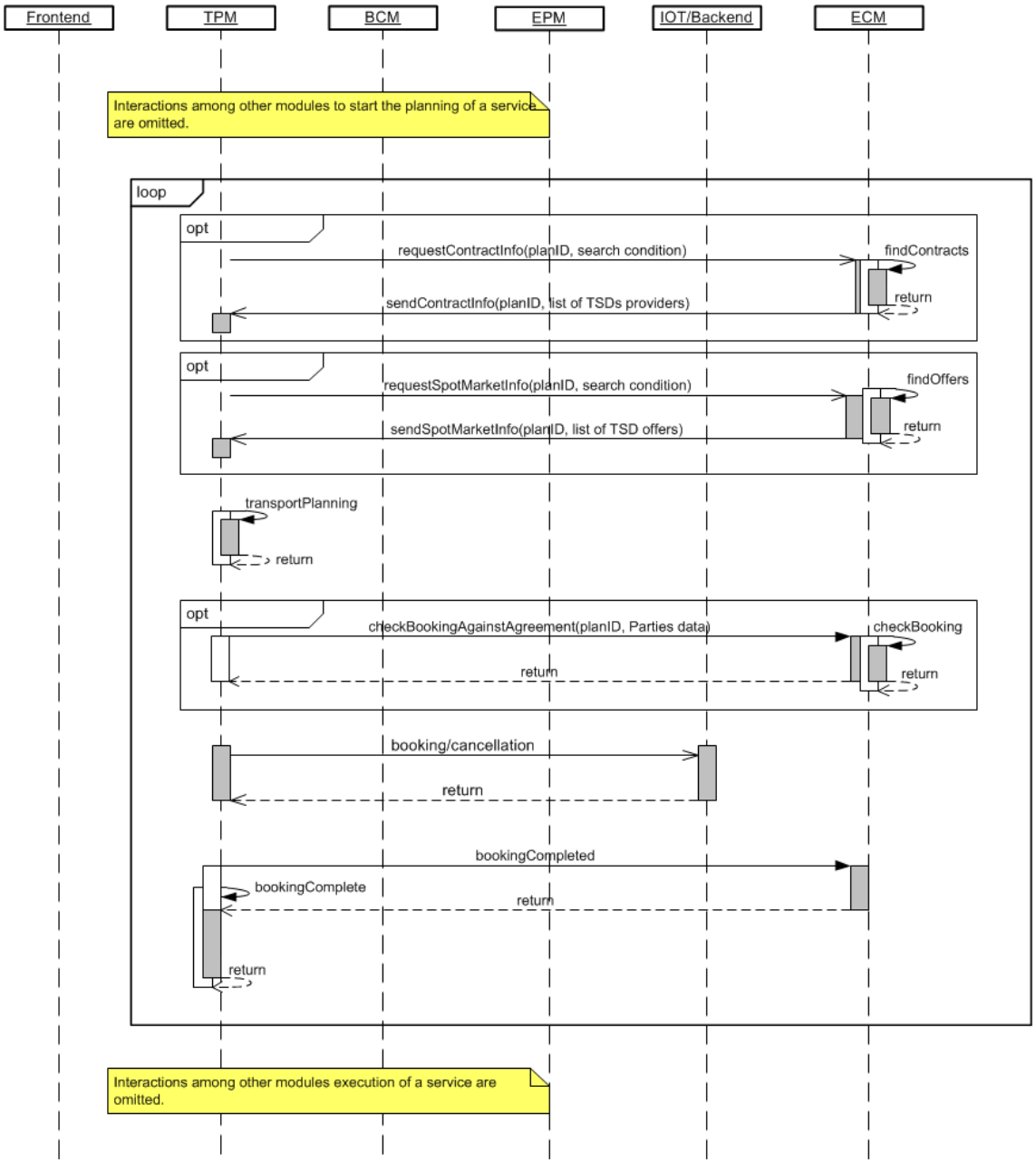


Figure 10 – Sequence diagram of interactions between ECM and TPM modules before the execution of a transport and logistics service

In the last interaction at this stage, the TPM informs the ECM of final agreed and booked information associated with the service being prepared. This information is essential in the case of services being executed under long term contracts with aggregated attributes, i.e., values that should be updated during the lifetime of the contract. The ECM must update the long term contracts attributes and maintain consistency between what happens in the daily operation and what is established in the contracts.

2.5.2 Interactions during the Execution of Transport and Logistics Services

In addition to the previous described interactions, the **ECM module can interact with the TPM and the BCM during the execution of a transport and logistics service.** There are two situations during the execution of a service in which the ECM interact with other modules as illustrated in Figure 11.

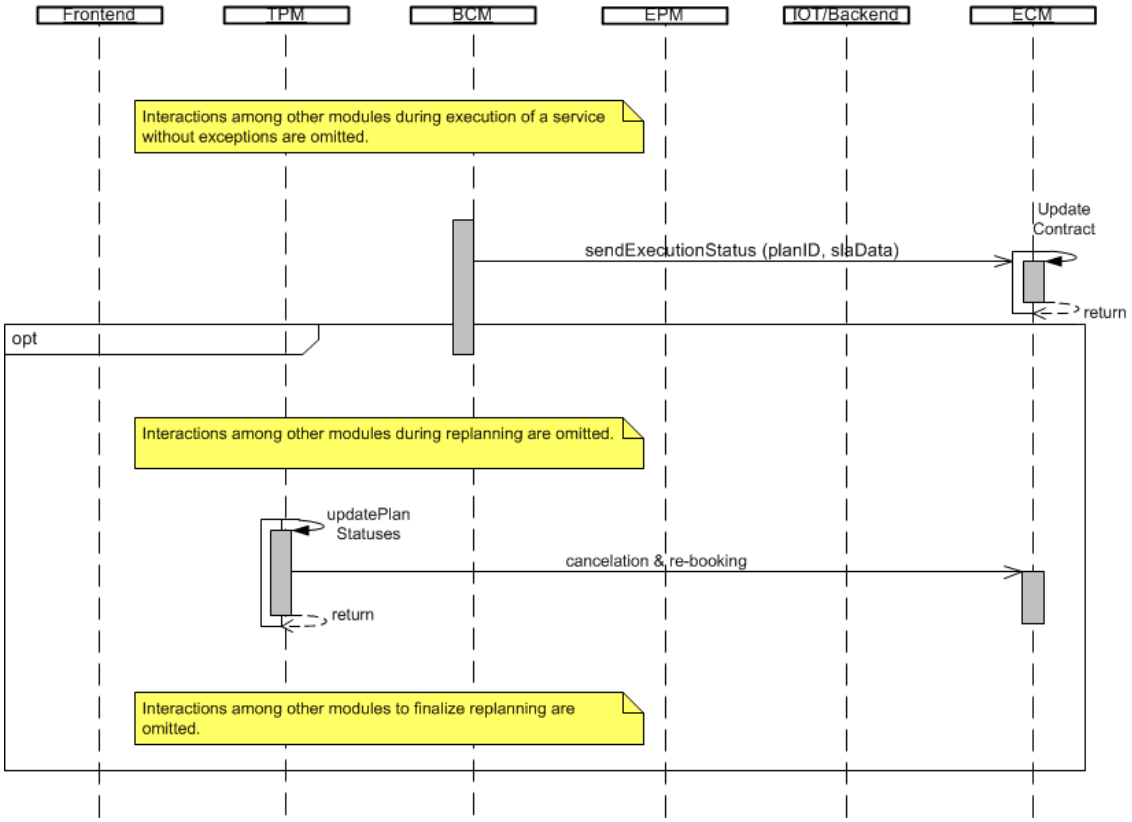


Figure 11 – Sequence diagram of interactions between ECM and other modules during the execution of a transport and logistics service

The first situation occurs right after a completion of a transport and logistics service in which a replanning of the execution was not necessary. In this case, although a replan was not required, different types of deviations could have happened, and this has to be checked against the contract SLAs. For example, during the pick-up of the goods the shipper delivers more goods than was booked. Thus, even if the booked plan is compliant with the contract information, there is also the chance that deviations from the booked SLAs happen during the actual execution of the plan. The interaction between the ECM and BCM is designed to handle this type of situation. The BCM can contact the ECM module and indicate that a transport and logistics service has finished and can be checked. The process of checking the compliance of the SLA values of an executed transport and logistics service is then executed within the scope of the ECM.

The second situation is during the replanning of an ongoing execution of a transport and logistics service. In this case, eventual booking cancelations can happen as well as new bookings for the same service. This information has to be available to the ECM, so that it can perform updates on the contracts and compliance checks on the transport plan. To conduct these actions, the TPM interacts with the ECM in order to inform the latter about the cancelations and re-bookings during the replanning phase.

It is important to notice that all these interactions among the modules are not mandatory. The ECM module is designed to provide the aforementioned functionalities for any kind of software component that "wants" to interact with the ECM module. The sequence diagrams presented in this section use as an example the TPM and BCM as software components interacting with the ECM. Nevertheless, the style of interaction will be the same for any other software component willing to use the functionalities described in these sequence diagrams.

3 Final Data Model Specifications

This section contains descriptions of the three different models/vocabularies developed in WP8. These are the Linked USDL vocabulary for transport and logistics service offers, the Linked Data vocabulary for transport and logistics service demands and the class diagram for contract models. For each of these models/vocabularies a subsection has been created in which the model/vocabulary will be described.

3.1 Linked USDL Vocabulary for T&L Service Offers

The initial specification of the transport and logistics service offer was reported D8.3. After refinements and validation with the domain partners, we updated the specification. Some elements on the data model were removed and others better specified, but the overall structure of service offers in transport and logistics remains the same. As previously described, we extended the core Linked USDL (Unified Service Description Language using the Linked Data approach [2]) vocabularies defined by FI-WARE in order to create a description of transport and logistics service offers. During this work the collaboration with experts in Linked-USDL from FI-WARE was essential to enable a better refinement of the data model.

Figure 12 shows a RDF graph for the final version of the transport and logistics service offer vocabulary developed in FInest between M18 and M24. With RDF/Linked Data, it is common to describe vocabularies using RDF graphs[1] that are a kind of visualization of such vocabularies. Besides Linked USDL, the vocabulary uses several existing and established vocabularies such as GoodRelations[2], GeoNames[3] and Dublin Core Terms[4]. Because of the linking approach of Linked USDL, all classes from Linked USDL can be used with the transport and logistics service offer vocabulary. In Figure 12 classes reused from Linked USDL are orange while parts reused from other vocabularies are blue. Classes in green and purple are defined in FInest. The purple classes represent existing transport and logistics codes such as the *UNECE Package Type Code* or the *UN/CEFACT 8275 Container* or *Package Contents Indicator Code*.

Instead of describing in detail every term defined in the offer service vocabulary, we divided the vocabulary into parts. The expanded visualization of each one of the service offer parts can be found in Appendix A – Parts of Service Offer Vocabulary. Figure 12 illustrates the parts (A-G) of the graph highlighting them. In this document only those parts are described and an overview is given about the information that can be stored in the parts. In Figure 12, the labels of relationships in the RFD graph in red indicate the changes introduced in the final version of the data model specification of the transport and logistics service offer. The description of the parts is discussed as follows.

