INFORMATION TECHNOLOGIES FOR SHIFT TO RAIL

D1.8 – Proof-of-Concept Packaged Resolvers Final Release Features

Due date of deliverable: 28/02/2017

Actual submission date: 22/06/2018

Leader/Responsible of this Deliverable: LEONARDO

Reviewed: Y

Document status		
Revision	Date	Description
0.1	27/07/2017	First draft of the deliverable
0.2	22/02/2017	Added a paragraph describing the operational environment in which the Interoperability Framework is deployed.
1	24/05/2018	Release for FREL milestone of the specifications document and a Proof-of-Concept implementation of the full features of Location, Travel Expert and Event resolvers,
2	04/06/2018	Draft version after WP7 review
3	13/06/2018	New version of the document after WP7 review
4	22/06/2018	Final version after TMC approval and Quality check

Project funded from the European Union's Horizon 2020 research and innovation program		
Dissemination Level		
PU	Public	X
CO	Confidential, restricted under conditions set out in Model Grant Agreement	
CI	Classified, information as referred to in Commission Decision 2001/844/EC	

Start date of project: 01/05/2015 Duration: 36 months

INTRODUCTION

This document represents an update concerning the design and implementation of the Interoperability Framework, envisaged for the Final Release (FREL) of IT2Rail. The Final Release builds on elements developed during the Core and Additional Releases of the Interoperability Framework documented in its respective project Deliverables D1.6 and D1.7. Chapters 1 through 8 of this document describe the purpose, design drivers, use cases, provided capabilities, logical function sequences, components and interfaces of the IT2Rail Interoperability Framework.
Chapter 9 documents the implementation of the additional release (FREL) features of the design, namely:

1. The **it2rail rdf-framework** foundation framework for processing data expressed in the Resource Descriptor Framework (RDF) language, semantically annotated with terms described in the domain's ontology, the latter written in the Ontology Web Language (OWL). The framework provides additionally connectivity to distributed triple stores, including the IT2Rail triple store that implements the Ontology Repository, containing the OWL ontology, and the Semantic Web Service Registry containing service descriptors also represented as RDF statements.
2. The **Semantic Graph Manager** component, based on the it2rail RDF framework, which provides semantic data discovery, query an aggregation over linked, distributed triple stores.
3. The **Location Resolver** service, which uses the Semantic Graph Manager to provide "packaged" data discovery, query and aggregation about transport "infrastructures" such as Airports, Bus Stops, etc.
4. The **Travel Expert Resolver** service, which uses the Semantic Graph Manager to provide "packaged" service discovery from the Semantic Web Service Registry that can generate Offer Items for a given Route during the Shopping process.
5. The **Locations Identification** service, which provides the Travel Companion application with geographic coordinates of specified Points of Interest.
6. The **Resolve Events Source** service, which supports the Trip Tracking application by providing of Stop Places and Transportation Services for use with the Navitia platform[1] to identify disruptions to booked itineraries.
7. The **Travel Expert and Booking Engine Brokers**, which mediate the interactions of the Shopping and Booking & Ticketing applications with external services at Transport Service Providers.
8. The **NeTEx Producer** service, which implements a subset of the NeTEx 1.03 producer web service specification to return Stop Places in a bounding box in the NeTEx 1.03 format.
9. The **Network Statistics Provider** service, which generates lists of "meta travel episodes" associated with a Travel Expert for use by the Shopping application.

The content of related Work Packages with the IT2Rail project is not detailed in this document: readers are referred to the documents listed in the Referenced Documents chapter of this document.

[1] The Navitia platform (https://www.navitia.io/) provides application programming interfaces for access to Public Transport data sets and application services covering approximately 15.000 cities and 1.600 networks in 25 countries who participate in an "open data" approach to the implementation of Public Transport applications

All terms and acronyms are defined in the IT2Rails glossary, part of D7.10 Development Readiness Review Pack (FREL).

TABLE OF CONTENTS

Introduction 2
List of Figures 7
List of Tables 9
1. Referenced Documents 10
2. Principles and Purpose 11
2.1 Semantic Interoperability 11
2.2 Design Drivers 12
3. Operational Scenario 14
3.1 Actors and Context 14
3.1.1 Actors 14
3.1.2 Context 15
3.2 Use cases 16
3.2.1 IF in Manage Mobility and travel rights delivery Use Case 16
3.2.2 IF in Track Itinerary Use Case 17
3.2.3 Manage IF Assets and Publish Resources 17
Manage IF Assets 18
Publish Resources 18
4. Capabilities 19
4.1 Manage IF Assets 19
4.1.1 Manage Semantic Graph 19
4.1.2 Maintain Ontology 19
4.1.3 Maintain Service registry 19
4.1.4 Maintain Data Semantic Graph 20
4.2 IF Resolver Services 21
4.2.1 Resolve Locations 21
4.2.2 Resolve Travel Expert 21
4.2.3 Resolve Events Source 21
4.2.4 Provide Network Statistics Data 21
4.3 Mediate Travel Expert Invocation 22
4.3.1 Import Ontology and Mappings 23
5. Activity Diagrams 24
5.1 Manage IF Assets 24
5.1.1 Functions in Maintain Semantic Graph realisation 24

5.1.2 Functions in Maintain Ontology Realisation 24
5.1.3 Functions in Maintain Service Registry realisation 25
5.1.4 Functions in Maintain Data Semantic Graph realisation 25
5.2 IF Resolver Services 26
5.2.1 Location Identification 26
Functions in Location Identification realisation 26
5.2.2 Provide Network Statistics Data 26
Functions in Provide Network Data capability realisation 27
5.2.3 Resolve Locations 27
Functions in Resolve Locations capability realisation 28
5.2.4 Resolve Travel Expert 28
Functions in Resolve Travel Expert capability realisation 29
5.2.5 Resolve Events Source 29
Functions in Resolve Events Source capability realisation 30
5.3 IF Semantic Broker 30
Functions in Mediate Travel Expert Invocation capability realisation 31
Functions in Manage Broker Ontology capability realisation 31
6. Components and Component Exchanges 32
6.1 IF Asset Manager Component Exchanges 32
6.2 IF Resolver Services Component Exchanges 32
6.3 Semantic Broker Component Exchanges 33
7. Interfaces 34
7.1 IF Assets Manager 34
7.1.1 External Interfaces 35
Asset Management GUI 35
Ontology Upload 36
Annotations GUI 37
Import External Resources 38
Triples Store Interface 39
7.1.2 Internal Interfaces 40
Approve Graph 40
Store Semantic Graph 41
Store Mappings and Schemas 42
7.2 IF Resolver Services 43

7.2.1 External Interfaces 44
IdentifyLocations 44
NetworkStatistics 45
Locations Resolver 46
Travel Expert Resolver 47
Navitia Decoder 48
7.2.2 Internal Interface 49
Semantic Graph Manager 49
7.3 IF Semantic Broker 50
7.3.1 External Interfaces 52
Travel Expert Broker 52
Booking Engine Broker 53
Repository Download Services 54
7.3.2 Internal Interfaces 55
Travel Expert JSON Ontology 55
8. Technical Architecture 56
8.1 Deployment Scenarios 56
8.2 Native IT2Rail 57
8.3 Wrapped legacy service 57
8.4 Broker – Enterprise Service Bus 58
8.5 Operational Environment 59
9. Technical Demonstrator FREL Implementation 60
9.1 Technical demonstrator packaging for delivery 60
9.2 The IT2RAIL RDF framework 62
9.2.1 Empire Configuration 63
Default Empire Configuration 63
9.2.2 Entity Manager 64
9.2.3 RDF Generator 66
9.3 The Semantic Graph Manager 66
9.4 Location Resolver Service 68
9.5 Travel Expert Resolver Service 70
9.6 Semantic Broker and proxy injection 71
9.7 NeTEX Producer Service 74
10. Technical demonstRation Test Environment and Procedure 76

10.1 Technical Demonstration Installation 76
10.2 External Travel Expert and Booking Engine services 79
10.3 Technical Demonstration Test 80

LIST OF FIGURES

Figure 1: [LAB] IF Resolver Services – Shopping Context 15
Figure 2: [LAB] IF Semantic Broker – Shopping Context 15
Figure 3: Resolver services in relationship with Trip Tracker 15
Figure 4: [MCB] Manage Mobility request and travel rights (shopping) 16
Figure 5: : [MCB] Manage Mobility request and travel rights (booking) 16
Figure 6: [MCB] Track Itinerary 17
Figure 7: [MCB] IF Assets Management Capabilities 18
Figure 8: [CRB] Maintain Ontology 19
Figure 9: [CRB] Maintain Service Registry 20
Figure 10: [CRB] Maintain Data Semantic Graph 20
Figure 11: [CBR] IF Resolver Services 21
Figure 12: [CBR] Provide Network Statistics Data capability 22
Figure 13: [CRB] IF Mediate Travel Expert Invocation (shopping) 22
Figure 14: [CRB]IF Mediate Travel Expert Invocation (booking) 23
Figure 15: Manage IF Assets activity diagram 24
Figure 16 [LAB] IF Location Identification 26
Figure 17: Provide Network Statistics Data activity diagram 26
Figure 18: Resolve Locations activity diagram 27
Figure 19: Resolve Travel Expert activity diagram 28
Figure 20: Resolve Events Source activity diagram 29
Figure 21: IF Semantic Broker activity diagram 30
Figure 22: Exchanges across components of the IF Asset Manager 32
Figure 23: Exchanges across components of the IF Resolver Services 33
Figure 24: Exchanges across components of the IF Semantic Broker 33
Figure 25: IF Assets Manager interfaces 34
Figure 26: IF Resolver Services interfaces 43
Figure 27: IF Travel Expert Semantic Broker interfaces 50
Figure 28: IF Booking Engine Semantic Broker interfaces 51

Figure 29: Decoding of symbols used in Fig 30, Fig 31 and Fig 32 56
Figure 30: Native IT2Rail exchange scenario 57
Figure 31: Wrapped legacy service exchange scenario 58
Figure 32: Enterprise Service Bus / Broker scenario 58
Figure 33: Interoperability Framework packaging 61
Figure 34: The Empire Configuration 63
Figure 35: Entity Manager 65
Figure 36: RDF Generator 66
Figure 37 Semantic Graph Manager 67
Figure 38: instantiated resolver 68
Figure 39 Location Resolver service 70
Figure 40: Travel Expert Resolver 71
Figure 41: Asset Manager - Semantic Broker Proxies 72
Figure 42 - AMS Booking Engine proxy governance associations 73
Figure 43: Example "mediation" JAR file contents 74
Figure 44: NeTEX Producer service packaging 74
Figure 45: Sample NeTEX Producer request 75
Figure 46: Fragment of NeTEX Producer service response 76
Figure 47 - Apache Tomcat installation 77
Figure 48 - Apache WSO2 Carbon installation 77
Figure 49 - Ontotext GraphDB Semantic Graph database 78
Figure 50 - Interoperability Framework deployed services 79
Figure 51 - SoapUI configuration for testing campaign 81
Figure 52 - Travel Expert Broker automated test 82

LIST OF TABLES

Table 1: Referenced documents 10
Table 2: Asset Management GUI interface 35
Table 3: Ontology Upload interface 36
Table 4: Annotations GUI interface 37
Table 5: Import External Resources interface 38
Table 6: Triples Store Interface 39
Table 7: Approve Graph interface 40
Table 8: Store Semantic Graph interface 41
Table 9: Store Mappings and Schemas interface 42
Table 10: IdentifyLocations interface 44
Table 11: NetworkStatistics interface 45
Table 12: Locations Resolver interface 46
Table 13: Travel Expert Resolver interface 47
Table 14: Navitia Decoder interface 48
Table 15: Semantic Graph Manager interface 49
Table 16: Travel Expert Broker interface 52
Table 17: Booking Engine Broker interface 53
Table 18: Repository Download Services interface 54
Table 19: Travel Expert JSON Ontology interface 55

1. REFERENCED DOCUMENTS

This section lists the document reference number, title, revision, and date of all documents referenced in the specifications document.

Table 1: Referenced documents

Reference Number	Title	Revision	Date
	IT2RAIL-Proposal_second stage_SECTION 1-3_28082014_.pdf	1	Oct. 09, 2014
ITR-WP1-D-FIN-026-02	D1.6 Proof of Concept Packaged Resolvers Core Features	2	Dec. 13, 2016
ITR-WP1-D-FIN-028-04	D1.7 Proof of Concept Packaged Resolvers Additional Features	4	May 14, 2018
ITR-WP7-D-THP-111-01	D7.10 Development Readiness Review Pack (FREL).		May 5th 2018

2. PRINCIPLES AND PURPOSE

In order to make travel across Europe attractive the rail system must be perceived and used by Customers in coordination with other transport modes as a natural extension of the increasingly digital environment in which they live, work and operate, i.e. an environment in which they live a fulfilling experience.
Such a digital environment is constituted of a multitude of networked, distributed heterogeneous devices systems and applications that cannot be assumed to be under the control of any one specific organization, and that join and leave the environment dynamically, e.g. according to prevailing market and other conditions.
The purpose of the Interoperability Framework is twofold:

1. To provide the cost-effective technical means that allow participant devices, systems and applications to interoperate in the sense that will be specified below.
2. While significantly reducing and potentially eliminating the need for centrally directed and coordinated adoption of centralized platforms or single standards.

While the first requirement is inherent to the customer experience 'environment' being distributed and heterogeneous and can be solved with multiple architectures, the second applies a critical *business* constraint on the solution that must be met for its *effective* adoption by market operators. This constraint determines the particular design of the Interoperability Framework documented in this paper.
Interoperability refers to the ability of devices or systems to participate in the coordinated performance of tasks and functions in the execution of some business process, in which exchanging data is a simple means, but not the purpose of interoperability itself. In fact, interoperability is predicated on the partners involved in the exchange of the data agreeing on the computational model that is applicable to such data and in processing them accordingly, i.e. according to some shared logical interpretation of what the data mean and what can be *meaningfully* be done with them.

2.1 SEMANTIC INTEROPERABILITY

While data can be harmonised syntactically to some common format, which is the approach of 'data formats standardisation', distributed heterogeneous systems are prevented from interoperating by *semantic* heterogeneity, which can be described as follows:

> "In current database systems most of the data semantics reside in the applications rather than in the DBMS[2]. Moreover, data semantics are often not represented directly in the application code, but rather in the assumptions which the application--or, more correctly, the programmer--makes about the data. This situation is tolerated in local database environments largely because the local applications work with a shared set of assumptions. However, serious problems are likely to occur during a database integration--or federation […]--effort because sets of local assumptions clash and local applications do not have access to the semantics represented in the "foreign" applications. This is the semantic heterogeneity problem. When semantic information that is hidden in applications is made explicit and accessible through the database then the semantic problem becomes a much more tractable syntactic problem […].[3]"

[2] DBMS: DataBase Management System

[3] Ventrone, V., & Heiler, S. Semantic Heterogeneity as a Result of Domain Evolution. *SIGMOD RECORD,* 20, 4: pp. 16.20, 1991

While the quotation refers to database system, it actually describes a situation applicable to any application, namely the fact that some fundamental *assumptions* underpin efforts at making them interoperable, and that these assumptions are held *implicitly,* often *informally,* and can only be controlled where some form of *local* sharing of these assumptions can be established, e.g. within a single organisation such as a single company or even a large association or public authority. Where these underlying assumptions remain out of reach of machines and automation, the fundamental obstacle to interoperability becomes by far the cost of the "local sharing", including the costs of participation in the controlling organisation, the cost of the organisation itself, the cost of evolution, etc.
The design of the Interoperability Framework addresses these issues through the creation of an explicit, formal, shareable, machine-readable and computable description of the computational model associated with data descriptions and exchanges in order to allow a higher degree of automation of distributed processes *across* multiple data formats and spanning unspecified actors. We define this approach *semantic interoperability* and we describe it in more detail in the rest of this paper.

2.2 Design Drivers

The Interoperability design follows broad common design for quality guidelines established across the It2Rail project and managed by the project's Technical Coordination.
However, some specific design drivers apply to the Interoperability Framework, and in particular in addressing the following engineering challenges:

- the creation of a shared domain ontology, i.e. of an explicit, formal, shareable, machine-readable and computable description of the computational model associated with data descriptions and exchanges in order to allow a higher degree of automation of distributed processes across multiple data formats and protocols, spanning unspecified actors.

- the provision of a set of semantic interoperability services that can be deployed in multiple architectures and configurations, and that do not mandate a specific set of communication protocols or frameworks, leaving the choice of deployment strategies to partners that may opt to re-use a shared enterprise service bus, perhaps on a virtual private network protected by specific security and authentication protocols, or decide to engage in pure peer-to-peer exchanges over the public world wide web, or a mixture of these or other options. This is also important to allow operators, including yet unknown companies who are not partners in an "integration project", to choose their own roadmap for adoption of the 'native' semantic language for their exchanges, using or discontinuing the semantic transformation services according to their own timeline.

In particular, the design of the IF is conducted so as to:

- Concentrate efforts on research and innovation topics, particularly on semantic technologies for interoperability. For this reason design is geared towards leveraging to the maximum existing open source platforms, frameworks and tooling for ordinary

functions and capabilities such as triple stores, web servers and web services frameworks, repositories, workflow managers, etc.

- Allow for multiple implementation and deployment options of the logical functions and interfaces. In this respect, the component structure and the allocation of functions to these components in this document should be intended to illustrate the specific implementation chosen for the IT2Rail demonstration scenario.
- Exchange Items, i.e. elements exchanged at the interface, describe concepts and relationships, i.e. the IT2Rail *ontology* as it is provided and consumed at interfaces.
- The design of the Interoperability Framework is model driven, and integrated with other components of the IT2Rail project in the same model stored in the modeling tool “Capella”. Diagrams in this document are extracted from the model.
- The Capella model is the fundamental specification of the design: this document is an illustration of the fundamental elements of the model.

3. OPERATIONAL SCENARIO

3.1 ACTORS AND CONTEXT

3.1.1 Actors

The Interoperability Framework is technical enabler and as such it does not interact directly with Passengers or Customers.

It is however involved in interactions with the following Actors:

- **Travel Shopping** is a logical component in the IT2Rail eco-system involved in the realisation of the “Manage Mobility and travel rights delivery” Use Case. The IF provides Resolve Locations, Resolve Travel Expert and Mediate Travel Expert Invocation capabilities to this Actor in the execution of the Use Case.

- **Trip Tracker** is a logical component in the IT2Rail eco-system involved in the realisation of the “Track Itinerary” Use Case. The IF provides the Resolve Events Source capability to this Actor in the execution of the Use Case.

- **Transport Service Operator** is an Actor that publishes and provides resources such as data and web service descriptors that the IF uses to build the distributed semantic web of ransportation data.

- **Travel Event Provider** is an Actor that exposes services describing travel events, such as delays or cancellations. The IF provides capabilities to discover these services and publish their descriptors.

- **OfferProvider** is an Actor that generates mobility service offers in the execution of a shopping process instance. The IF provides capability to broker the request of these offers from Travel Shopping to Offer Provider.

- **IF Assets Manager** is an Actor responsible for maintaining , e.g. approving and versioning of the semantic resources of the Interoperability Framework in a workflow process.

- **Ontology Engineer** is an Actor responsible for developing and maintaining the IT2Rail ontology.

- **Annotation Engineer** is an Actor responsible for the semantic annotation of web service descriptors and data provided by Transport Service Providers, OfferProviders and Travel Event Providers, and for the generation of schemas and transformation mappings.

3.1.2 Context

The following diagrams show the Interoperability Framework resolver services in relation with Travel Shopping components:

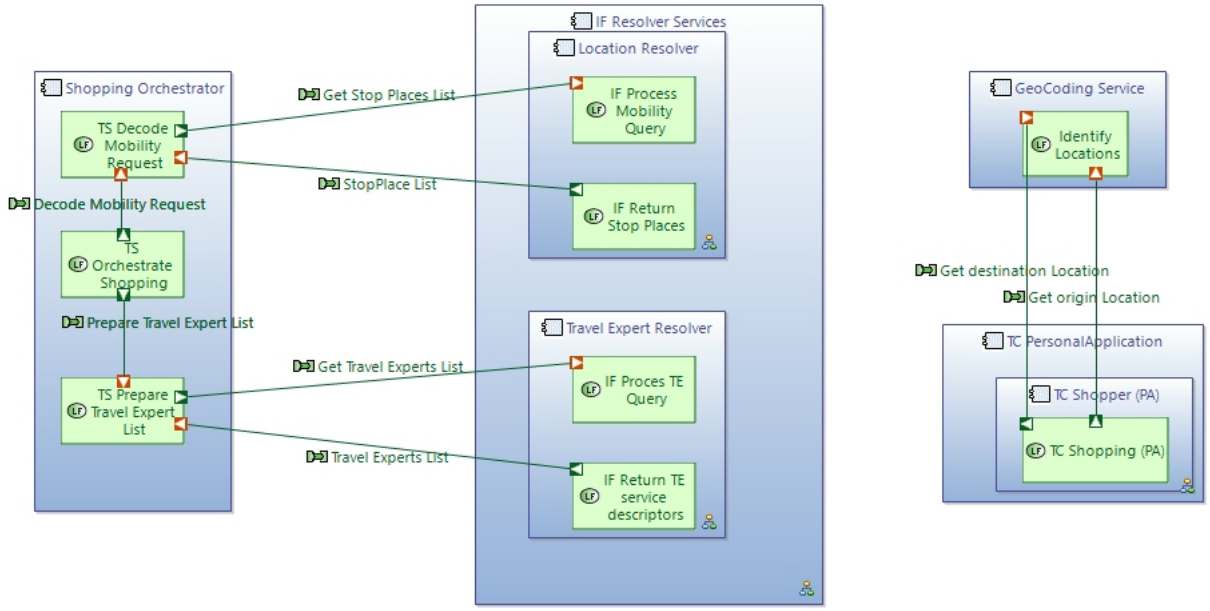


Figure 1: [LAB] IF Resolver Services – Shopping Context

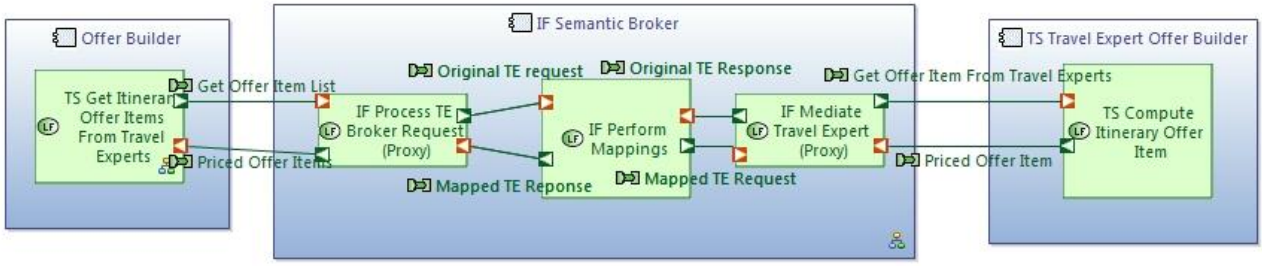


Figure 2: [LAB] IF Semantic Broker – Shopping Context

The following diagram shows the Interoperability Framework resolver services in relation with Trip Tracker components:

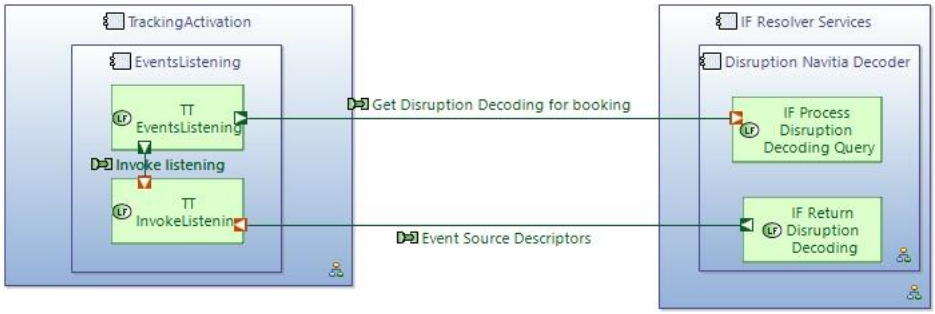


Figure 3: Resolver services in relationship with Trip Tracker

3.2.1 IF in Manage Mobility and travel rights delivery Use Case

The Interoperability Framework provides capabilities included in the Travel Shopping capabilities (Shop - Provide Itinerary Offers and Shop – Provide itinerary offer items) of the "Manage Mobility and travel rights delivery" and in the "Booking and Ticketing" Use Cases.

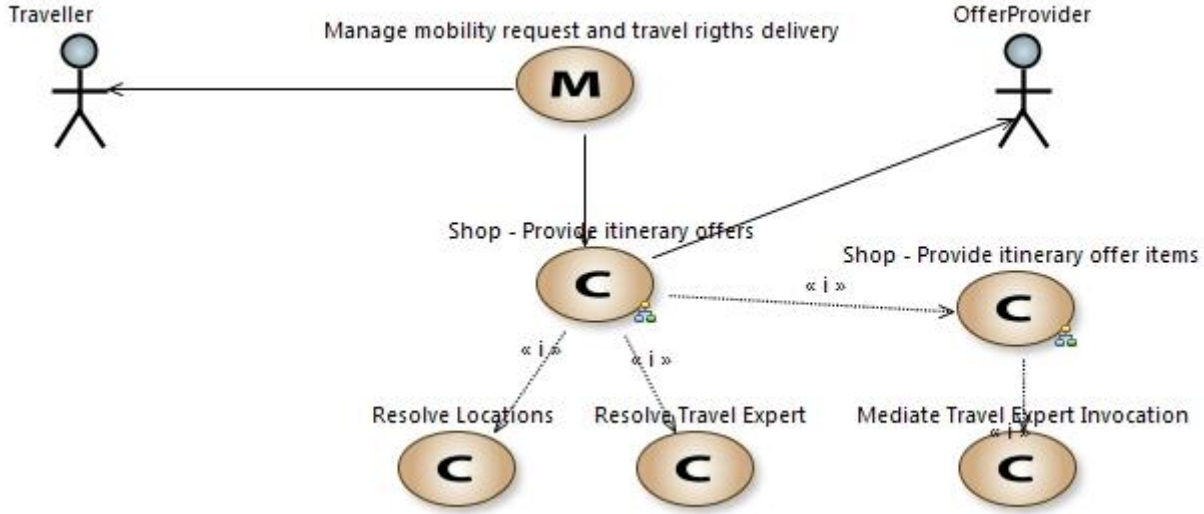


Figure 4: [MCB] Manage Mobility request and travel rights (shopping)

The Interoperability Framework participates in the shopping Use Case by providing the Resolve Locations, Resolve Travel Expert and Mediate Travel Expert Invocation capabilities to the Use Case.

It also participates in the booking Use Case by providing Resolve Travel Expert and Mediate Booking Engine Invocation capabilities to the Use Case.

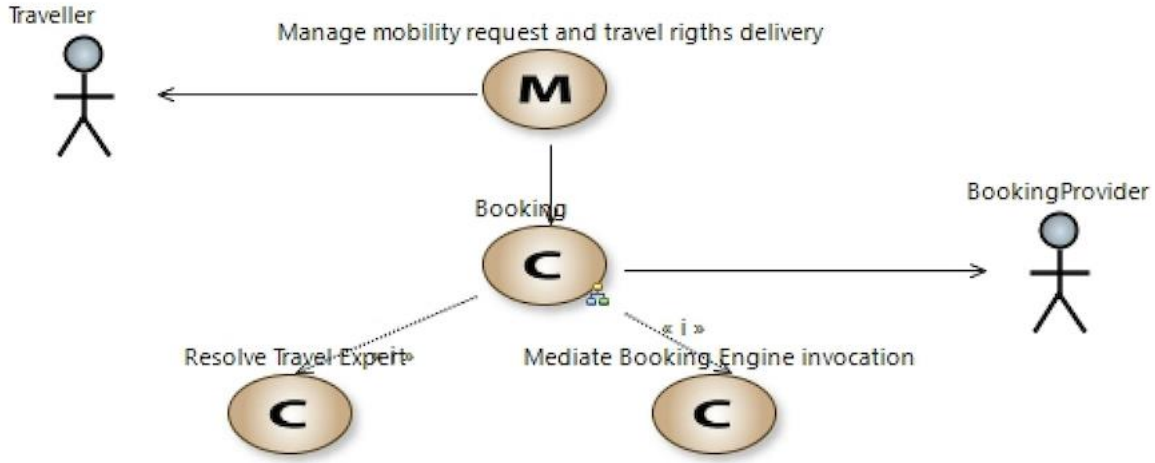


Figure 5: [MCB] Manage Mobility request and travel rights (booking)

3.2.2 IF in Track Itinerary Use Case

The Interoperability Framework provides capabilities included in the Activate Tracking capability of the “Track Itinerary “Use Case.

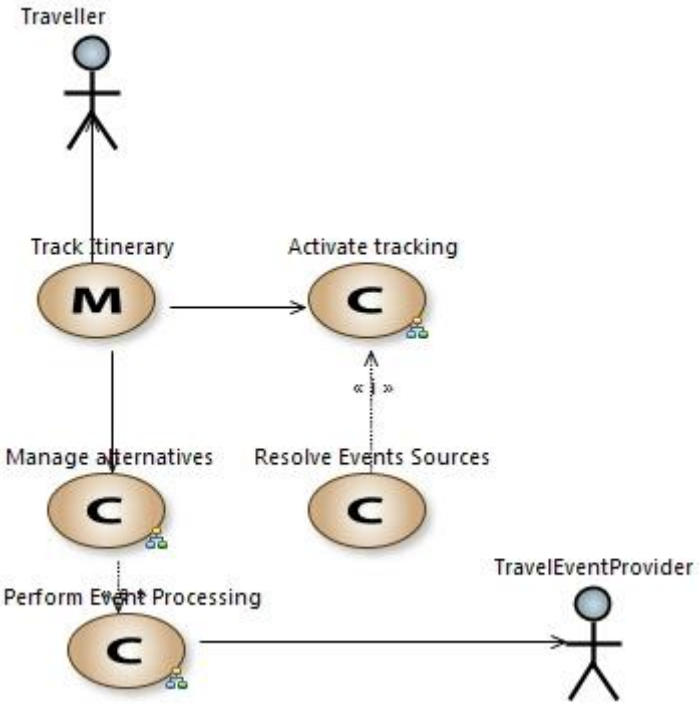


Figure 6: [MCB] Track Itinerary

The Interoperability Framework participates in the Use Case by providing the Resolve Events Source capability to the Use Case. This capability is described in the “Capabilities” chapter of this document.

3.2.3 Manage IF Assets and Publish Resources

In addition to participating in the “Manage Mobility and travel rights delivery” and “Trip Tracking” Use Cases, the Interoperability Framework provides capabilities to execute the following specific Use Cases:

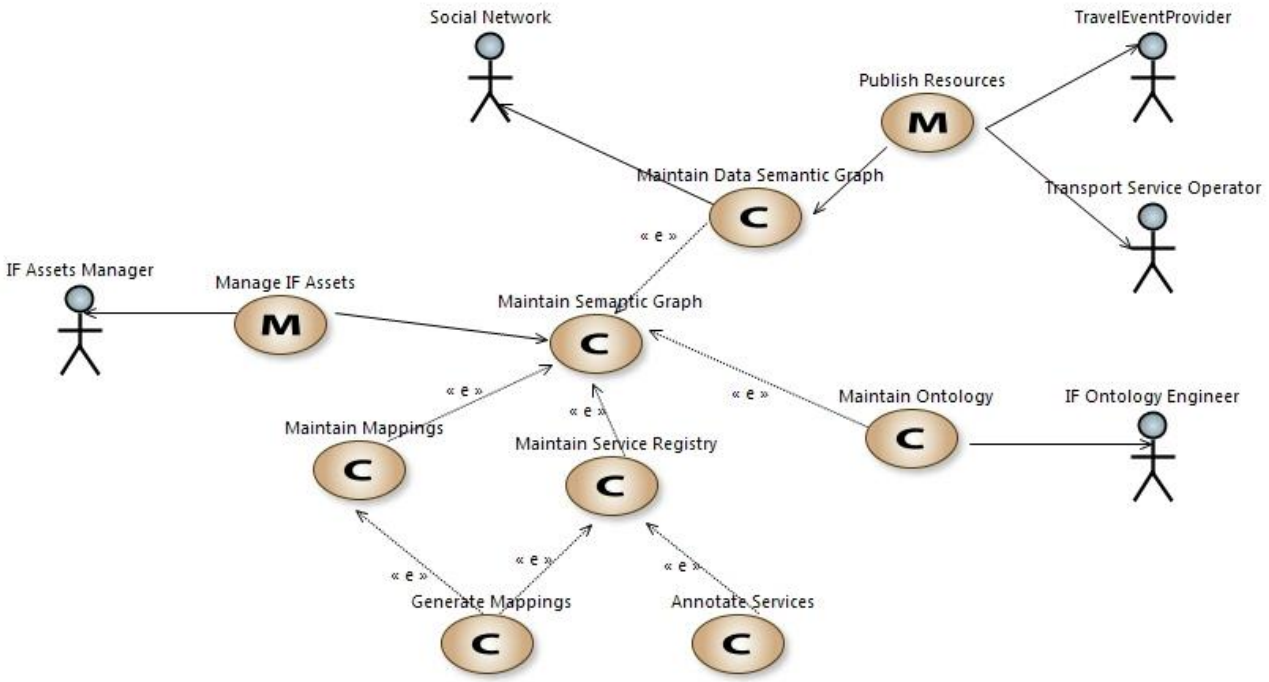


Figure 7: [MCB] IF Assets Management Capabilities

Manage IF Assets

The Manage IF Assets Use Case is concerned with the maintenance of semantic resources throughout the Interoperability Framework, consisting of:

- The IT2Rail ontology in the Ontology Repository.
- Semantically Annotated Web Services in the Semantic Web Service registry.
- Distributed semantic graphs of data, i.e. the web of transportation data.
- Schemas and mappings to support semantic transformation of data and messages to/from heterogeneous data / service providers.