[1] http://www.w3.org/TR/rdf-concepts/#section-data-model
[2] http://www.heppnetz.de/projects/goodrelations/
[3] http://www.geonames.org/
[4] http://dublincore.org/

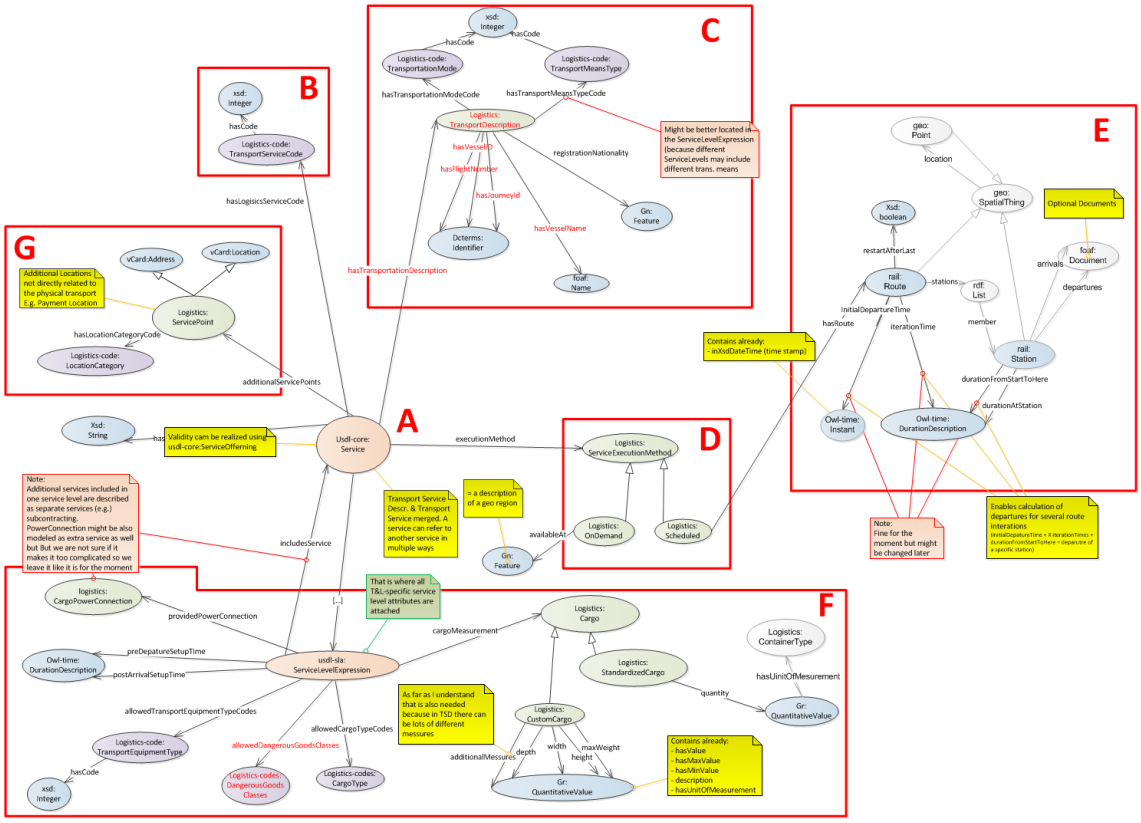


Figure 12 – RDF Graph describing the T&L Service Offer Vocabulary based on Linked-USDL

Part A – Linked USDL service class: The basis for the entire transport service offer vocabulary is the service class that is part of the existing Linked USDL vocabulary. Through this connection all classes existing in Linked USDL can be used for the description of transport and logistics service offers. These include, for example, the description of pricing schemas or the description of service providers. Almost all transport and logistics related extensions are attached to the service class so it is the main touch point between Linked USDL (i.e., FI-WARE core vocabularies) and the transport and logistics extensions (specific domain issues developed in FInest).

Part B – Transport service type: In the transport and logistics domain many different types of services can be offered such as warehousing, freight forwarding, agent services or basic transportation. To take into account these different service types, a transport service code (specified in the FP7 e-Freight project [1]) needs to be specified for each transport and logistics service offer. However, the vocabulary currently only supports services of the type *"Transport"* and it is expected that for other service types additional extensions are needed. These will be developed in the phase II follow on project cSpace.

Part C – Transport mode and transport means: Part C describes mainly the transportation mode, which can be maritime, land, rail and air. Common transport service offers will only specify one transportation mode while freight forwarding services, which can be multimodal, will specify multiple modes. In addition, in part C the transport means is described. Each service can specify one or multiple vehicle types that are available for the transportation service.

Possible vehicle types are provided by the Transport Means Type Code (UN/CEFACT Trade Facilitation Recommendation No. 28).

Part D – Service execution mode: In this part it can be specified if a service will be executed only on demand (typically truck transports) or on a scheduled basis (e.g., container vessel transport). This is of special importance for rail, air and maritime transport because both execution modes are present in these domains. If the service will be executed on a scheduled basis it is possible to also maintain a route (see part E). If the service will be executed on demand it is only possible to give one or more areas in which the service is active. This area could be a single region or a list of countries.

Part E – Route of a scheduled transport service: For scheduled transport services a route can be maintained with stops and arrival/departure timetables. Stops are locations so they can have an address or GPS coordinates attached. For each stop one timetable document for arrivals and one for departures can be attached. In addition, a concept has been developed that makes it possible to calculate arrivals and departures for each stop of a route for an arbitrary number of iterations.

Part F – T&L-specific service levels: Linked USDL service offers can always contain several so called service level profiles that are used to specify different variants of the service with different features and quality of service (QoS), but that can have a different price or pricing model. However, in Linked USDL the description of features is very generic, which, on one hand, enables the LSP to describe almost any feature, but on the other hand makes it very difficult for a customer to compare services because the same features may not be described in the same way. For this reason it has been decided to implement native feature descriptions in the Transport and logistics specific extension instead of using the generic ones. So far the following features can be described:

- List of cargo types that can be transported (based on UN/CEFACT 7085 Nature of cargo)
- List of transport equipment types allowed for the transport (based on UN/CEFACT 8053 Equipment type code qualifier)
- Duration the cargo needs to be at the pickup point before the actual transport can be executed (to prepare the cargo for the transport)
- Duration until the cargo can be pickup after the arrival (because of post-processing)
- Availability of a power connection for the cargo or container (e.g. needed for a temperature-controlled containers)
- Description of the maximum cargo measurements (e.g. maximum weight and height) or container quantity if standard containers will be transported

In addition to the transport and logistics specific feature descriptions, the generic concept of Linked USDL can be used for transport service offers if additional features need to be described.

Part G – Additional service points: In TSD single locations, called service points, can be described. These are either directly (e.g., place of loading) or indirectly (e.g., payment location) related to the physical transport. Transport service offers modelled in Linked USDL are not related to specific requests, which mean that locations directly related to the physical transport may not be able to be described – at least for services executed on demand. However, indirectly related locations, such as the payment location, can still be described in part G of the extension.

As promised in D8.2 the vocabulary has been aligned to the TSD model but it has been found that only parts of it are needed. The reason for this is that the TSD has been developed differently than Linked USDL. With Linked USDL service offers will be published on an electronic marketplace without knowing the concrete demands of potential customers. That means that only abstract information is stored like a price of 100€ per 100km. In contrast, with TSD the customer sends a so called TSD Request, which can be understood as transportation demand, to the marketplace and then receives TSD Responses that are concrete offers from TSPs. Those include service information for that specific request like a price of 900€ for the requested transportation demand. From a TSP perspective the TSD approach could lead to extra manual effort because TSD Responses are always related to one TSD Request while one service offer described in Linked USDL can be used by many customers having different transportation demands. For that reason, only information which fits the purpose of publishing the T&L service offer directly on an electronic marketplace has been extracted from the TSD to create the Linked USDL extension and information only relevant for one TSD Request has either been not considered or generalized to fit the purpose. We already introduced updates in the current version of the vocabulary but changes may still be needed for the refinement of the model during the work to be done in phase II.

3.2 Linked Data Vocabulary for T&L Service Demand

Similar to the RDF graph for the transport and logistics service offer vocabulary based on Linked USDL (see section 3.1), a RDF graph has been modeled for the transport and logistics service demand vocabulary. Figure 13 shows the final version of the demand RDF graph.

In contrast to the transport and logistics service offer vocabulary, the transport and logistics service demand vocabulary is not based on Linked USDL. This decision has been made because the major part of Linked USDL vocabulary is related solely with the provider side announcing its offer, and there is no support in the Linked USDL vocabulary for the client description of its demand. In WP8 we introduced the idea of clients also being able to describe what they are looking for, i.e., what are their transport and logistics service demands. Thus, in WP8 a lean and independent vocabulary has been created for this purpose. Similar to the transport and logistics service offer vocabulary, the transport and logistics service demand vocabulary reuses existing RDF vocabularies such as GoodRelations and GeoNames, which are marked with a blue background color in Figure 13. Additionally, classes in green and purple are completely new and have been designed in WP8. Similar to Figure 12 purple classes represent the integration of existing logistics codes that are also implemented in RDF. The detailed visualization of each of the parts of the demand vocabulary can be found in Appendix B – Parts of Service Demand Vocabulary.

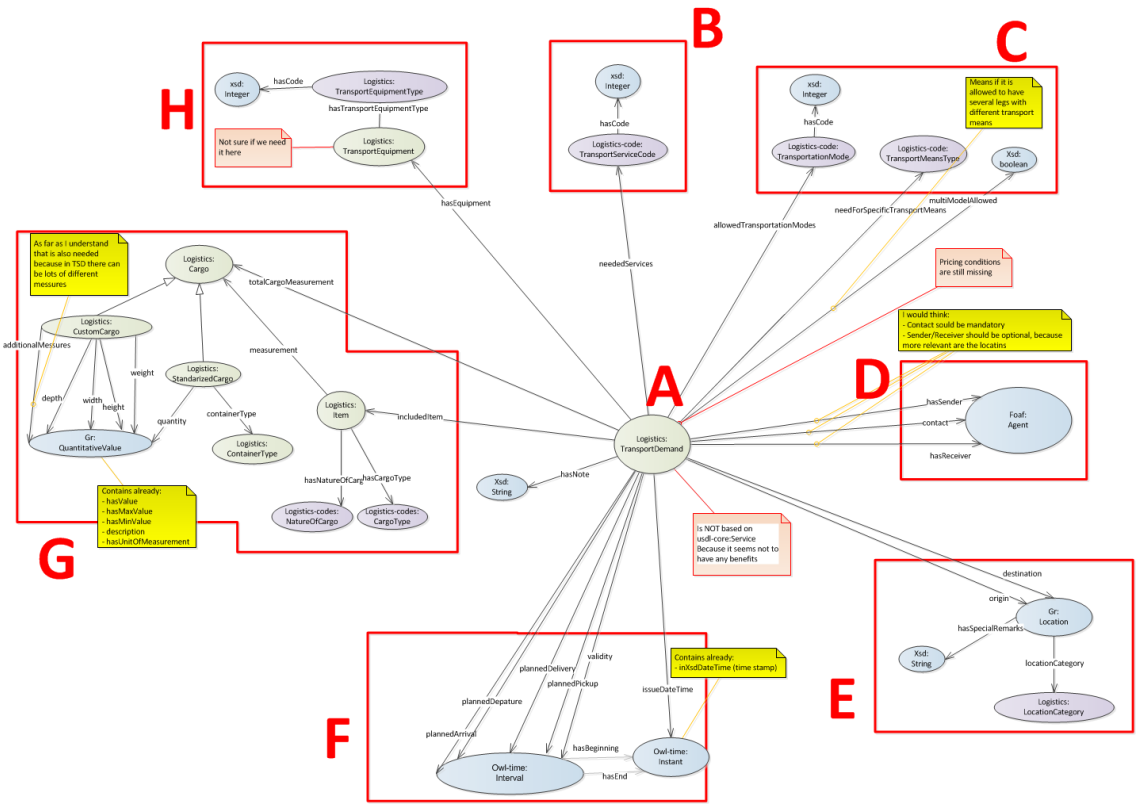


Figure 13 – RDF Graph describing the T&L Service Demand Vocabulary

In Figure 13 several sections of the RDF graph are highlighted and will be described briefly in the following paragraphs:

Part A – Demand Root Class: For the transport and logistics service demand a new root class has been defined to which all other information is attached. It has been found that it has no benefits to reuse Linked USDL as the basis for this vocabulary because almost none of the information stored in it is relevant for a transport and logistics service demand.

Part B – Transport service type: In the transport and logistics domain many different types of services can be offered such as warehousing, freight forwarding, agent services or basic transportation. To enable customers to create service demands for these different service types, a transport service code (specified in the FP7 e-Freight project) needs to be specified for each transport and logistics service demand. However, the transport and logistics service demand vocabulary only supports services of the type "Transport" and it is expected that for other service types additional extensions may be necessary.

Part C – Transport mode and transport means: Part C describes the requested transport modes and requested transport means. One or more of both kinds of requested services can be stored for a transport and logistics service demand. Transport modes are maritime, land, rail and air while transport means are vehicle types like container vessels or different kinds of trucks. Since it might not be relevant to the customer what transport means or transportation modes are used, they are optional for a demand.

Part D – Parties: This part can contain information regarding the consignor and the consignee of the goods to be shipped. However, because this information might be confidential these are optional. The origin and destination locations are stored in part E; thus, there is no issue if these parties are not stored in the transport and logistics service demand. The only mandatory party to be stored for a transport and logistics service demand is the contact party, which is equal to its creator.

Part E – Locations: in this part two locations need to be maintained – the origin and destination. Clearly these are very important for a transportation demand; therefore, they are mandatory.

Part F – Times: this part contains several times that can be specified and that are directly taken from a TSD Request. These are:

- Planned Pickup (time range)
- Planned Departure (time range)
- Planned Arrival (time range)
- Planned Delivery (time range)
- Validity of the service (time range)
- Issue Date (single timestamp)

Part G – Cargo, Cargo Measurement and Cargo Types: For a transport demand the cargo can be specified in several ways. First, it is possible to specify the total measurements of the entire cargo. Second, it can be specified that a certain number of standardized containers need to be shipped for which measurements are standardized. Third, multiple single items can be specified and measurements for them can be stored. In addition, for the single items, a cargo type can be attached that is very important for a LSP who is looking for transport demands.

Part H – Transport Equipment: finally, it is possible to attach a list of transport equipment that is used together with the cargo. This might be a container, a postal bag or a roll pallet, for example. Possible transport equipment types are specified in the Transport Equipment Type Code (UN/CEFACT 8053 Equipment type code qualifier).

3.3 Linked USDL Contract Model

In this section we will explain the overall information model used by the ECM to represent the most relevant information of contracts that need to be manipulated in an electronic form. This model was inspired by the TEP, TSD models from e-Freight [1, 3]. Following the same approach of the previous section, we illustrate the contract vocabulary using a RDF graph. The classes in green and purple are defined in FInest. The purple classes represent existing transport and logistics codes as previously detailed. The blue classes represent existing vocabularies, such as GoodRelations[5] and vCard[6] vocabularies, that are re-used in the contract vocabulary. The

[5] http://www.heppnetz.de/ontologies/goodrelations/v1
[6] http://www.w3.org/Submission/vcard-rdf/

visualization of each one of the parts of the contract can be found in Appendix C – Parts of Contract Vocabulary.

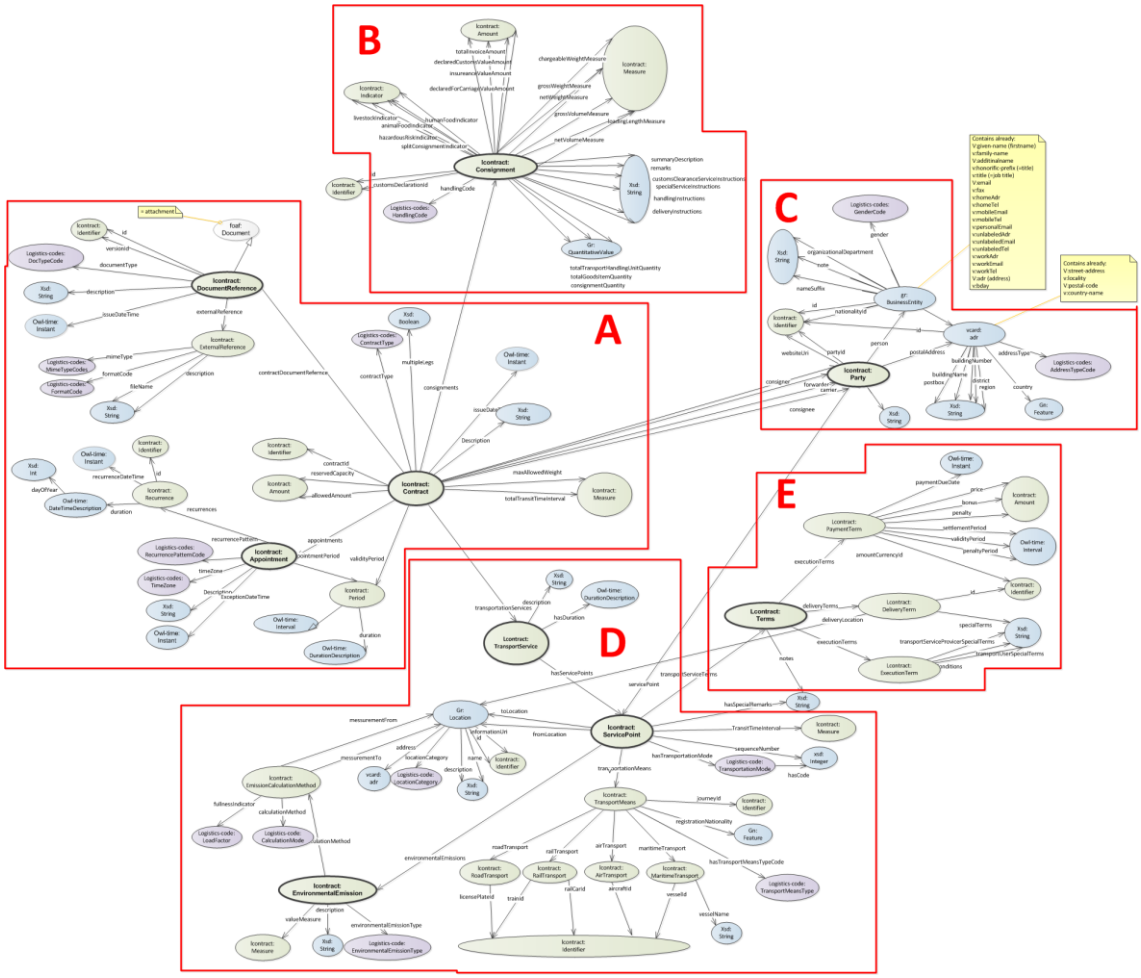


Figure 14 – RDF Graph describing the T&L Service Demand Vocabulary

Part A – Basic information about contracts: The basis for the entire transport and logistics contract vocabulary is the Contract class. We defined this new class due to the fact that it was not previously part of existing Linked USDL vocabularies. Through this connection all classes existing in Linked USDL can be used for the description of transport and logistics contracts. These include, for example, the description of pricing schemas or the description of parties to the contract. Almost all transport and logistics related extensions are attached to the Contract class so it is the main touch point between Linked USDL (e.g., FI-WARE core vocabularies) and the transport and logistics extensions that are specific domain issues developed in FInest.

Part B – Consignment Information: This class is designed to enable a very detailed description of the goods that could be transported under the contract terms. Nonetheless, the attributes and relationships of this section of the contract are not mandatory and can be used according to the needs of more detailed information or not. Examples of detailed information of the goods are: the measurements (e.g., volume, weight), amounts (e.g., for invoice, customs), indicators (e.g., hazardous).

Part C – Parties involved in the contract: The party class and its associated classes define the information of the parties to the contract. This vocabulary re-uses the vCard vocabulary and the GoodRelations vocabulary.

Part D – Transportation Service Description: We designed the contract vocabulary to be flexible and represent contracts that specify transportation and logistics services in two ways. One is the minimal representation of a service in a transport and logistics contract, in which only one service point is used. This means that the contract specifies the requirements (i.e., Service level Objects - SLOs) for a single transportation leg. The granularity of this leg is irrelevant. For example, the leg could be from Turkey to UK, or from Amsterdam Airport to the Rotterdam port. The second way of specifying the requirements of the transport and logistics service agreed to in the contract is to use multiple service point descriptions, each one with its own requirements. In this case, the contract vocabulary is able to represent contracts that specify different services agreed to by the same parties. For example, the contract could specify that goods from party A should be transported by party B from Turkey to UK. This would require specific requirements for the sea transportation from Turkey, and another set of requirements for the road transportation once the goods arrived in UK.

Part E – Terms of the Contract: For each service point specified in the contract, terms must be associated with them. Using the TSD and TEP data models as a basis, we defined in the contract vocabulary three minimal terms that must be specified: payment, delivery, and execution.

The requirements established in Part D and E of the contract constitute the minimal set of SLOs (Service Level Objects) that are very important for the integration of contract information into the daily operation of transport and logistics services. We are aware that many other types of requirements can be specified in the contract vocabulary, and this can be easily incorporated in to the Linked-USDL vocabulary by associating the new requirements specified into the one defined in WP8. In the same way that we re-used other existing vocabularies, the Linked-USDL Transport and Logistics Vocabulary can also be extended and re-used.