Publish Resources

The Publish Resources Use Case is concerned with the provision of capabilities enabling external Data Providers to publish and semantically annotate data and service descriptor resources with the terms of the IT2Rail ontology.

4.1.1 Manage Semantic Graph

Manage Semantic Graph is the main IF asset management capability offered to the IF Asset Manager Actor to maintain semantic assets in the form of triples through the execution of an approval and versioning process.
It is extended for maintenance of specific resources.

4.1.2 Maintain Ontology

Extends Manage Semantic Graph to support management of the Ontology by the Ontology Engineer Actor.

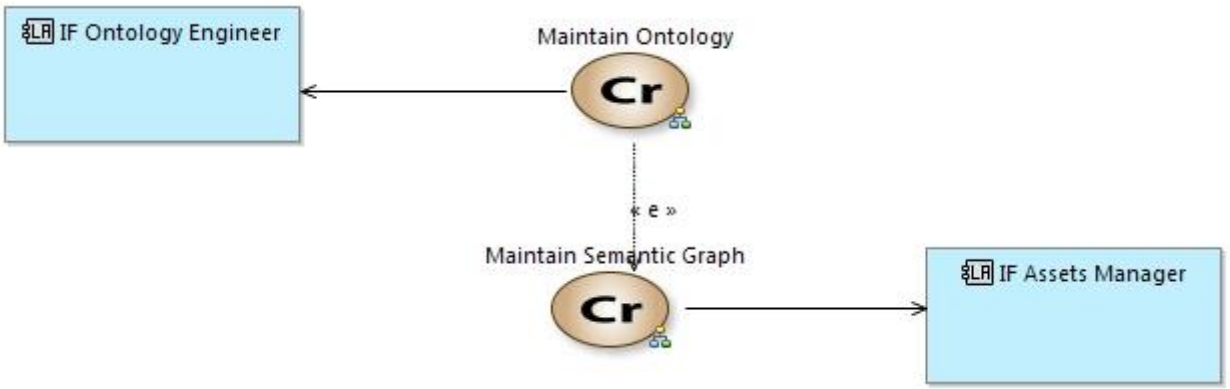


Figure 8: [CRB] Maintain Ontology

4.1.3 Maintain Service registry

Extends Manage Semantic Graph to support the maintenance of the semantic web service registry by the Annotation Engineer Actor operating on web service descriptors published by external Transport Service Operators and Travel Event Provider actors.

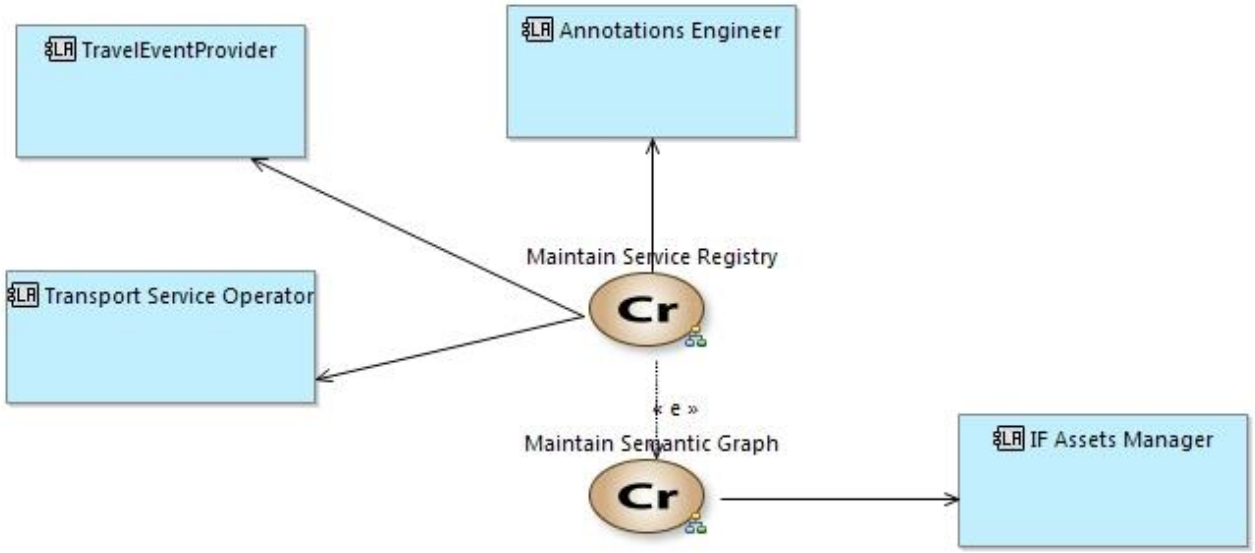


Figure 9: [CRB] Maintain Service Registry

4.1.4 Maintain Data Semantic Graph

Extends Manage Semantic Graph to support the importation and transformation into triples in the RDF format by the Annotations Engineer Actor of external data sources provided by Transport Service Operator, Travel Event Provider and Social Network external Actors.

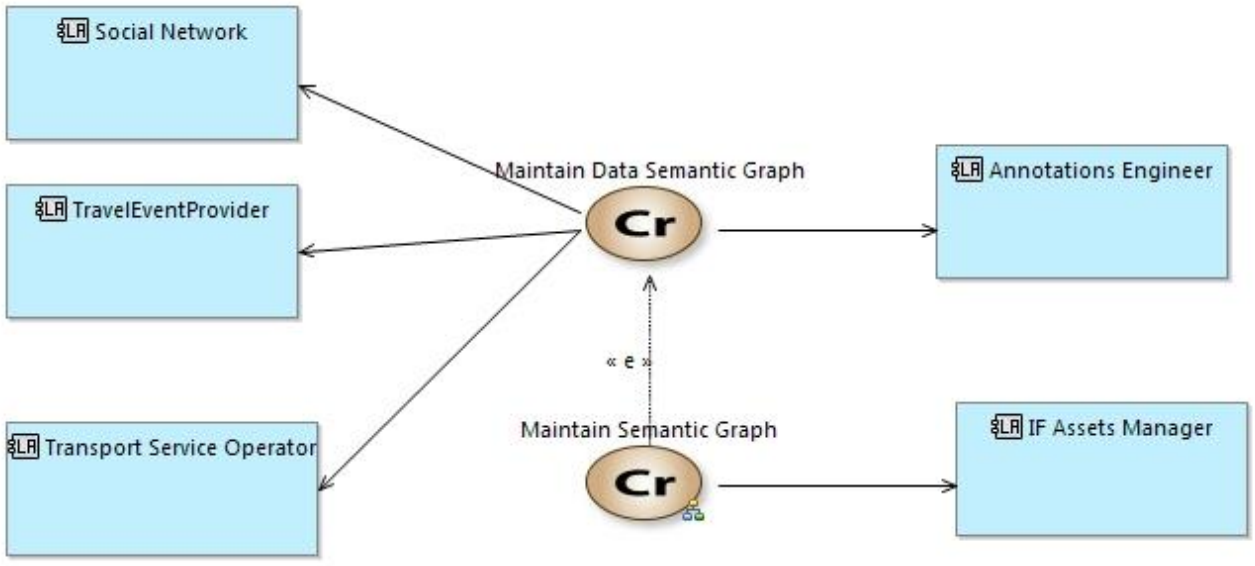


Figure 10: [CRB] Maintain Data Semantic Graph

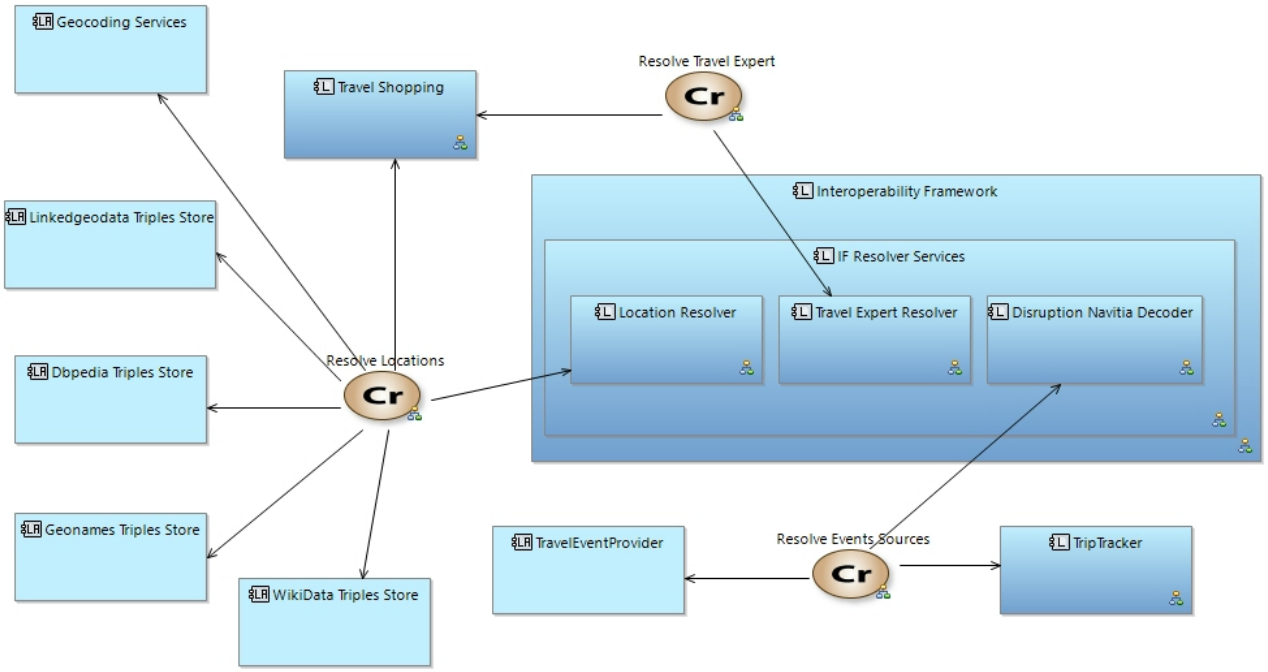


Figure 11: [CBR] IF Resolver Services

4.2.1 Resolve Locations

Resolve Locations is a capability used by the Travel Companion and Travel Shopping logical components to – respectively - identify and obtain Stop Places associated with a mobility request from a distributed semantic graph spanning the Interoperability Framework's internal and external Triple Stores.

4.2.2 Resolve Travel Expert

Resolve Travel Expert is a capability used by the Travel Shopping logical component to identify and obtain Travel Expert service descriptors associated with meta-travel expert episodes from a distributed semantic graph spanning the Interoperability Framework's internal and external Triple Stores.

4.2.3 Resolve Events Source

Resolve Events Source is a capability used by the Trip Tracking logical component to identify and obtain Events Source service descriptors associated with journeys from a distributed semantic graph spanning the Interoperability Framework's internal and external Triple Stores.

4.2.4 Provide Network Statistics Data

The Interoperability Framework provides a capability to generate "Network Statistics Data" out of the distributed semantic graph, i.e. a list of transportation connections for the different modes of transport, each associated with fundamental "statistics" such as the operating

times, average connection times, frequency and density of service. Statistics data are additionally associated with the Travel Expert service that can provide *actual* details and offers at the time of a shopping session. The statistics data is used by the Travel Shopper to build a "meta-network" over which smart algorithms may be run to identify candidate routes between any two points before submitting them to Travel Experts for detailed scheduling.

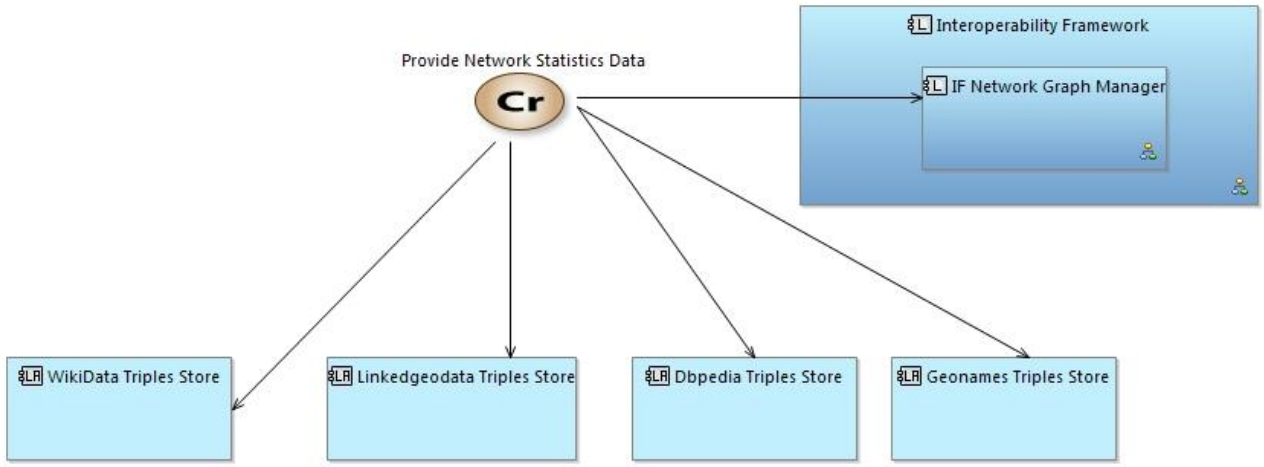


Figure 12: [CBR] Provide Network Statistics Data capability

4.3 MEDIATE TRAVEL EXPERT INVOCATION

Mediate Travel Expert Invocation is a capability used by the Travel Shopping logical component to obtain OfferItems from external providers (Travel Experts) using semantic brokering to transform the remote interfaces to/from the IT2Rail ontology specification.
This capability realises one of the possible deployable architectures of the Interoperability Framework.

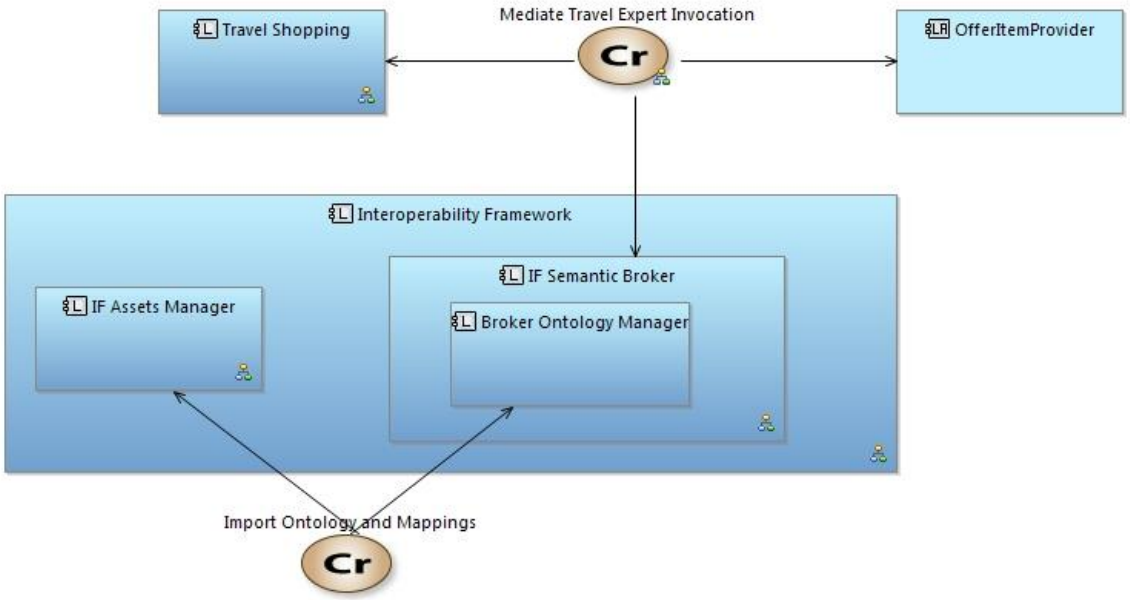


Figure 13: [CRB] IF Mediate Travel Expert Invocation (shopping)

The same capability is used by the Booking and Ticketing logical component to perform bookings and issuance operations at external providers (Booking Provider) using semantic brokering to transform the remote interfaces to/from the IT2Rail ontology specification.
This capability also realises one of the possible deployable architectures of the Interoperability Framework.