3.4 Linked-USDL Vocabulary for Transport and Logistics Codes

Beside the design of the transport and logistics vocabularies for service offer, demand, and contract as detailed in the sections above, progress has been made in order to create auxiliary vocabularies for typical transport and logistics codes. Thus, in addition to the three main vocabularies, logistic code types [4] have also been implemented as single RDF vocabularies/taxonomies. Table 14 contains all logistic codes that have been implemented for use in the transport and logistics vocabularies.

Table 14 – Implemented Logistics Codes used by Offer Vocabulary

Code Name	Defined in/by
CargoTypeCode	UN/CEFACT 7085 Nature of cargo, coded
LocationTypeCode	UN/CEFACT Location function code qualifier

Code Name	Defined in/by
PackagingTypeCode	UNECE Package Type Code
TransportEquipmentTypeCode	UN/CEFACT 8053 Equipment type code qualifier
TransportMeansTypeCode	UN/CEFACT Trade Facilitation Recommendation No. 28
TransportModeCode	UN/CEFACT Recommendation 19 Code for Modes of Transport
TransportServiceCode	FP7 e-Freight project

In order to evaluate the implemented RDF vocabulary together with the code taxonomies, initial transport and logistics service offer examples have been created in Linked USDL. So far three example files exist that describe different scheduled maritime services.

Three variants of the same service type have been created instead of creating one example of each service type. The reason for this is the close interaction between FI-WARE and WP8 that led to this request by FI-WARE. In the context of the Discovery & Matchmaking component (part of the Marketplace GE) FI-WARE is experimenting with a service offer comparison algorithm; thus, similar transport and logistics service offer examples with only minor differences were needed, and WP8 contributed to the development of these FI-WARE activities.

In September 2012 FI-WARE started testing the Marketplace and the Service Repository GEs with the mentioned example service offers. However, the tests were not completed for the final version of this document; therefore, no results can be presented here.

4 Phase II Implementation Plan

cSpace is the follow up project in Phase II in which the ECM specification and acquired knowledge will be transferred. The detailed description of how the overall cSpace project is organized and will be implemented can be found in deliverable D3.4. In this report we focus on the specific parts of cSpace that are related to the Phase II implementation plan for the ECM module. As illustrated in Figure 15, the ECM module is present in the cSpace project in terms of technical and knowledge transfer.

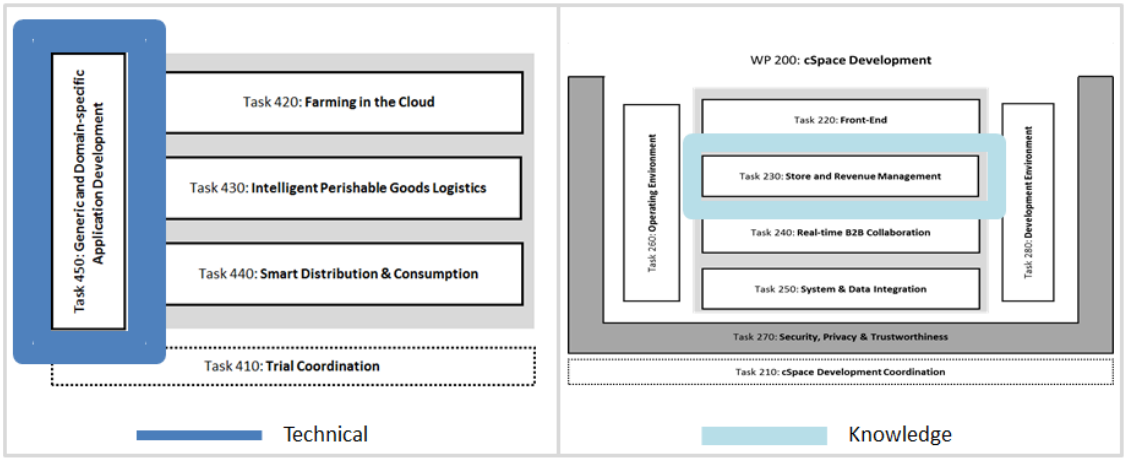


Figure 15 – Mapping of ECM into cSpace Project in Phase II

The concrete development of the ECM technical specification will be implemented under the scope of WP400 ("Use Case Trials") under the task 450 ("Generic & Domain-Specific App Development"). In addition, the consolidated knowledge about defining Linked-USDL vocabularies and how to use the Repository, Marketplace and Store FI-WARE GEs will be essential to support the development of the activities in Sub-task 230 ("Store and Revenue Management") under the scope of WP200 (cSpace Development).

4.1 ECM in Phase II cSpace Project

This section provides the discussion about how the achievements under WP8 will be transferred to cSpace project, as well as, how the ECM will be further implemented in Phase II.

4.1.1 Technical Implementation into Sub-Task 451

One of the main activities in ST451 (Sub-Task 451) is the development of the so called baseline applications (i.e., baseline apps), which are based on the services provided by the cSpace

platform. Four baselines apps were initially identified to be developed in this task. These baselines apps are, in fact, the follow up implementation of domain specific services defined in Phase I. One of these baselines apps is the *Business Services & Contract Management* (BS&CM).

The BS&CM baseline app in cSpace is strongly but not solely based on the ECM module of the Finest project. Thus, the implementation of the ECM will be conducted under the scope of the BS&CM baseline app.

The work in the BS&CM has two main streams of effort: management of business service relationships that have not been established and management of business service relationships established in contracts. The focus of the BS&CM is on the business service and contract management perspectives and takes advantage of the management of software services, part of the core architecture of cSpace. Therefore, the core objectives of the BS&CM Baseline app are:

- Support marketplace operations (e.g., publish offers and demands, search for services) related with business services based on Linked-USDL vocabulary
- Enable the integration with external marketplaces
- Create a prospection mechanism to find business opportunities
- Provide online and run-time information to support contract lifecycle management operations and the execution of business services according to the terms agreed in established contracts.
- Feedback management for service recommendation

4.1.2 Knowledge Support in Task 230

The Task 230 in WP200 will implement the software infrastructure for consuming, re-using, purchasing cSpace Apps (like the ones developed in ST451). Technologies developed by the FI-WARE project are considered to be used in order to build such infrastructure. Candidates for this are: Linked-USDL and the GEs in the Application Chapter, such as Store, Repository, and Marketplace.

The knowledge aggregated in WP8 during the specification and development of the proof-of-concept can be transferred to the context of Task 230. The specifications defined for the ECM cannot be directly mapped or implemented in Task 230, but the experience of how to extend Linked-USDL vocabularies and how the GEs work can support the technical specification of components to be conducted in this task.

It is important to remark that there is not a direct mapping nor an implementation plan for transferring the knowledge because this will happen based on the interaction (and continuation) of people that worked in the ECM module in Phase I and the ones that will work in cSpace during Phase II.

4.2 List of Relevant Milestones

The milestones relevant for the ECM module implementation in Phase II are listed in Table 15, and are all related with the activities that will be conducted under the scope of Task 450 in the cSpace project.

Table 15 – Relevant Milestones in cSpace Project associated with Task 450

Nr.	Milestone Name	Date Month	Associated Deliverable	Description
1	Consolidation	3	D400.6	The first milestone is associated with the consolidation of the technical specification of the baseline apps. The technical specification from Phase I will be adapted to the requirements and context of the cSpace platform.
3	Release V1	9	D400.7	These three milestones are associated with the baseline app development. The software development in cSpace project, and thus the baseline apps, is planned to follow an incremental and iterative cycle composed of three releases. Each release is, in fact, associated with one milestone.
5	Release V2	15	D400.8	
7	Release V3	21	D400.9	

4.3 List of Related Planned Deliverables

The list of deliverables relevant for the ECM implementation in Phase II are listed in Table 16 and are all related with the activities that will be conducted under the scope of task 450 in the cSpace project.

Table 16 – List of Relevant Deliverables in cSpace Project associated with Task 450

Nr.	Deliverable Name	Type	Delivery Month	Description
D400.6	Functionalities of baseline applications	Report	3	Full definition and explanation of the functionalities of the baseline apps, documented in the project Wiki.
D400.7	Baseline applications 1st release	Prototype	9	First release of the implemented four different baseline applications, including the release related documentation of the applications in the project Wiki.
D400.8	Baseline applications 2nd release	Prototype	15	Second release of the implemented four different baseline applications, including the release related documentation of the applications in the project Wiki.
D400.9	Baseline applications 3rd release	Prototype	21	Third release of the implemented four different baseline applications, including the release related documentation of the applications in the project Wiki

4.4 Gant Chart

The timeframe for the development of the cSpace project is 2 years. Within this period the activities for the implementation of the ECM in Phase II comprise the interval between M1 and M21 of the project. Figure 16 depicts the Gant chart with the task in cSpace project for the implementation of the ECM in Phase II and its associated milestones.

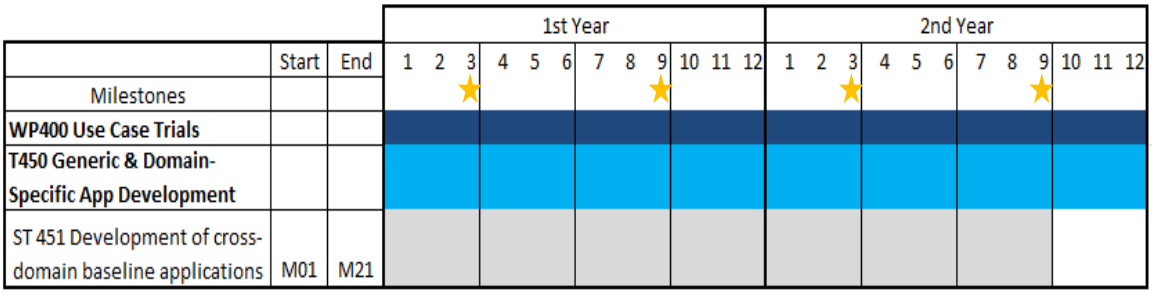


Figure 16 –Gant Chart of ECM Implementation Plan in Phase II

5 Conclusions

This document describes the final technical specification of the ECM module (including, components, interfaces, data types and data models developed within the report period) and the ECM implementation plan for Phase II. The ECM specification will be used as the baseline for the follow up project in Phase II, called cSpace, in order to develop one of the baseline applications of the project.

In addition, to the work conducted in this work-package we established a very close channel for discussion with the Application Chapter of FI-WARE, in which the most relevant GEs associated with the ECM module are designed and developed. The iterative process established between WP8 and the App Chapter from FI-WARE helped to better understand and develop the ECM based on the interfaces of the GEs. As described in this document, this knowledge and the communication channel with App Chapter of FI-WARE will be further exploited in Phase II.

Based on the extensive and detailed work done during this period of the report, we have been able to achieve the objectives of deliverable D8.5, which are: (i) provide the final technical specification of the ECM (task T8.2); (ii) provide domain-specific capabilities of the ECM module using GE (Generic Enablers) offered by FI-WARE (task T8.3); (iii) technical design and implementation of the ECM module aligned with concrete technical realization of GEs (task T8.4); (iv) technical implementation of the proof-of-concept of the ECM module (task T8.4); (v) and detailed implementation plan for the follow up of the ECM module in Phase II (task T8.5).

References

[1] eFreight Project . Accessed March 2012. Available at: http://www.efreightproject.eu/default.aspx?articleID=18749&heading=The%20Project

[2] Linked-USDL. Accessed March 2012. Available at: http://linked-usdl.org/

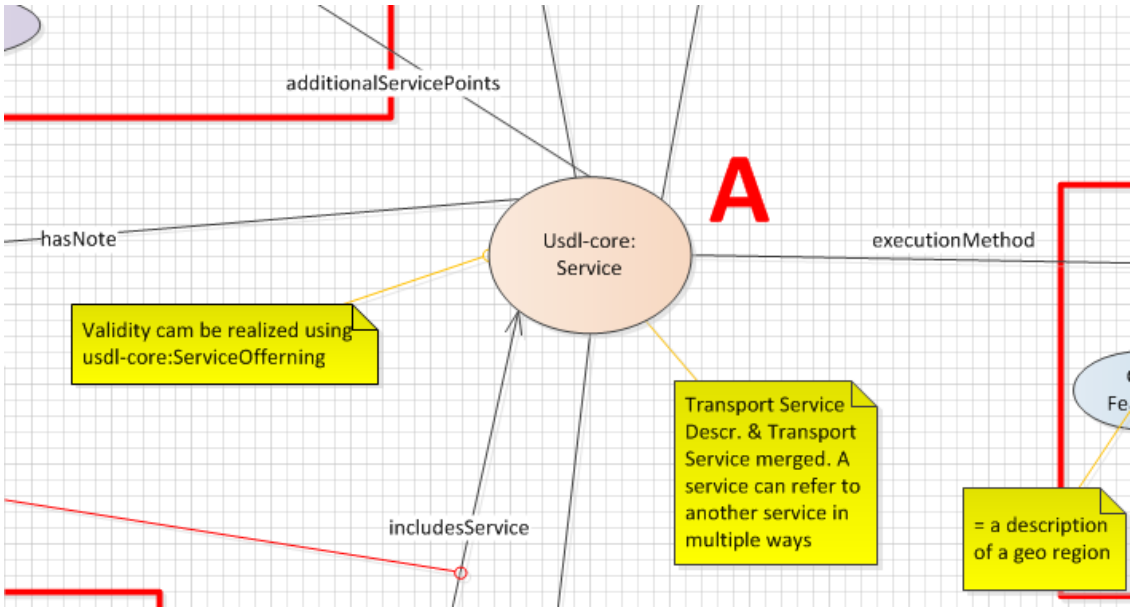
[3] A. Vennesland, H. Westerheim, M. Rosén, C. Kjellberg, V. Mary, P. Wiman og J. T.

Pedersen: e-Freight D1.3b – e-Freight Framework - Information Models, 2010

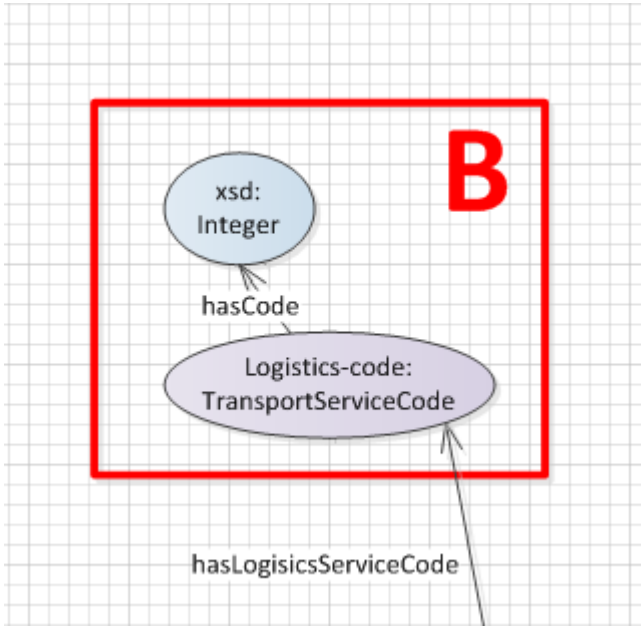
[4] A. Vennesland, H. Westerheim, T. Cane, O. Klein, N. Bento, M. Rosén, C. Kjellberg, P. Wiman and J. T. Pedersen: e-Freight Framework – Annex 3 Data Types and Code Lists, 2012