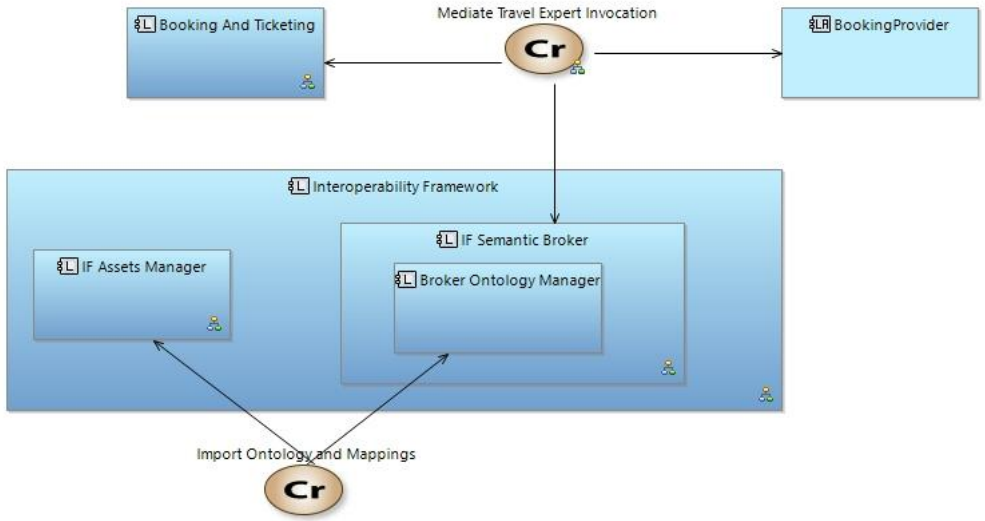


Figure 14: [CRB]IF Mediate Travel Expert Invocation (booking)

4.3.1 Import Ontology and Mappings

Import Ontology and Mappings is a capability used by the Semantic Broker to obtain the ontology and mappings from the Interoperability Framework ontology repository, and deploy them locally for processing.

5. ACTIVITY DIAGRAMS

5.1 MANAGE IF ASSETS

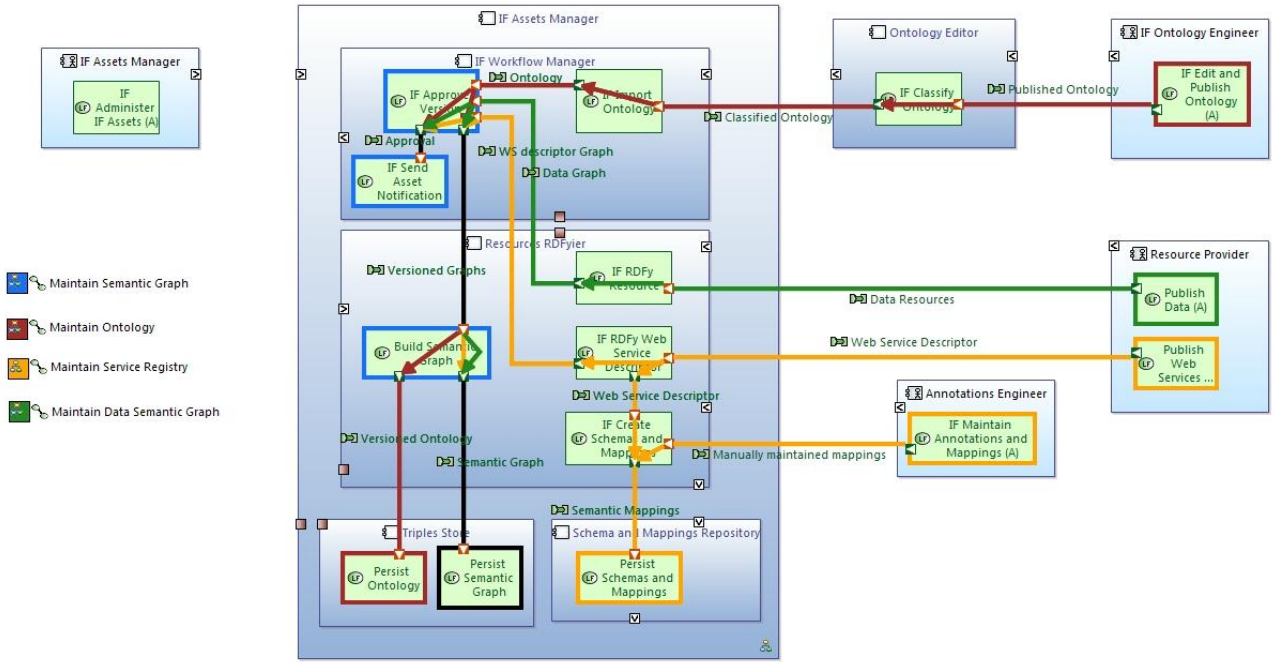


Figure 15: Manage IF Assets activity diagram

5.1.1 Functions in Maintain Semantic Graph realisation

Logical Function	Description	Component
IF Approve Version	Perform approval process on submitted resources	WorkFlow Manager
Build Semantic Graph	Create semantic graph from approved resources to be stored in triple store	Resource RDFyer
IF Send Asset Notification	Send a notification of successful resource approval to registered recipients	WorkFlow Manager

5.1.2 Functions in Maintain Ontology Realisation

Logical Function	Description	Component
IF Edit and Publish Ontology (A)	Activity performed by Ontology Engineer to publish Ontology	Ontology Engineer (actor)
IF Classify Ontology	Ontology is classified with inference engine to materialise inferred triples and provide ontology validation	Ontology Editor
IF Import Ontology	Published and classified ontology is imported into workflow management	WorkFlow Manager

Perform Maintain Semantic Graph realisation activities		
Persist Ontology	Persist Ontology as semantic graph in triples store	Triples Store

5.1.3 Functions in Maintain Service Registry realisation

Logical Function	Description	Component
IF Publish Web Services (A)	Activity performed by Resource Provider to publish Web Service descriptor files	Resource Provider (external Actor)
IF RDFy Resource	Transform Web service descriptors into RDF triples	Resource RDFyer
Perform Maintain Semantic Graph realisation activities		
IF Maintain Annotations and Mappings (A)	Create and maintain semantic annotations, mappings and schemas	Annotations Engineer (Actor)
Persist semantic graph	Persist Web Service descriptors as semantic graph in triples store	Triples Store
Persist Schemas and Mappings	Persist mappings and schema files in web server repository	Schema and Mappings Repository

5.1.4 Functions in Maintain Data Semantic Graph realisation

Logical Function	Description	Component
Publish Data (A)	Activity performed by Resource Provider to publish data resources, e.g. Stop Place information	Resource Provider (external Actor)
IF RDFy Resource	Transform data into RDF triples	Resource RDFyer
Perform Maintain Semantic Graph realisation activities		
Persist semantic graph	Persist data as semantic graph in triples store	Triples Store

5.2.1 Location Identification

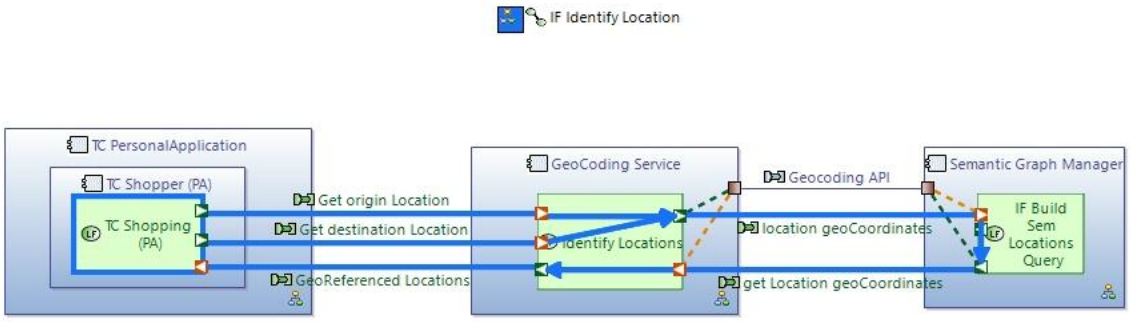


Figure 16 [LAB] IF Location Identification

Functions in Location Identification realisation

Logical Function	Description	Component
Identify Locations	Uses origin and destination string names entered by the Traveller on the Travel Companion Personal Application and invokes Semantic Graph Manager to identify locations with their geo-coordinates	GeoCoding Service
IF Build sem Locations Query	Performs a distributed query over the world wide web to retrieve location geo-coordinates	Semantic Graph Manager

5.2.2 Provide Network Statistics Data

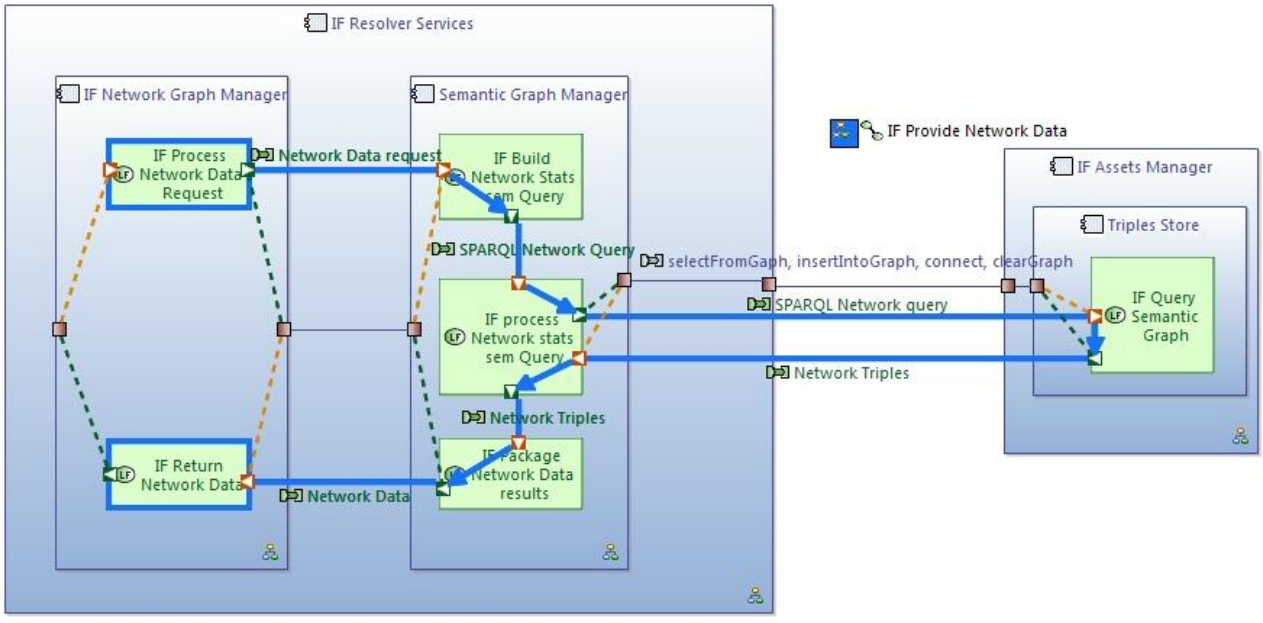


Figure 17: Provide Network Statistics Data activity diagram

Functions in Provide Network Data capability realisation

Logical Function	Description	Component
IF Process Network Data Request	Receives and validates Network Data Request	Network Graph Manager
IF Build Network stats sem Query	Transforms network data request and prepares network data semantic query	Semantic Graph Manager
IF Process Network stats sem Query	Connects to Triple Stores and submits network data semantic query for processing	Semantic Graph Manager
IF Query Semantic Graph	Queries semantic graph and retrieves result triples	Triple Store
IF Package Network Data Results	Processes triples results set and generates Network Data results	Semantic Graph Manager
IF Return Network Data	Generates Network Data response	Network Graph Manager

5.2.3 Resolve Locations

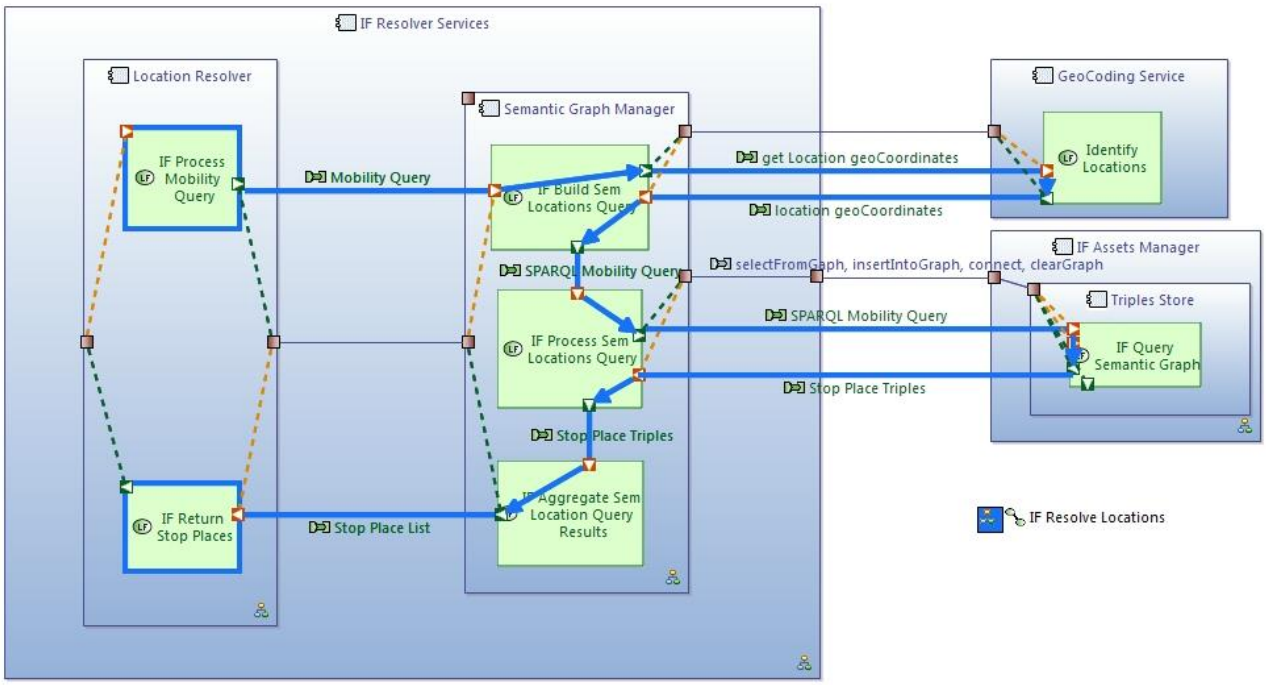


Figure 18: Resolve Locations activity diagram

Functions in Resolve Locations capability realisation

Logical Function	Description	Component
IF Process Mobility Query	Receives and validates mobility request	Location Resolver
Identify Locations	Gets geo-coordinates for mobility request	GeoCoding Service
IF Build Sem Locations Query	Transforms mobility request and prepares locations semantic query	Semantic Graph Manager
IF Process Sem Locations Query	Connects to Triple Stores and submits locations semantic query for processing	Semantic Graph Manager
IF Query Semantic Graph	Queries semantic graph and retrieves result triples	Triple Store
IF Aggregate Sem Location Query Results	Processes triples results set and generates Stop Place results	Semantic Graph Manager
IF Return Stop Places	Generates list of Stop Place response	Locations Resolver

5.2.4 Resolve Travel Expert

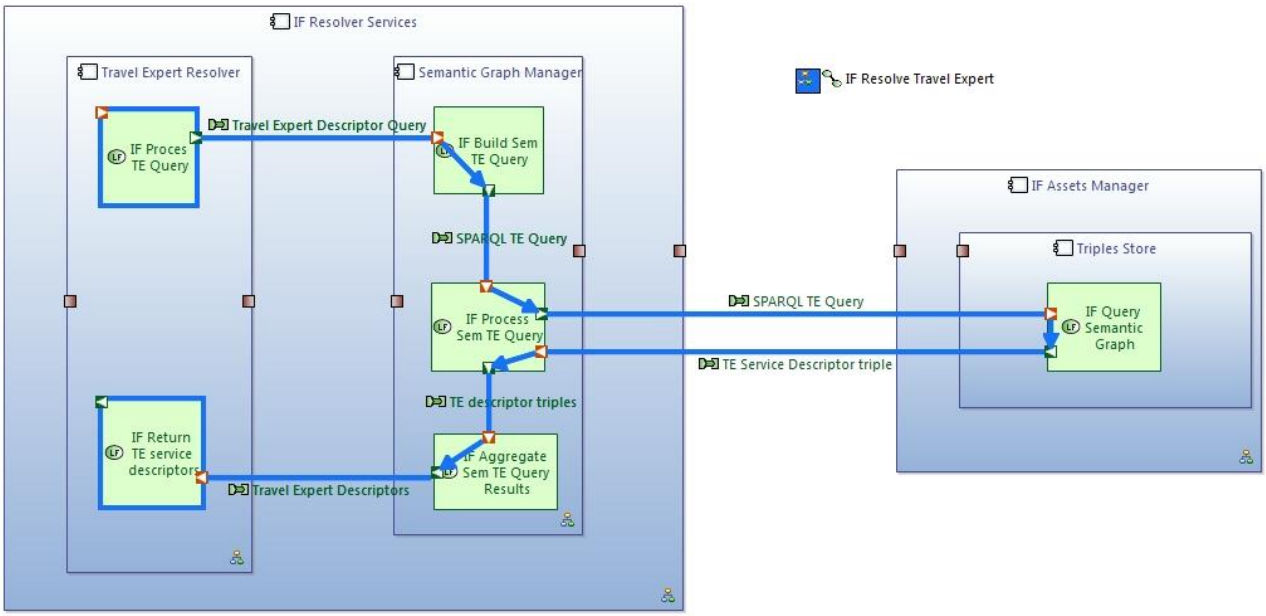


Figure 19: Resolve Travel Expert activity diagram

Functions in Resolve Travel Expert capability realisation

Logical Function	Description	Component
IF Process TE Query	Receives and validates Travel Expert request	Travel Expert Resolver
IF Build Sem TE Query	Transforms travel expert request and prepares travel expert semantic query	Semantic Graph Manager
IF Process Sem TE Query	Connects to Triple Stores and submits travel expert semantic query for processing	Semantic Graph Manager
IF Query Semantic Graph	Queries semantic graph and retrieves result triples	Triple Store
IF Aggregate Sem TE Query Results	Processes triples results set and generates travel expert descriptor results	Semantic Graph Manager
IF Return TE service descriptors	Generates list of Travel Expert descriptor response	Travel Expert Resolver

5.2.5 Resolve Events Source

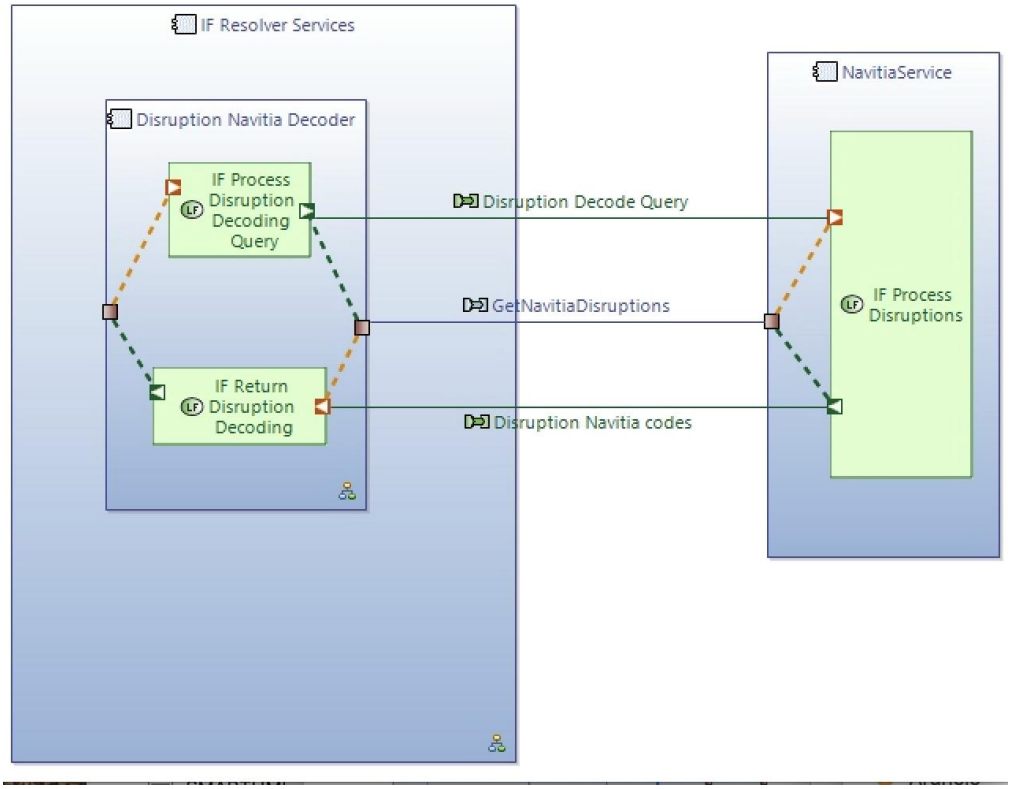


Figure 20: Resolve Events Source activity diagram

Functions in Resolve Events Source capability realisation

Logical Function	Description	Component
IF Process Disruption Decode Query	Receives and validates Confirmed Booking from Trip Tracker	Disruption Navitia Decoder
IF Process Disruptions	Performs a query and retrieves Stop Place and Vehicle codes from the Navitia platform at https://www.navitia.io/ for use by the Trip Tracker	Navitia Service
IF Return Disruption Decoding	Returns Confirmed Booking with Navitia encoding of Stop Places and Vehicles	Disruption Navitia Decoder

5.3 IF SEMANTIC BROKER

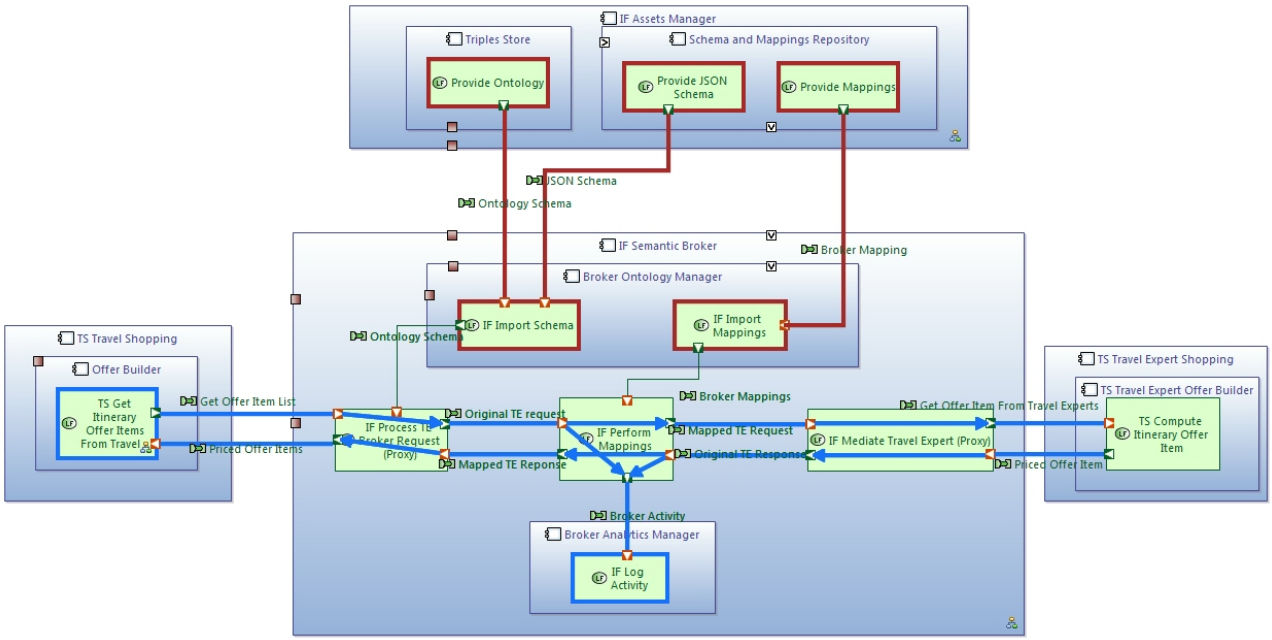


Figure 21: IF Semantic Broker activity diagram

Functions in Mediate Travel Expert Invocation capability realisation

Logical Function	Description	Component
IF Process TE Broker Request (Proxy)	Receives and validates a request for OfferItems from OfferBuilder (WP2 defined function), or booking/issuance requests from Booking and Ticketing (WP3 components), and binds to JSON schema ontology	Semantic Broker
IF Perform Mappings	Applies transformation of request to target Travel Expert interface using imported mappings from Broker Ontology	Semantic Broker
IF Log Activity	Logs input message and transformation	Broker Analytics Manager
IF Mediate Travel Expert (Proxy)	Invokes remote Travel Expert with transformed interface	Semantic Broker
IF Mediate Travel Expert (Proxy)	Receives remote Travel Expert response with travel expert specific OfferItems, or Bookings	Semantic Broker
IF Perform Mappings	Applies transformation of response from Travel Expert interface using imported mappings from Broker Ontology	Semantic Broker
IF Log Activity	Logs response and fault message and transformation	Broker Analytics Manager
IF Process TE Broker Request (Proxy)	Creates response containing OfferItems for OfferBuilder (WP2 defined function), or Bookings / Entitlements for Booking and Ticketing (WP3 component)	Semantic Broker

Functions in Manage Broker Ontology capability realisation

Logical Function	Description	Component
Provide Ontology	Performs query on Triple Store to find JSON Schema for requested Travel Expert	Triple Store
Provide JSON Schema	Serves JSON schema	Schema and Mappings Repository
IF Import Schema	Imports identified JSON Schema from repository	Broker Ontology Manager
Provide Mappings	Imports identified transformation mappings from repository	Schema and Mappings Repository
IF Import Mappings	Invokes remote Travel Expert with transformed interface	Broker Ontology Manager

6. COMPONENTS AND COMPONENT EXCHANGES

6.1 IF ASSET MANAGER COMPONENT EXCHANGES

The following figure describes the exchanges across components of the IF Asset Manager. Exchanges are realised through the interfaces described in the Interfaces chapter of this document

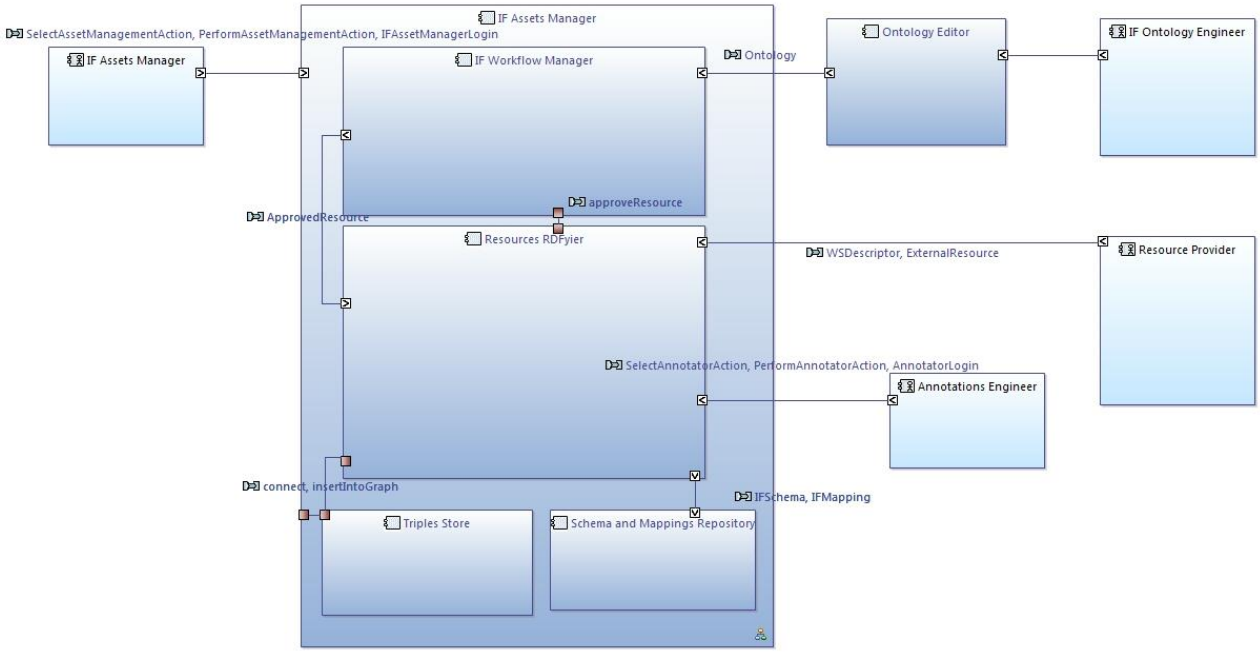


Figure 22: Exchanges across components of the IF Asset Manager

6.2 IF RESOLVER SERVICES COMPONENT EXCHANGES

The following figure describes the exchanges across components of the IF Resolver Services. Exchanges are realised through the interfaces described in the Interfaces chapter of this document

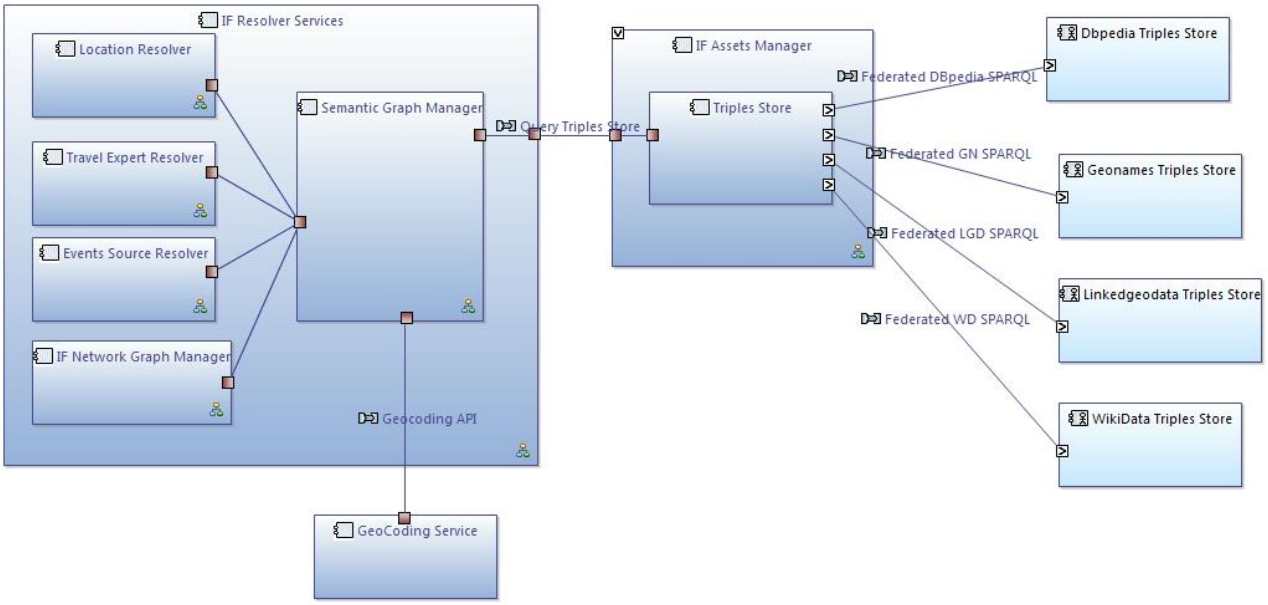


Figure 23: Exchanges across components of the IF Resolver Services

6.3 SEMANTIC BROKER COMPONENT EXCHANGES

The following figure describes the exchanges across components of the IF Semantic Broker. Exchanges are realised through the interfaces described in the Interfaces chapter of this document.