Appendix A – Parts of Service Offer Vocabulary

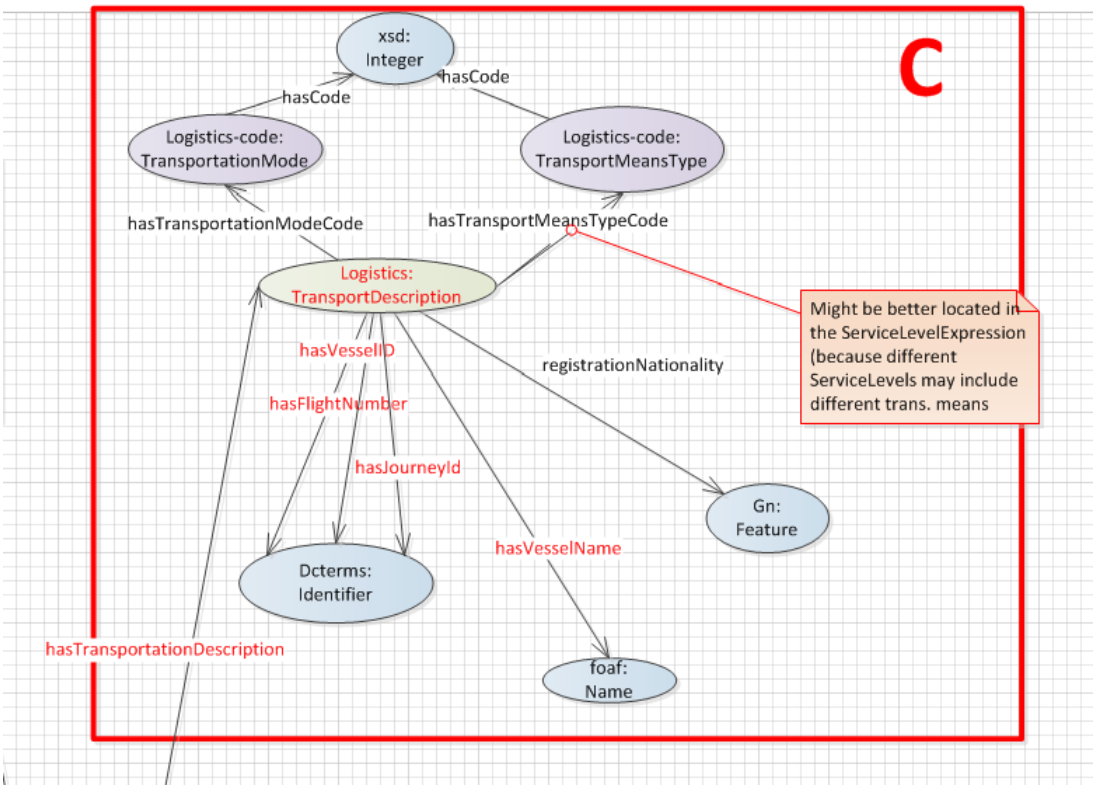
Service Offer Vocabulary – Part A



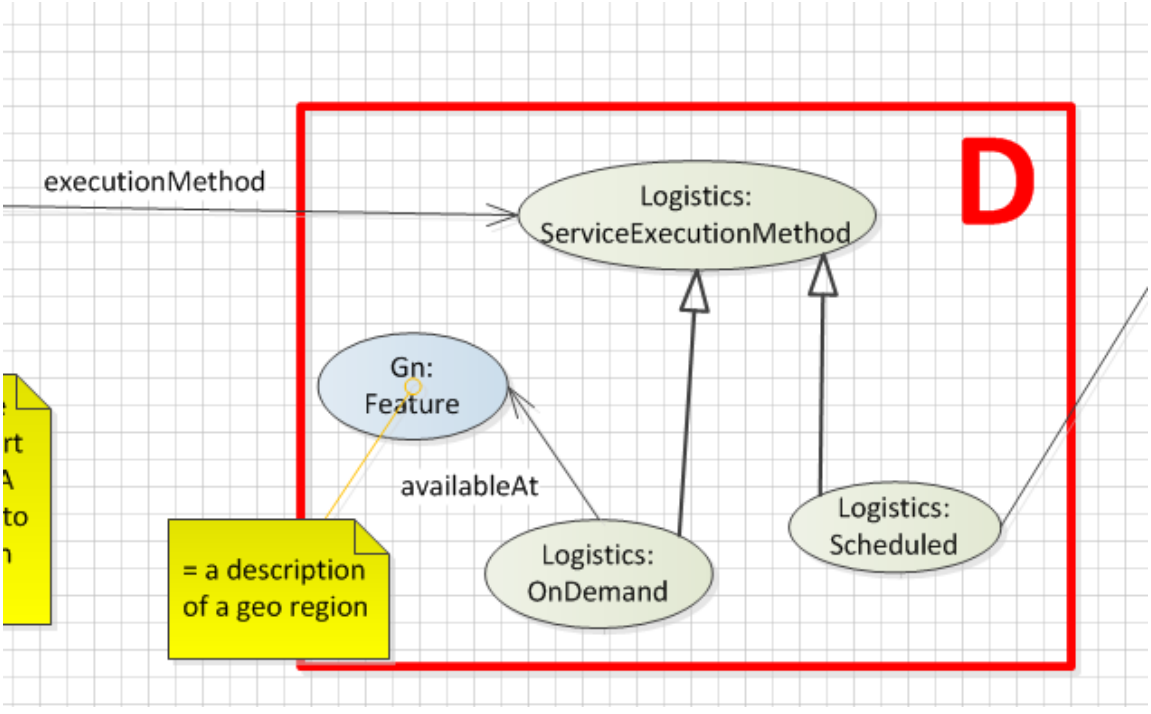
Service Offer Vocabulary – Part B



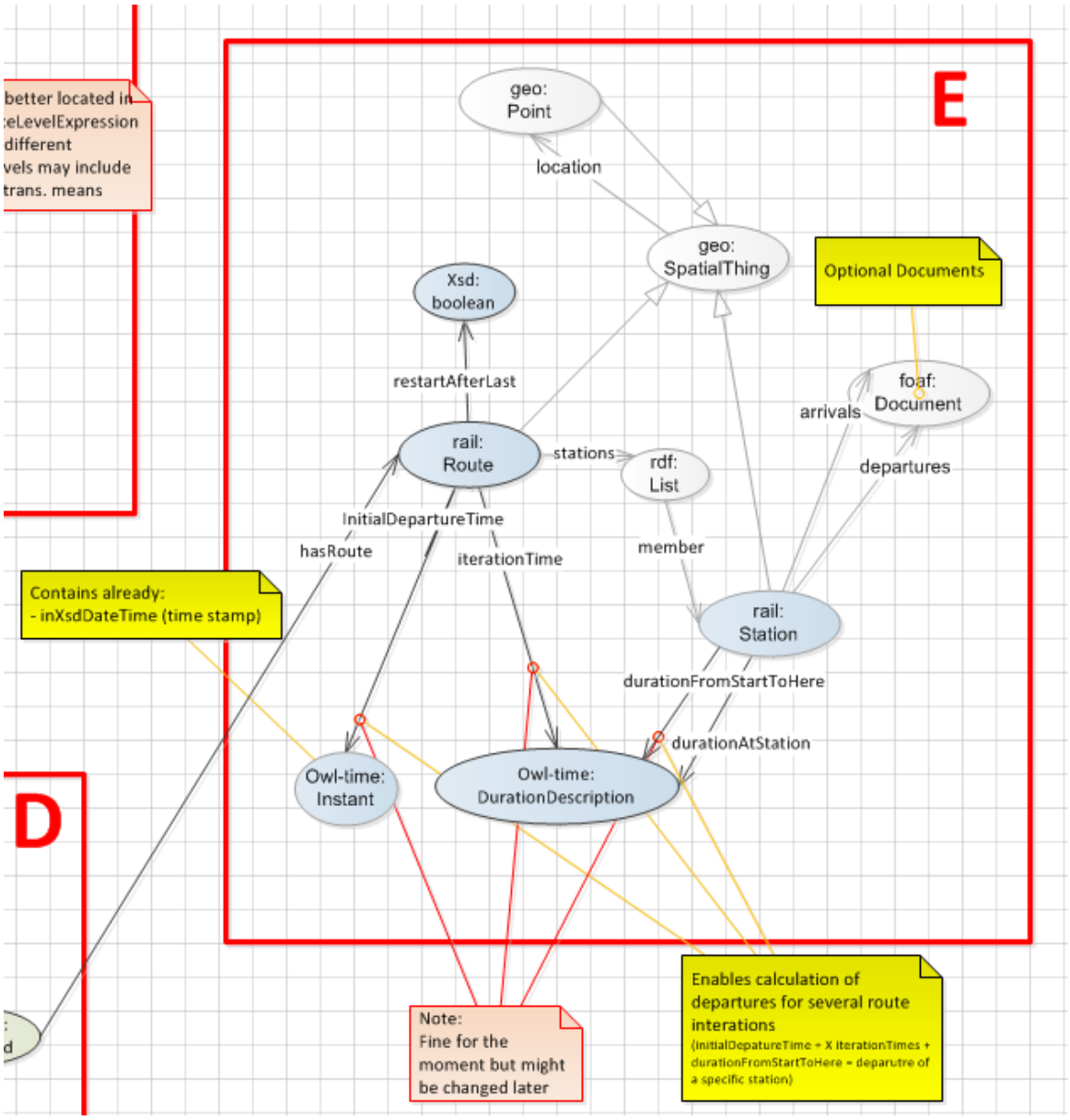
Service Offer Vocabulary – Part C



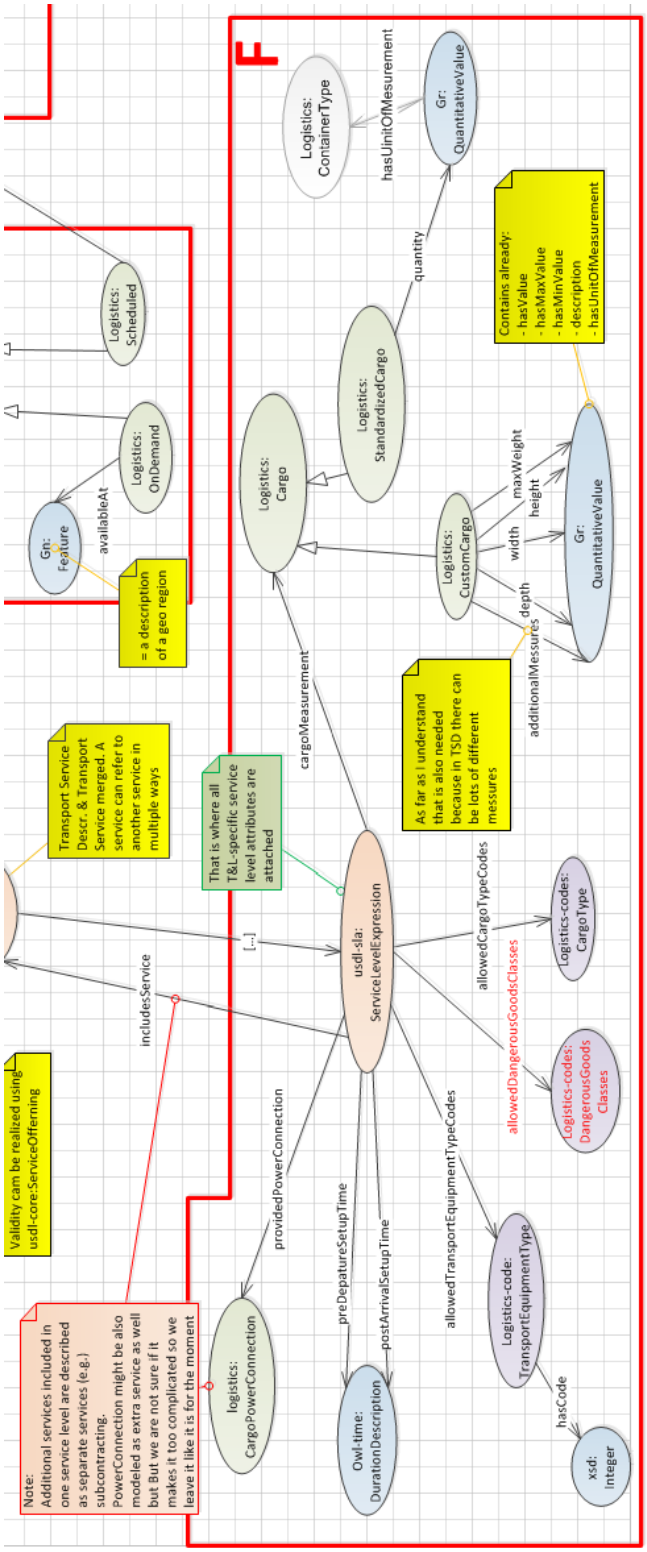
Service Offer Vocabulary – Part D



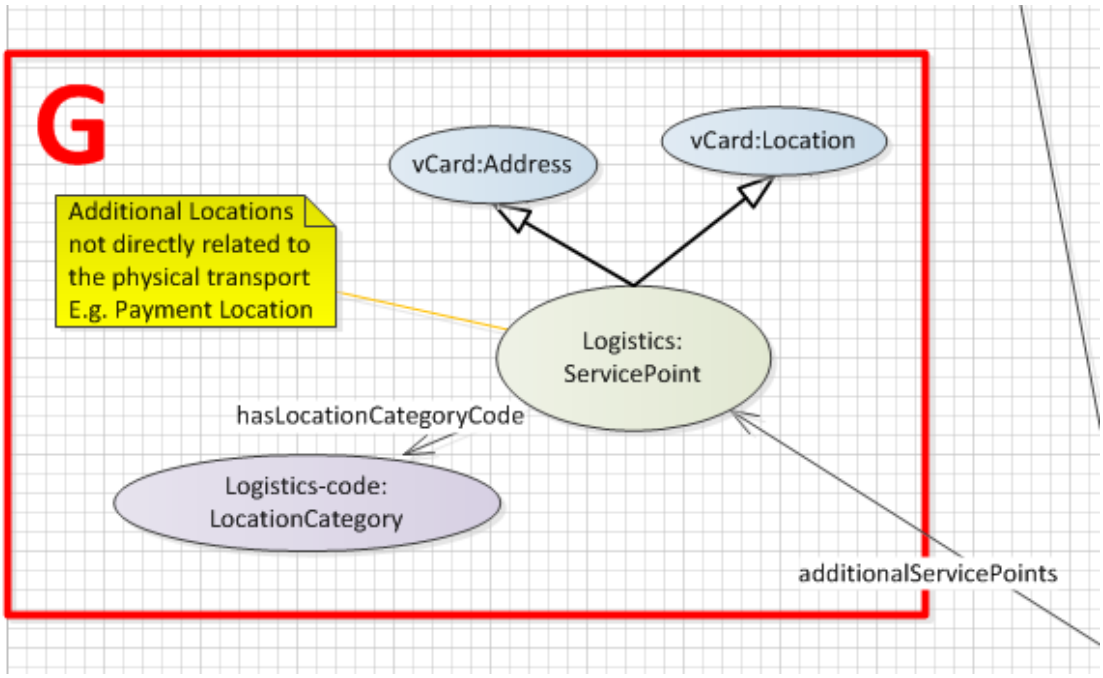
Service Offer Vocabulary – Part E



Service Offer Vocabulary – Part F

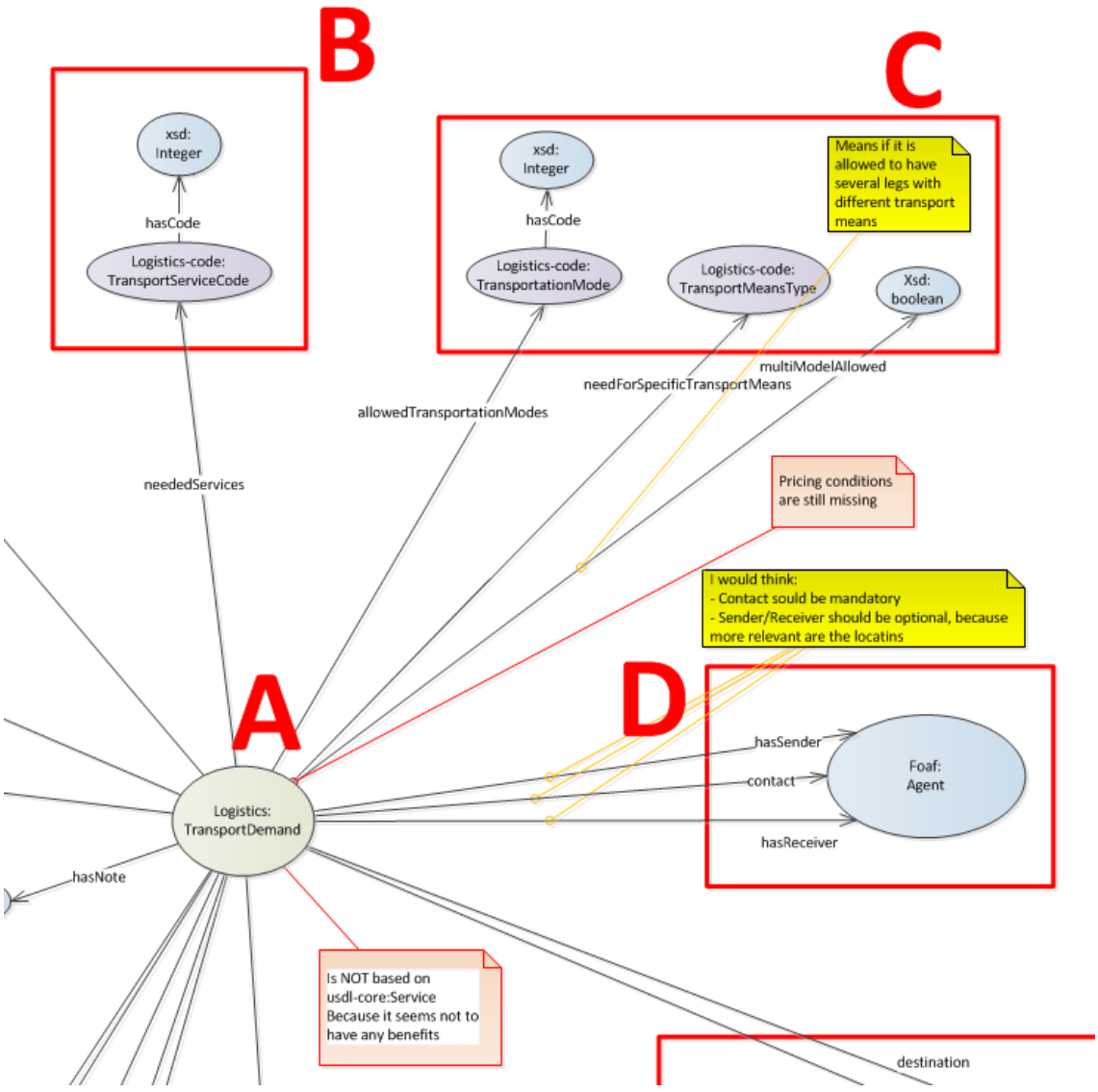


Service Offer Vocabulary – Part G

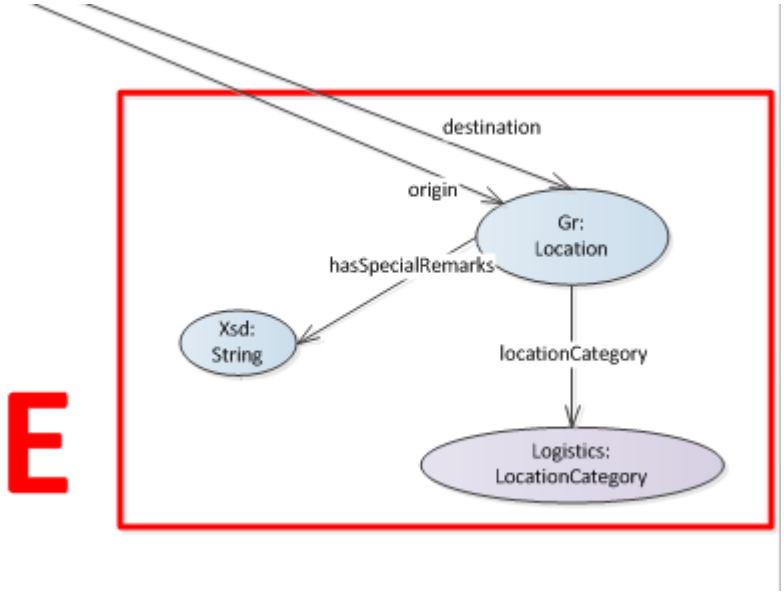


Appendix B – Parts of Service Demand Vocabulary

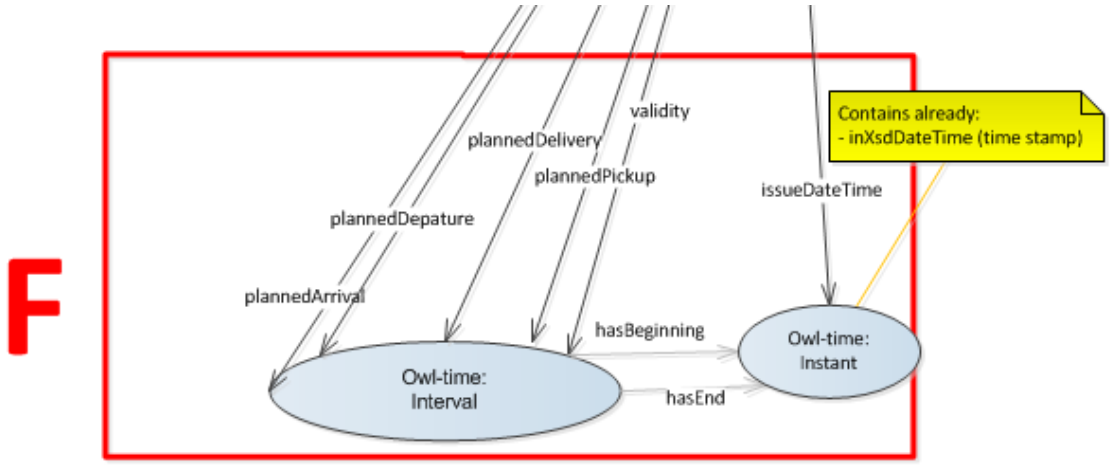
Service Demand Vocabulary – Parts A -D



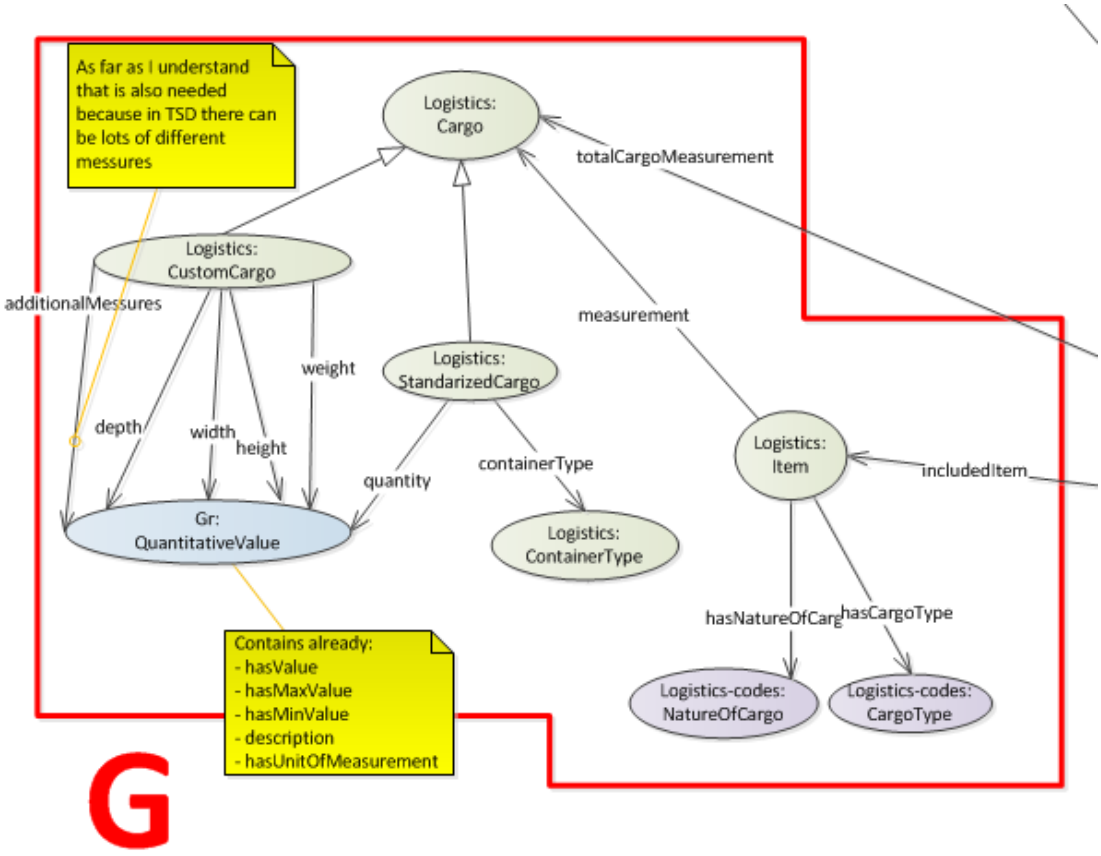
Service Demand Vocabulary – Part E



Service Demand Vocabulary – Part F