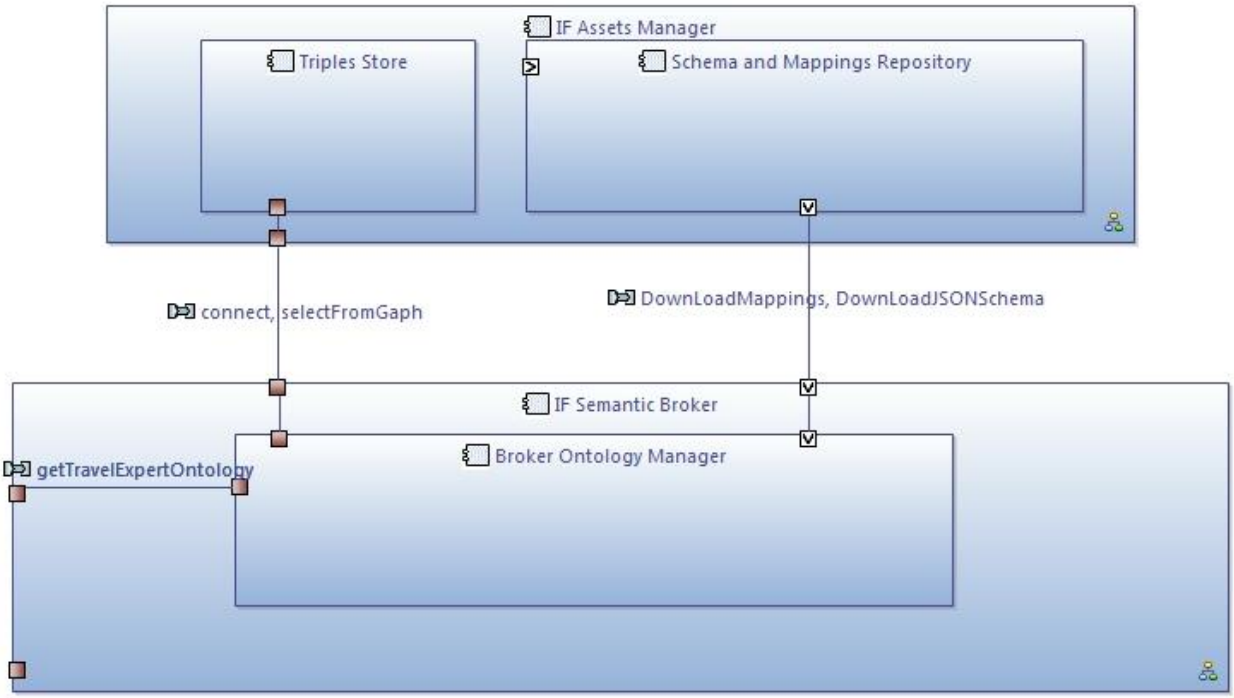


Figure 24: Exchanges across components of the IF Semantic Broker

7. INTERFACES

7.1 IF ASSETS MANAGER

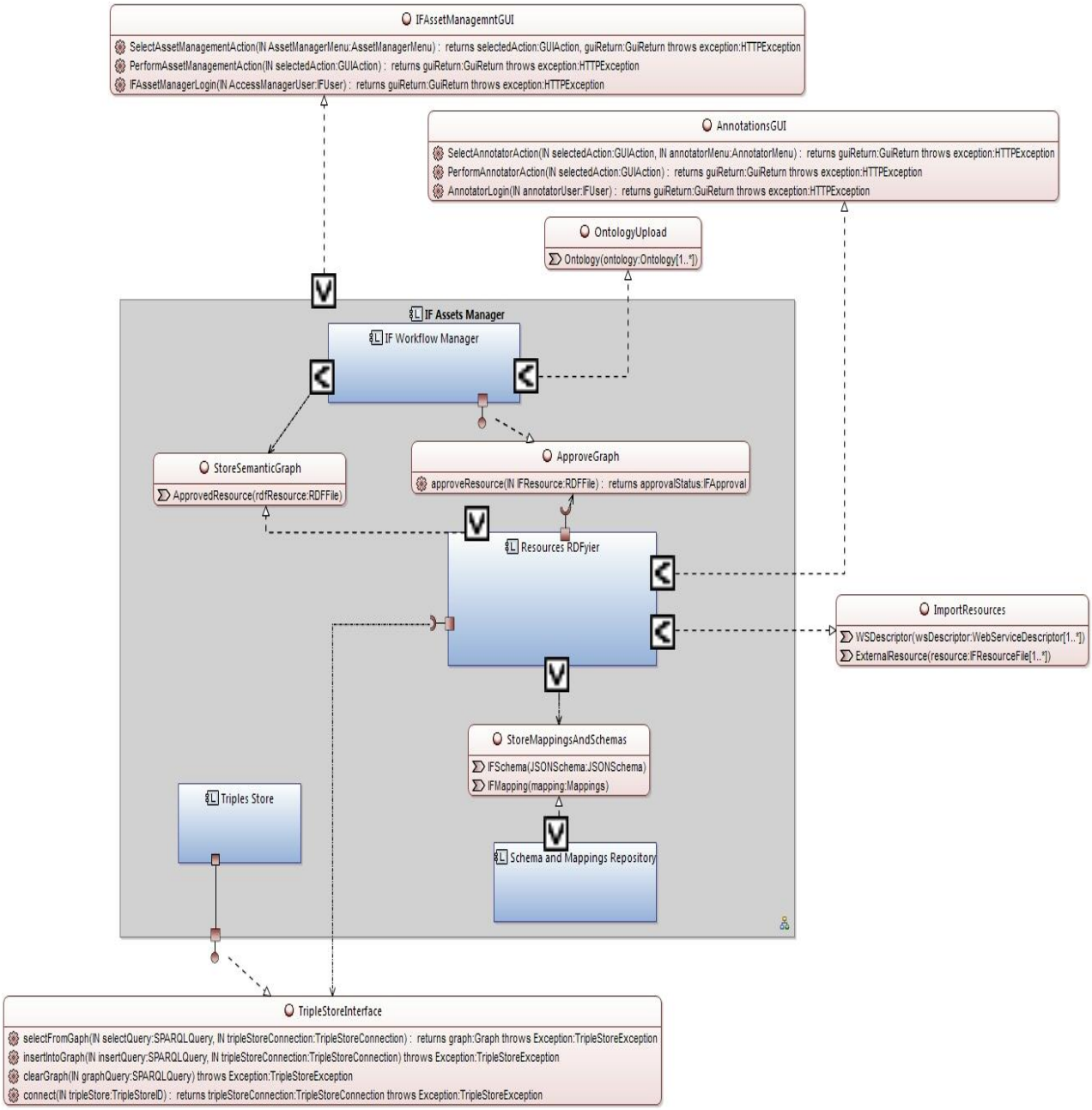


Figure 25: IF Assets Manager interfaces

7.1.1 External Interfaces

Table 2: Asset Management GUI interface

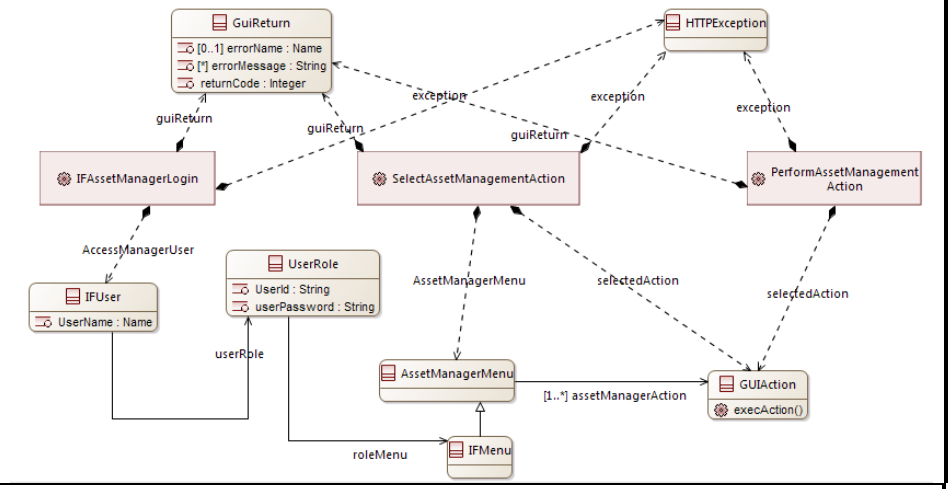
Interface ID:	**Asset Management GUI**
Interface Name:	IFAssetManagemntGUI SelectAssetManagementAction(IN AssetManagerMenu:AssetManagerMenu) : returns selectedAction:GUIAction, guiReturn:GuiReturn throws exception:HTTPException PerformAssetManagementAction(IN selectedAction:GUIAction) : returns guiReturn:GuiReturn throws exception:HTTPException IFAssetManagerLogin(IN AccessManagerUser:IFUser) : returns guiReturn:GuiReturn throws exception:HTTPException
Purpose of the Interface	Graphical User Interface used by Asset Manager role to approve and maintain interoperability framework assets, e.g. ontology repository, semantic web services registry, mappings and JSON schemas
Requestor:	Interoperability Framework Asset Manager (Actor)
Provider	IF Asset Manager
Description:	Interoperability Framework Asset Manager graphical user interface
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	IF Asset Manager account and role created in IF Asset Manager
Postconditions:	n/a
Exchange Items	
Exceptions:	Inexistent account, account not authorised
Notes and Issues:	Standard, open source work flow and content management software to be used for implementation

Table 3: Ontology Upload interface

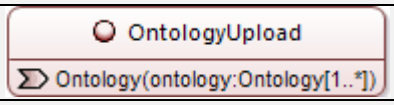
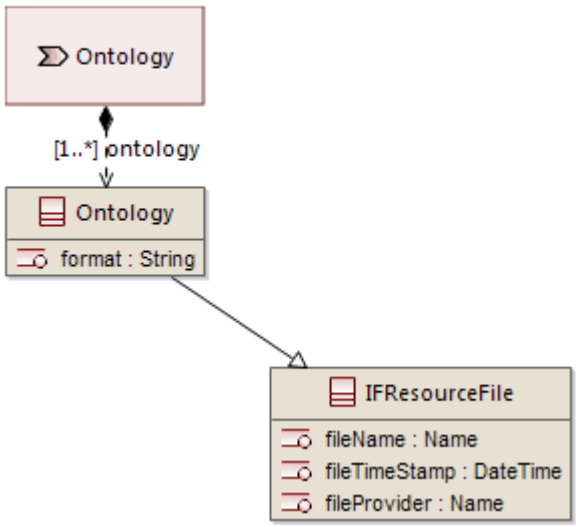
Interface ID:	**Ontology Upload**
Interface Name:	
Purpose of the Interface	Upload validated ontology files edited by external Ontology Editor for approval in the approval workflow process
Requestor:	Ontology Editor
Provider	Workflow Manager
Description:	Upload validated ontology files edited by external Ontology Editor
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Ontology files have been validated prior to upload
Postconditions:	Ontology files successfully uploaded
Exchange Items	
Exceptions:	Invalid ontology file
Notes and Issues:	Standard, open source work flow and content management software to be used for implementation

Table 4: Annotations GUI interface

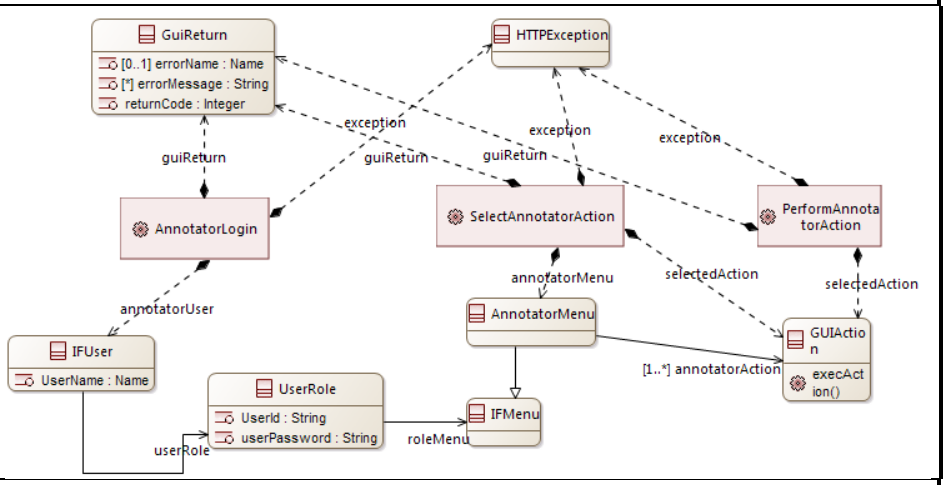
Interface ID:	**Annotations GUI**
Interface Name:	AnnotationsGUI SelectAnnotatorAction(IN selectedAction:GUIAction, IN annotatorMenu:AnnotatorMenu) : returns guiReturn:GuiReturn throws exception:HTTPException PerformAnnotatorAction(IN selectedAction:GUIAction) : returns guiReturn:GuiReturn throws exception:HTTPException AnnotatorLogin(IN annotatorUser:IFUser) : returns guiReturn:GuiReturn throws exception:HTTPException
Purpose of the Interface	Graphical User Interface used by Annotations engineer role to create/maintain semantic annotations and schemas on imported non semantic resources
Requestor:	Annotations Engineer (Actor)
Provider	Resource RDFyer
Description:	IF Asset Manager graphical user interface for Annotations Engineer
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Annotations Engineer account and role created in IF Asset Manager
Postconditions:	n/a
Exchange Items	
Exceptions:	Inexistent account, account not authorised
Notes and Issues:	Standard, open source rdfyzer software to be used for implementation

Table 5: Import External Resources interface

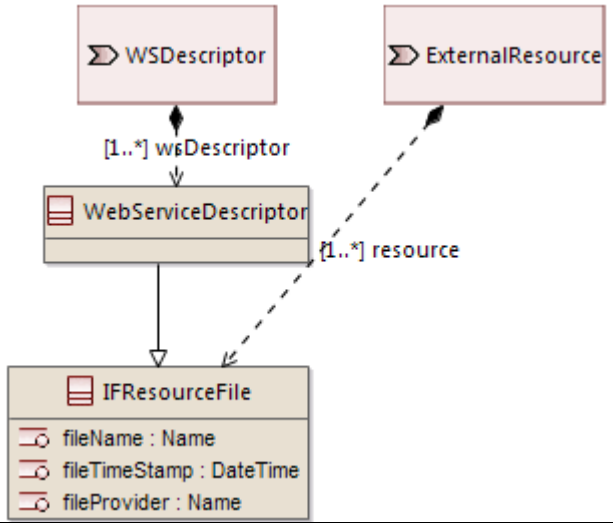
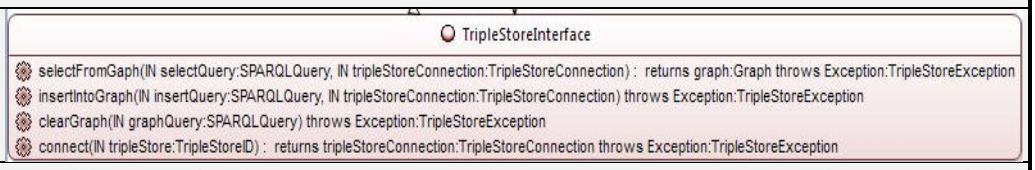
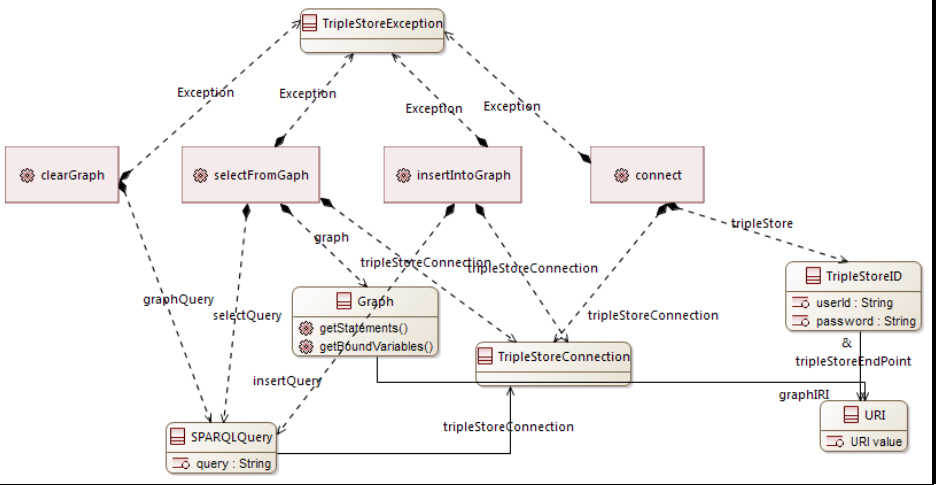
Interface ID:	**Import External Resources**
Interface Name:	
Purpose of the Interface	Import external resources, e.g. data, web service descriptors, for semantic annotation and conversion into semantic graph
Requestor:	Resource Provider (external Actor)
Provider	Resource RDFyer
Description:	Import external resources, e.g. data, web service descriptors, for semantic annotation and conversion into semantic graph
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	External resources provided in files in CSV, RDF, n3, turtle, JSON, XML formats. Data quality to be ensured by Resource Provider
Postconditions:	External resources successfully imported
Exchange Items	
Exceptions:	Invalid resource file format
Notes and Issues:	Additional notes, issues and comments

Table 6: Triples Store Interface

Interface ID:	**Triples Store Interface**
Interface Name:	
Purpose of the Interface	Provides methods to connect and perform CRUD operations on triple store
Requestor:	Resources RDFyer, Semantic Broker, IF Resolver Services
Provider	IF Asset Manager through delegation to Triple Store
Description:	Provides methods to connect and perform CRUD operations on triple store
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Triple Store service running
Postconditions:	Specific to methods, no Triple Store Exception
Exchange Items	
Exceptions:	TripleStoreException, specific to interface implementation
Notes and Issues:	Interface is abstract, implemented through specific triple store connectors and drivers, e.g. Apache Jena, Virtuoso Jena, Sesame

7.1.2 Internal Interfaces

Table 7: Approve Graph interface

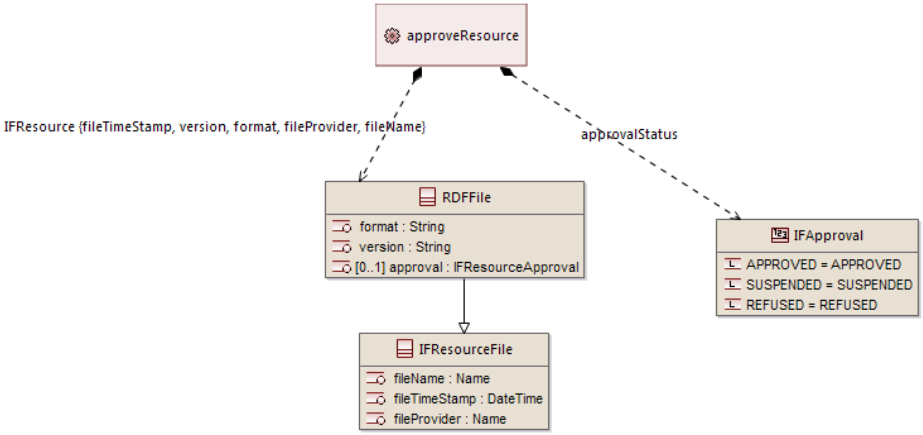
Interface ID:	**Approve Graph**
Interface Name:	
Purpose of the Interface	Submit resource to approval workflow process
Requestor:	Resource RDFyer
Provider	Workflow Manager
Description:	Submit resource to approval workflow process
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	RDFyed resource validated by Resource RDFyer
Postconditions:	Resource approved, suspended or refused
Exchange Items	
Exceptions:	Invalid RDF resource
Notes and Issues:	Standard, open source workflow management software to be used for implementation

Table 8: Store Semantic Graph interface

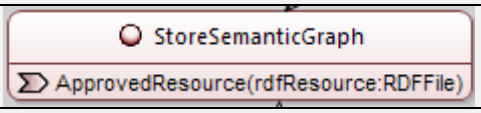
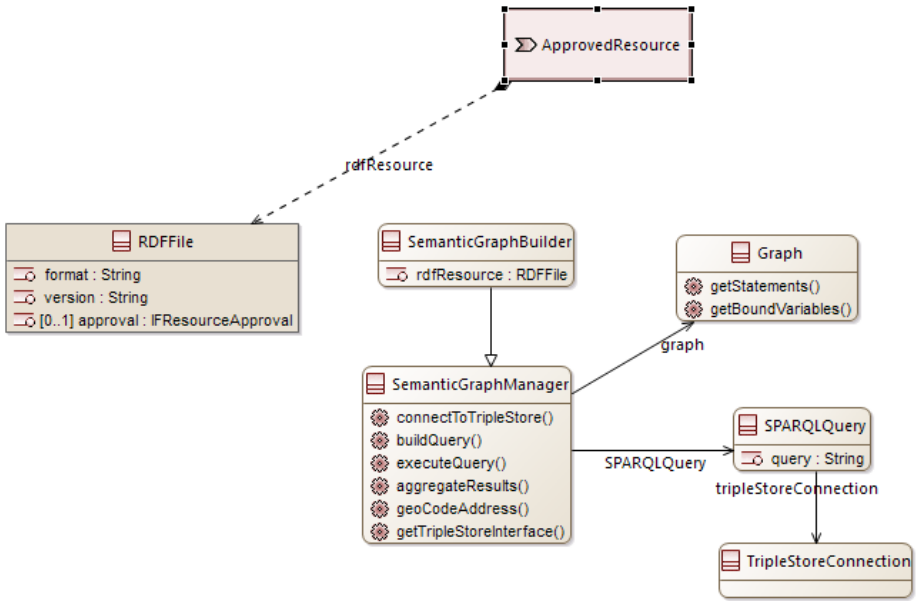
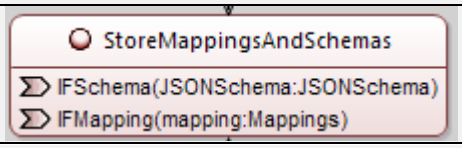
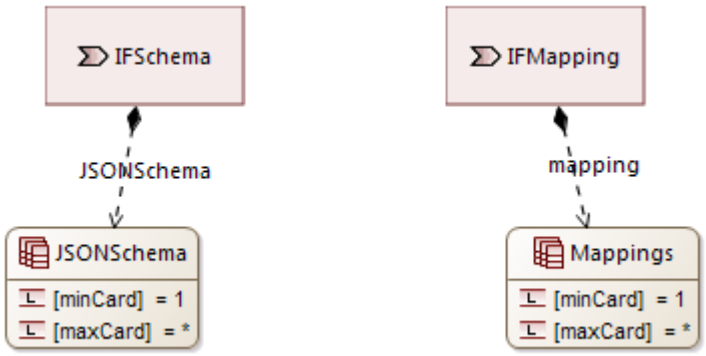
Interface ID:	**Store Semantic Graph**
Interface Name:	
Purpose of the Interface	Store approved RDF resource as semantic graph in triple store
Requestor:	WorkFlow Manager
Provider	Resource RDFyer
Description:	Store approved RDF resource as semantic graph in triple store
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	ValidationStatus indicates passed validation, token to be updated retrieved and altered with updated payload
Postconditions:	Token successfully retrieved and altered in Travel Companion.
Exchange Items	
Exceptions:	Which exceptions could appear
Notes and Issues:	Additional notes, issues and comments

Table 9: Store Mappings and Schemas interface

Interface ID:	**Store Mappings and Schemas**
Interface Name:	
Purpose of the Interface	Store semantic mappings and schemas
Requestor:	Resource RDFyer
Provider	Schema and Mappings Repository
Description:	Store semantic mappings and schemas in semantic repository
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Schemas and mappings validated, repository web server running
Postconditions:	Schemas and mappings persisted in repository
Exchange Items	
Exceptions:	HTTP exceptions from repository web server
Notes and Issues:	Schemas and mappings repository implemented as a web server using standard commercial or open source software

7.2 IF RESOLVER SERVICES

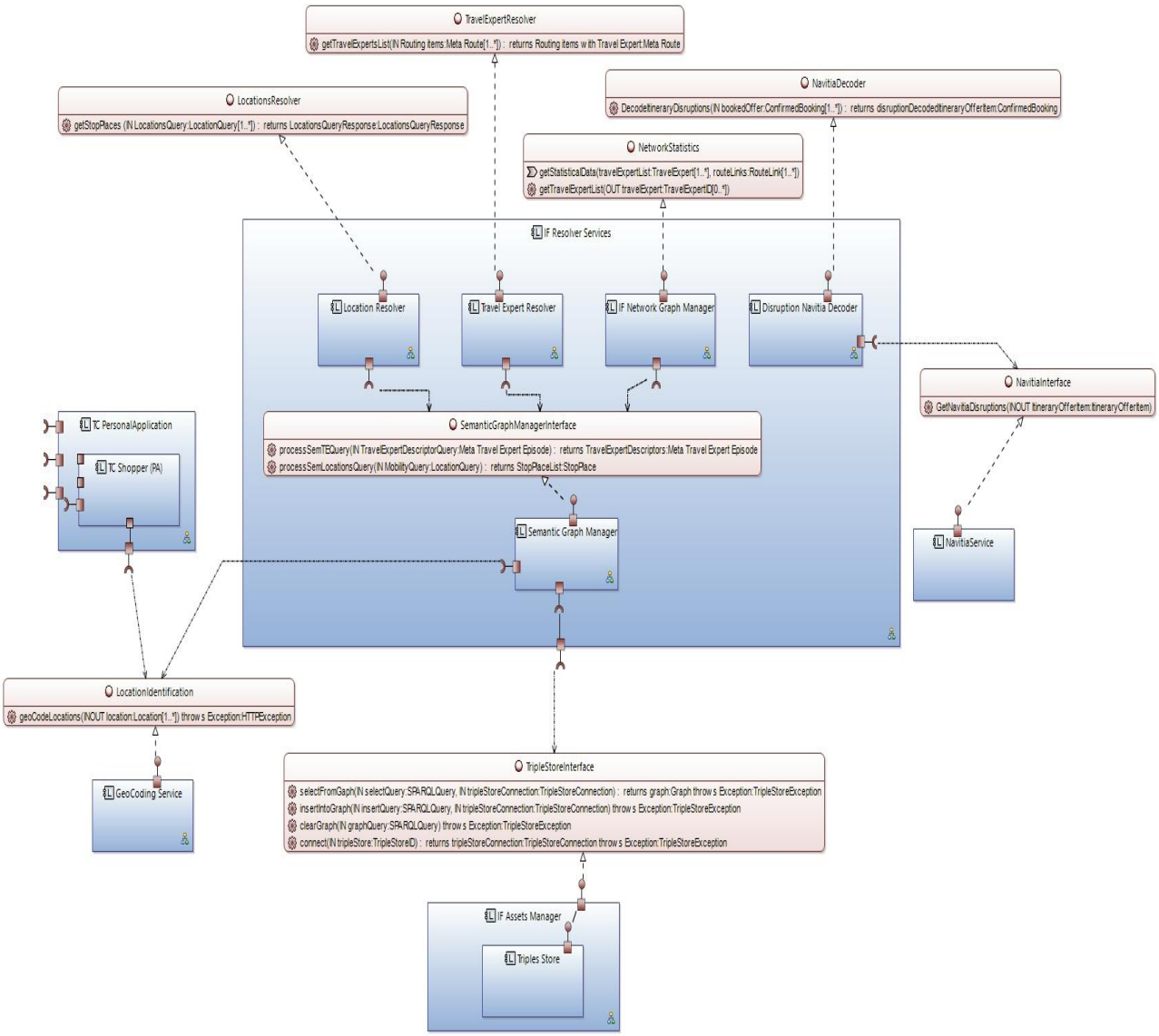


Figure 26: IF Resolver Services interfaces

7.2.1 External Interfaces

Table 10: IdentifyLocations interface

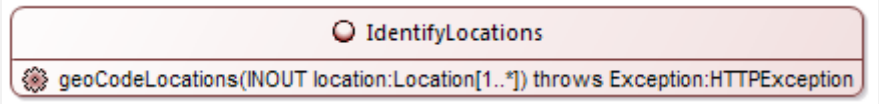
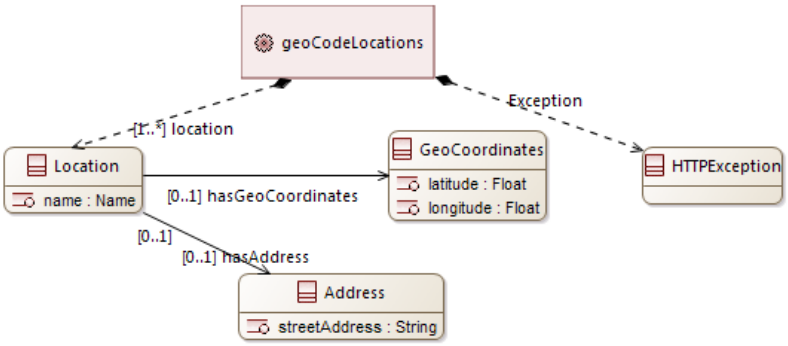
Interface ID:	**IdentifyLocations**
Interface Name:	
Purpose of the Interface	Provide list of Locations with geo coordinates
Requestor:	Travel Companion (WP5 component), Semantic Graph Manager
Provider	GeoCoding service
Description:	Provide list of Locations with geo coordinates
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Url encoded string for location name
Postconditions:	Zero, one or more locations with geo-coordinates returned
Exchange Items	
Exceptions:	Http exception for invalid request
Notes and Issues:	

Table 11: NetworkStatistics interface

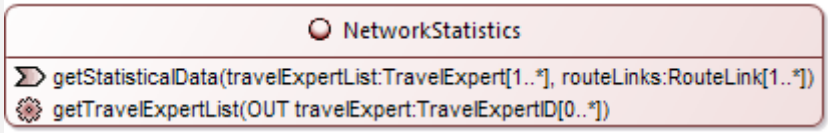
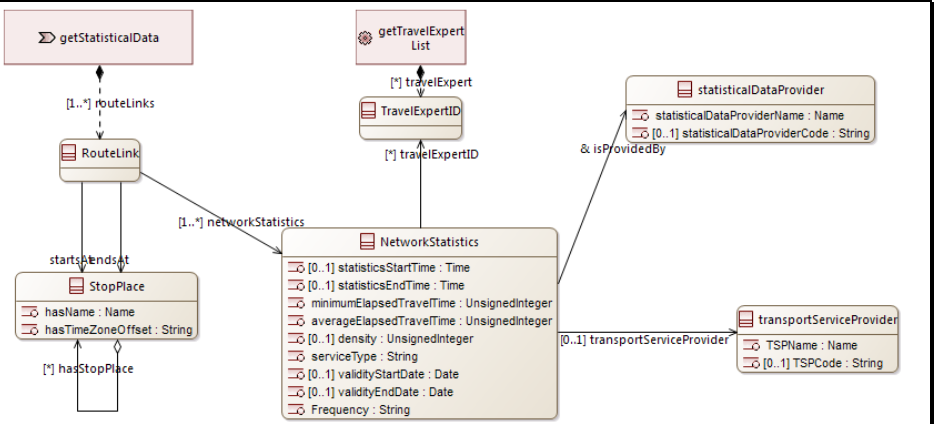
Interface ID:	**NetworkStatistics**
Interface Name:	
Purpose of the Interface	Provide Network data statistic as a list RouteLinks with associated statistics information Provide list of Travel Experts registered in Semantic Web Service Registry
Requestor:	Travel Shopping (WP2 component)
Provider	Network Graph Manager
Description:	getStatisticalData operation: Provide Network data statistic as a list RouteLinks with associated statistics information getTravelExpertList operation: Provide list of Travel Experts registered in Semantic Web Service Registry
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	getStatisticalData operation: Travel Expert Id getTravelExpertList: none
Postconditions:	getStatisticalData operation: zero or more RouteLinks with associated statistics getTravelExpertList operation: zero or more TravelExpertID
Exchange Items	
Exceptions:	Invalid request
Notes and Issues:	