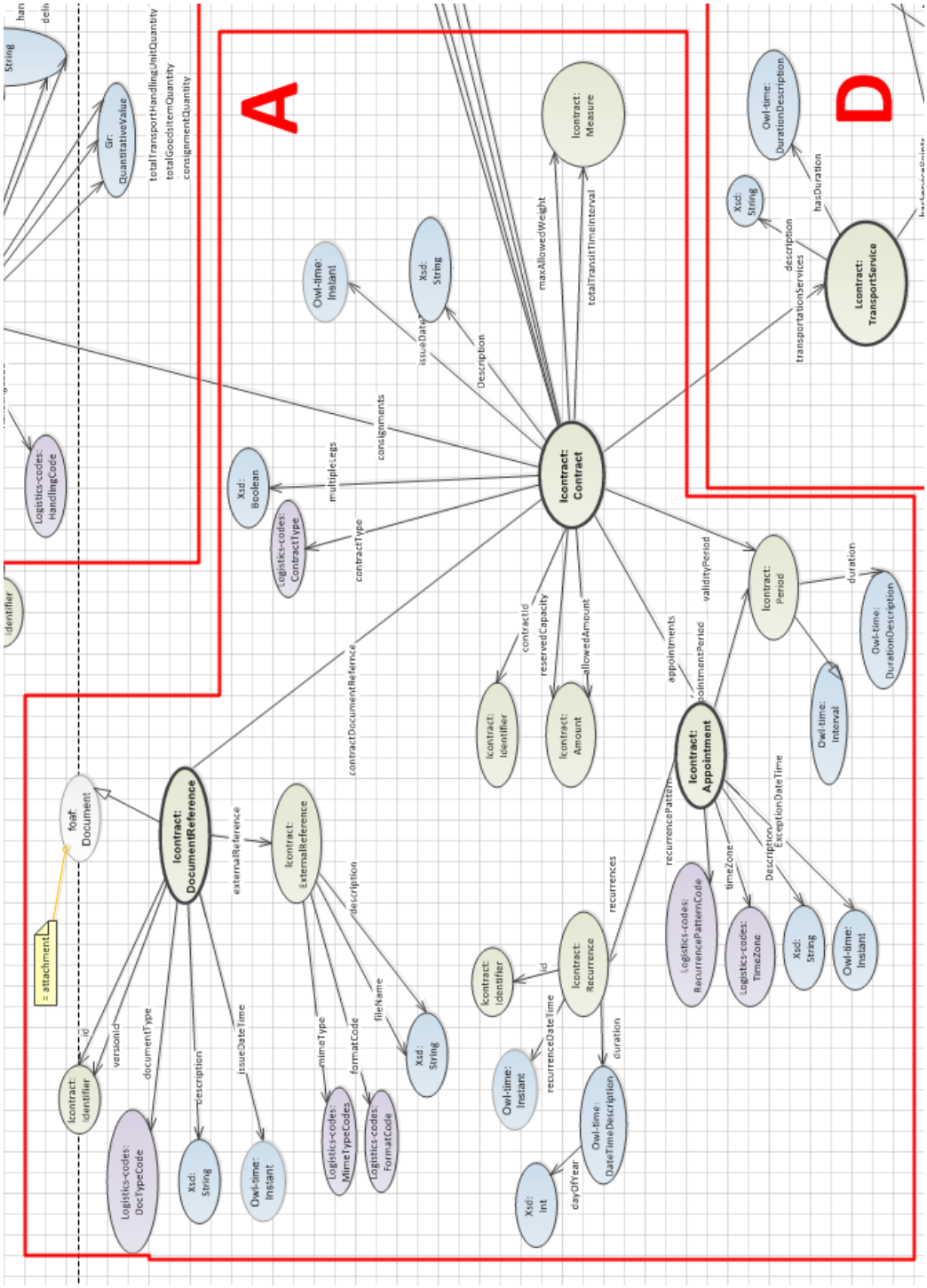


Service Demand Vocabulary – Part G

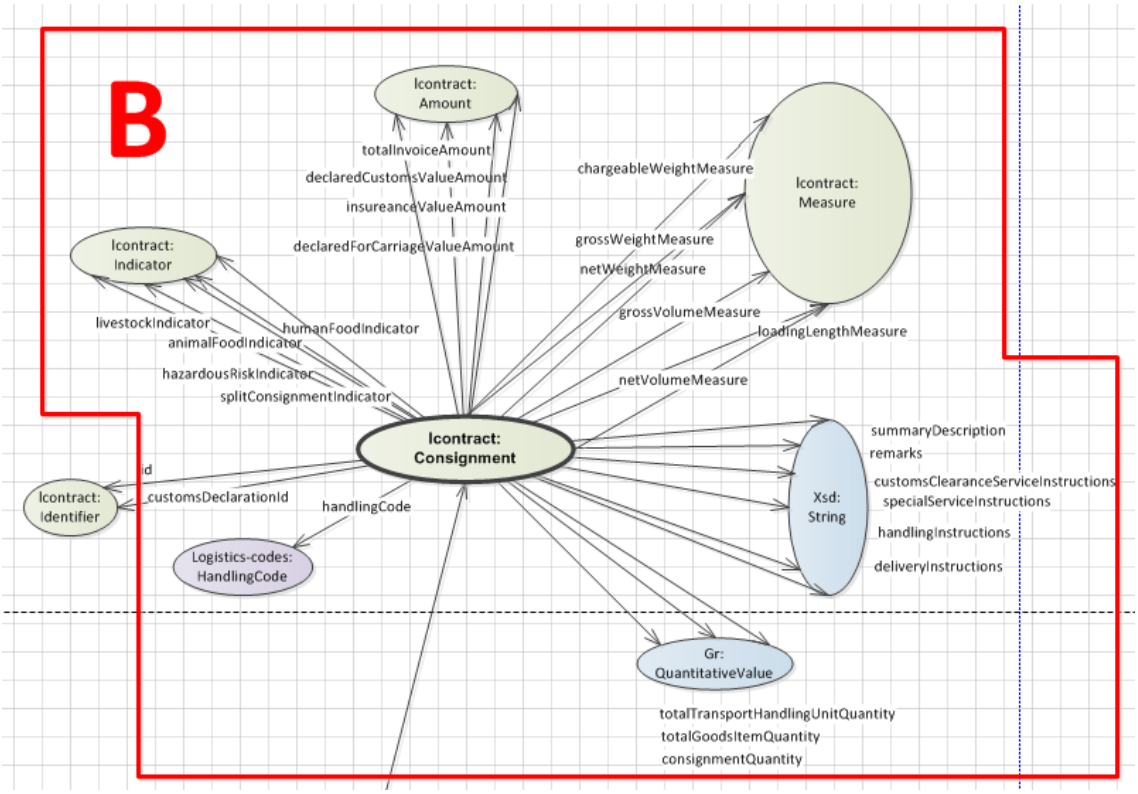


Appendix C – Parts of Contract Vocabulary

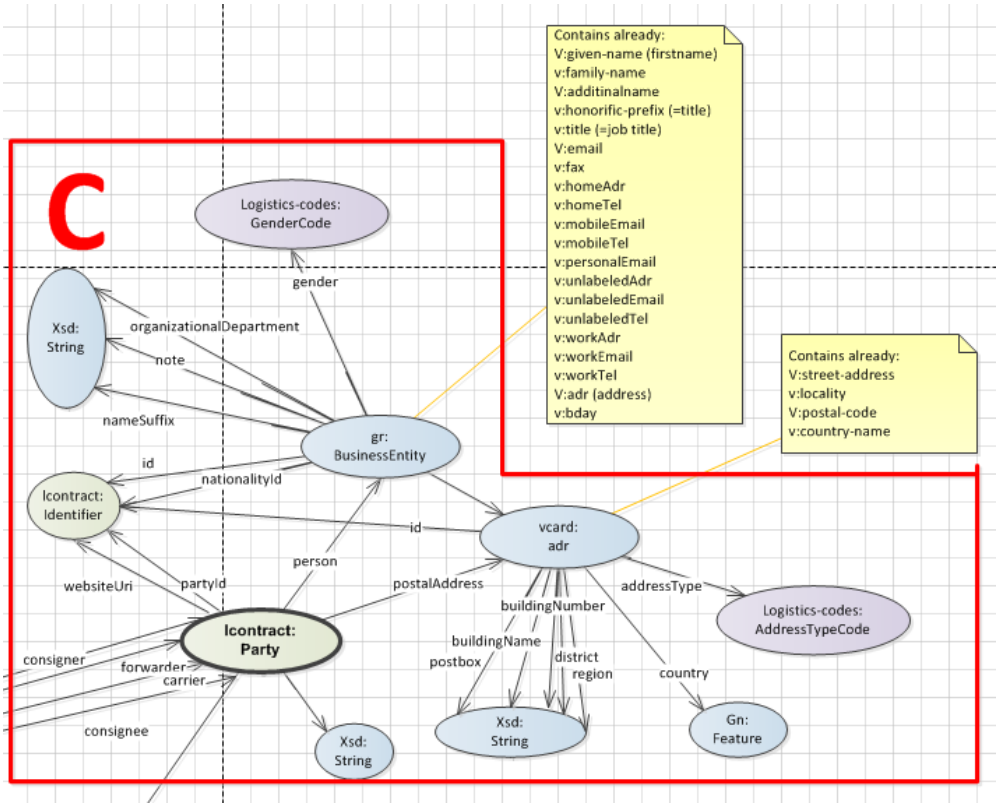
Basic Contract Information – Part A



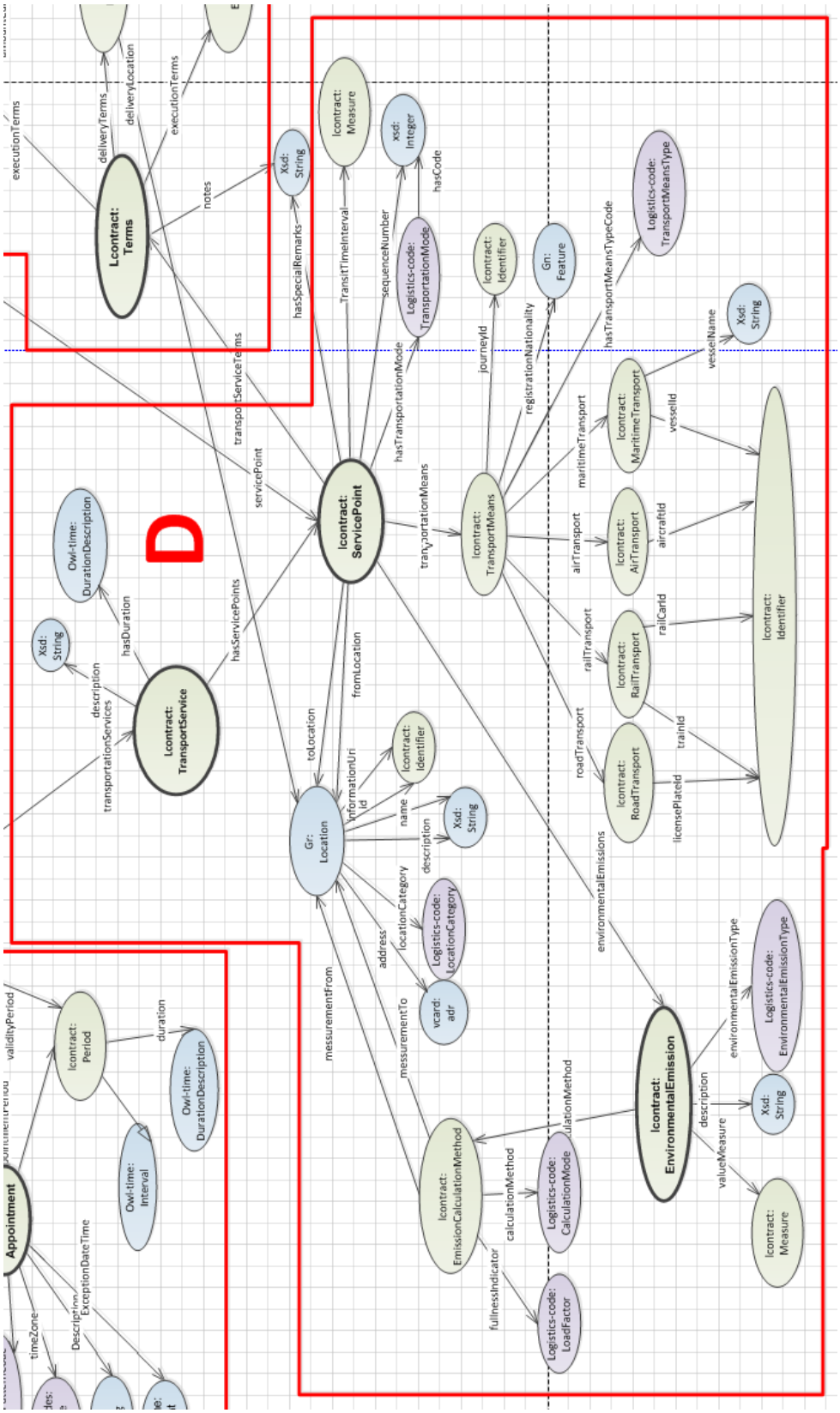
Consignment – Part B



Parties of the Contract – Part C



Transportation Service Description – Part D



Terms of Transportation Services – Part E