Table 12: Locations Resolver interface

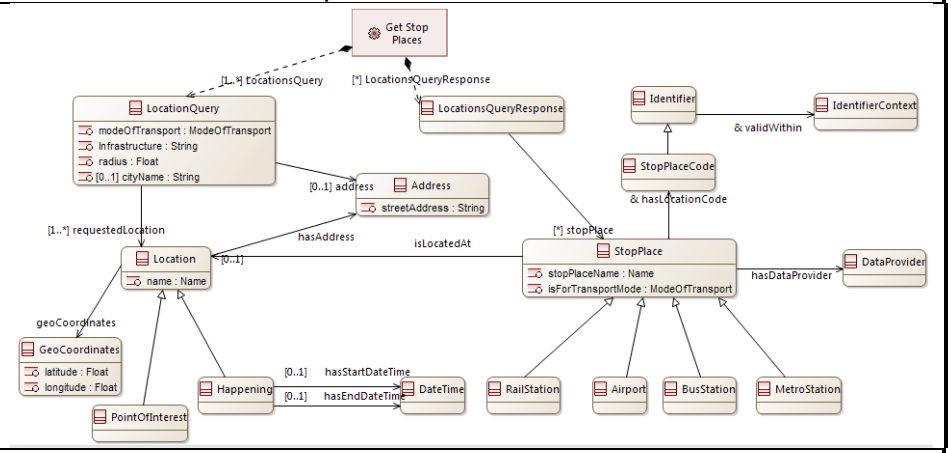
Interface ID:	**Locations Resolver**
Interface Name:	LocationsResolver Get Stop Places (IN LocationsQuery:LocationQuery[1..*]) : returns LocationsQueryResponse:LocationsQueryResponse
Purpose of the Interface	Provide list of Stop Place within a given radius of a Location
Requestor:	Travel Shopping (WP2 component)
Provider	Location Resolver
Description:	Provide list of Stop Place within a given radius of a Location
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Valid LocationQuery
Postconditions:	Zero, one or more StopPlace returned
Exchange Items	
Exceptions:	Invalid request
Notes and Issues:	

Table 13: Travel Expert Resolver interface

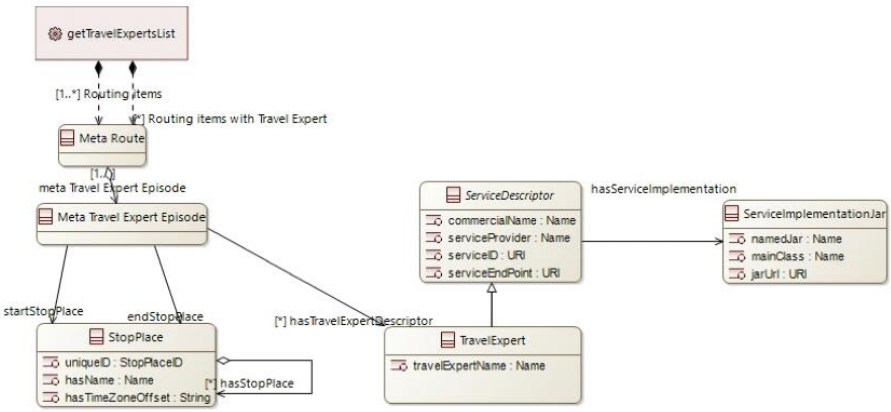
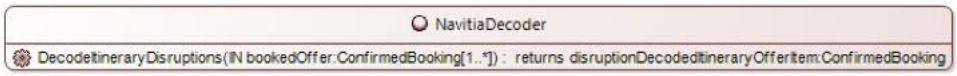
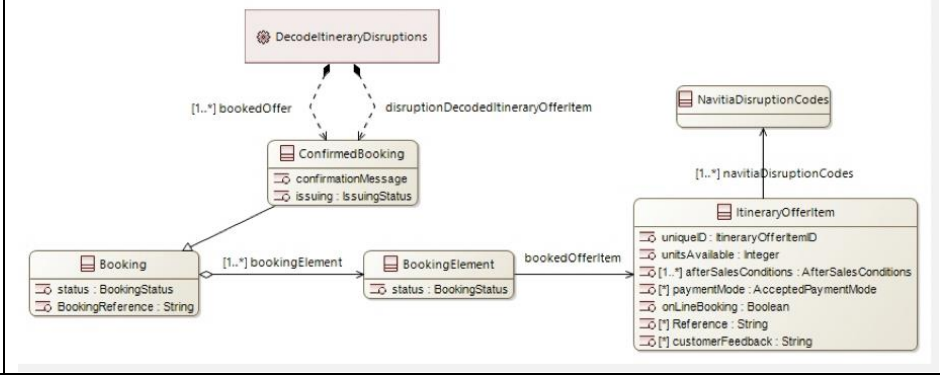
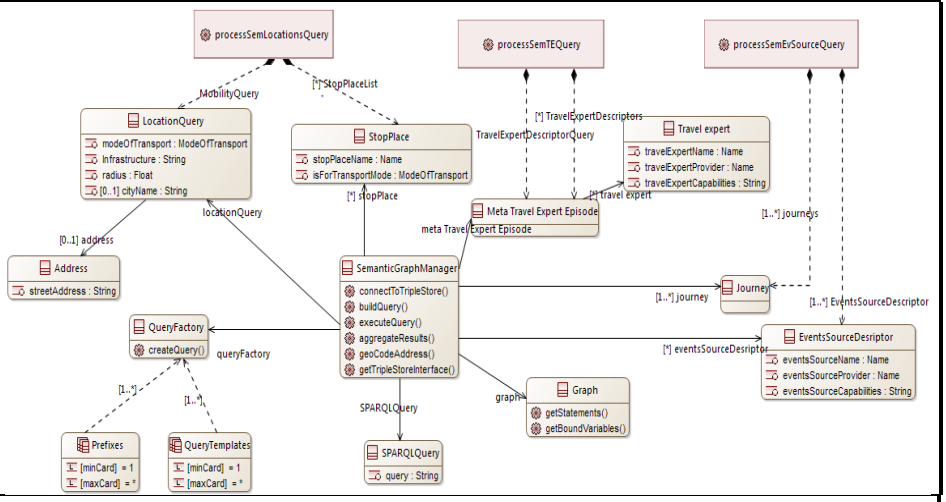
Interface ID:	**Travel Expert Resolver**
Interface Name:	
Purpose of the Interface	Associate Meta Travel Expert Episode with corresponding Travel Expert service descriptors
Requestor:	Travel Shopping (WP2 Components)
Provider	Travel Expert Resolver
Description:	Associate Meta Travel Expert Episode with corresponding Travel Expert service descriptors
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Valid Meta Travel Expert Episode request
Postconditions:	Zero, one or more Travel Expert service descriptors are associated with input Meta Travel Expert Episode
Exchange Items	
Exceptions:	Invalid request
Notes and Issues:	

Table 14: Navitia Decoder interface

Interface ID:	**Navitia Decoder**
Interface Name:	
Purpose of the Interface	Return Decode Confirmed Booking ItineraryOfferItem Stop Place and Transportation Vehicle data elements in the Navitia coding convention
Requestor:	Events Listening (WP4 component)
Provider	Disruption Navitia Decoder
Description:	Provide codes in the Navitia convention for elements of the itineraries of a Confirmed Booking
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Valid Confirmed Booking
Postconditions:	Confirmed Booking itinerary associated with Navitia codes
Exchange Items	
Exceptions:	Invalid Confirmed Booking
Notes and Issues:	

7.2.2 Internal Interface

Table 15: Semantic Graph Manager interface

Interface ID:	**Semantic Graph Manager**
Interface Name:	SemanticGraphManagerInterface processSemTEQuery(IN TravelExpertDescriptorQuery:Meta Travel Expert Episode) : returns TravelExpertDescriptors:Meta Travel Expert Episode processSemLocationsQuery(IN MobilityQuery:LocationQuery) : returns StopPlaceList:StopPlace processSemEvSourceQuery(IN journeys:Journey[1..*]) : returns EventsSourceDescriptor:EventsSourceDesriptor
Purpose of the Interface	Provide abstraction of semantic web of transportation data to IF Resolvers
Requestor:	Locations Resolver, Travel Expert Resolver, Events Source Resolver, Network Graph Manager
Provider	Semantic Graph Manager
Description:	Executes semantic query against triple store(s), assembles retrieved triples into requested output
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Acquire TripleStoreInterface
Postconditions:	Semantic query successfully executed
Exchange Items	
Exceptions:	Unavailable TripleStoreInterface
Notes and Issues:	Specific TripleStoreInterface implementation injected at startup or runtime

7.3 IF SEMANTIC BROKER

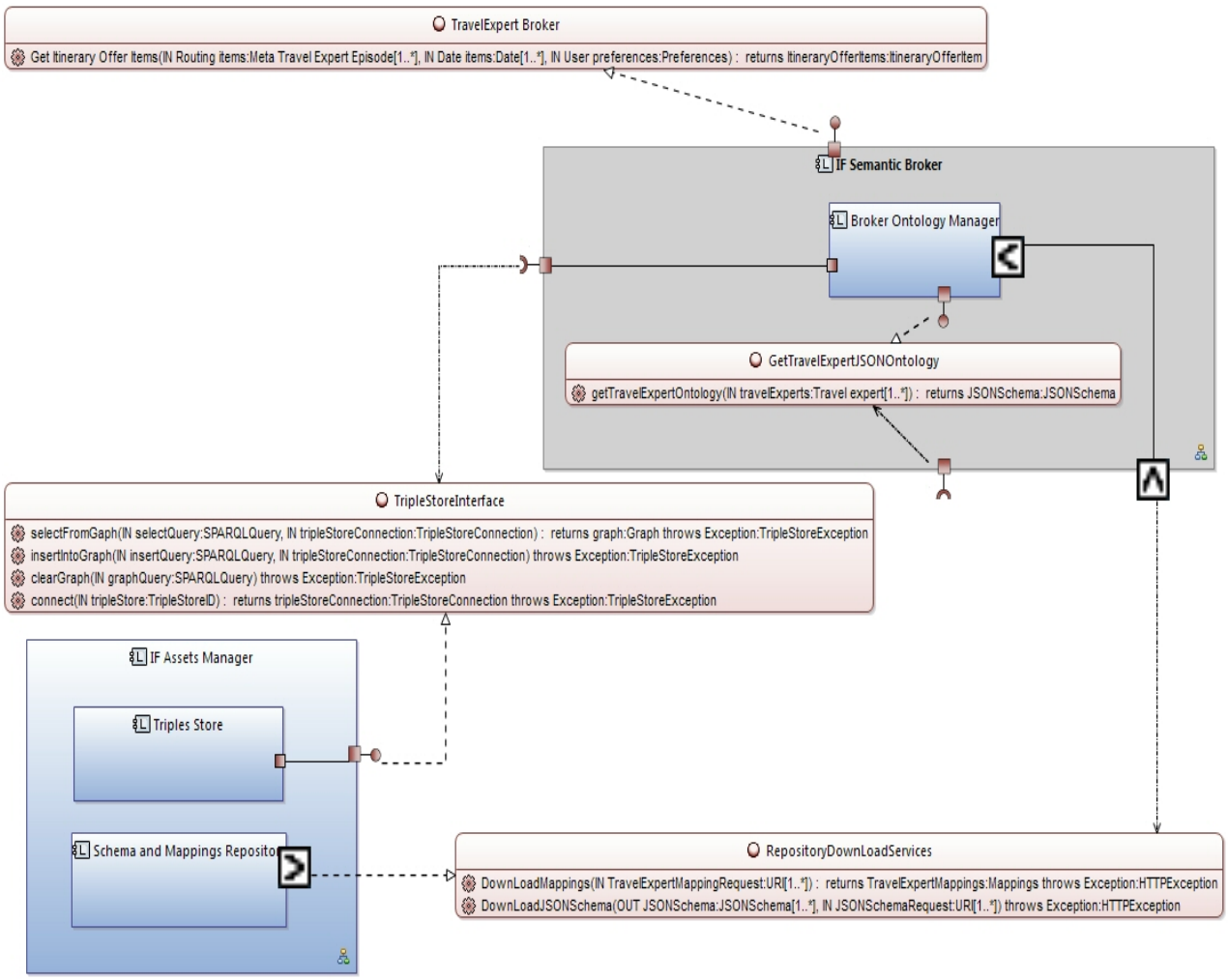


Figure 27: IF Travel Expert Semantic Broker interfaces

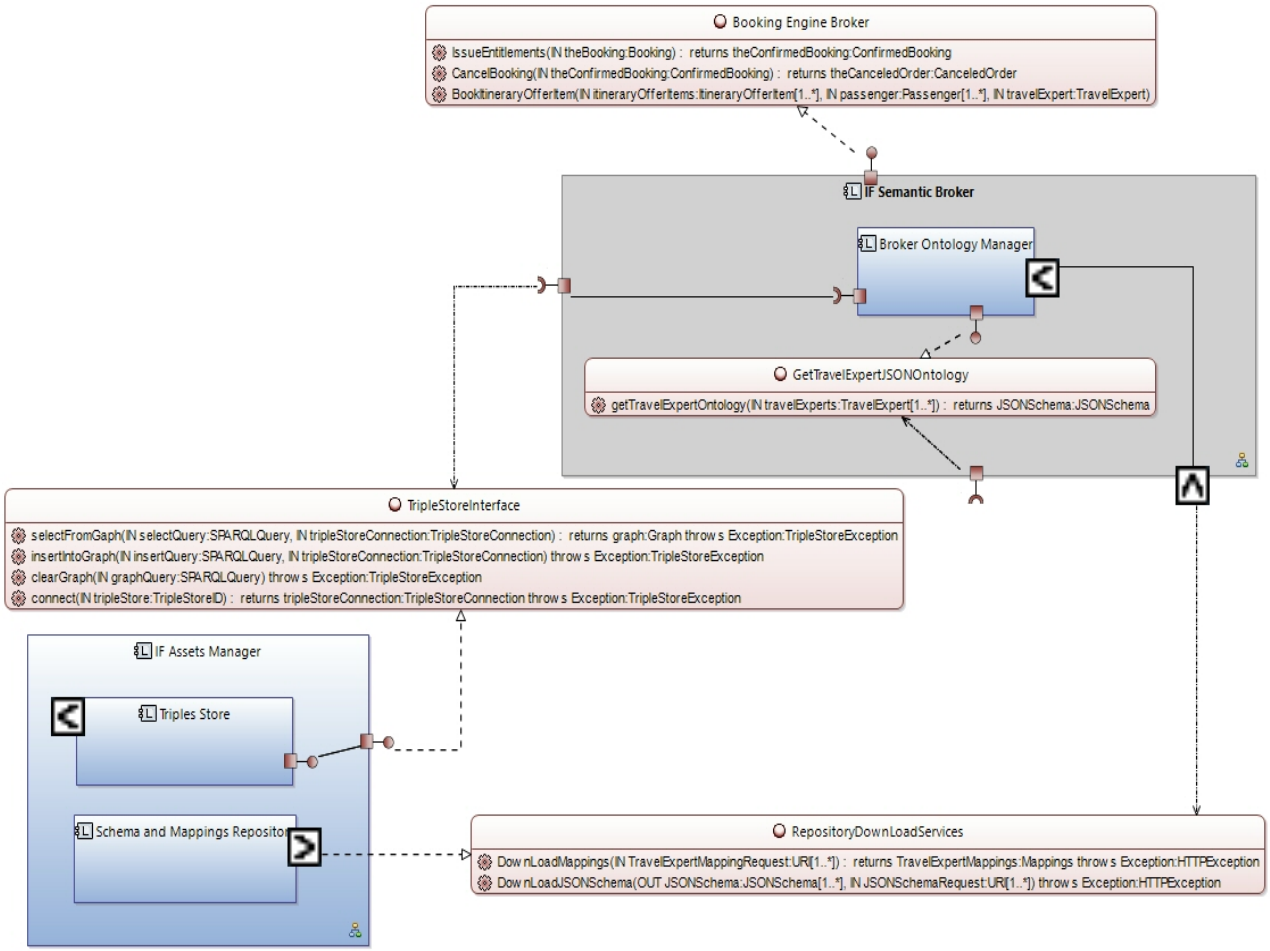


Figure 28: IF Booking Engine Semantic Broker interfaces

7.3.1 External Interfaces

Table 16: Travel Expert Broker interface

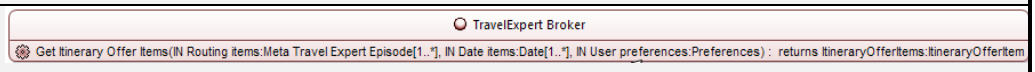
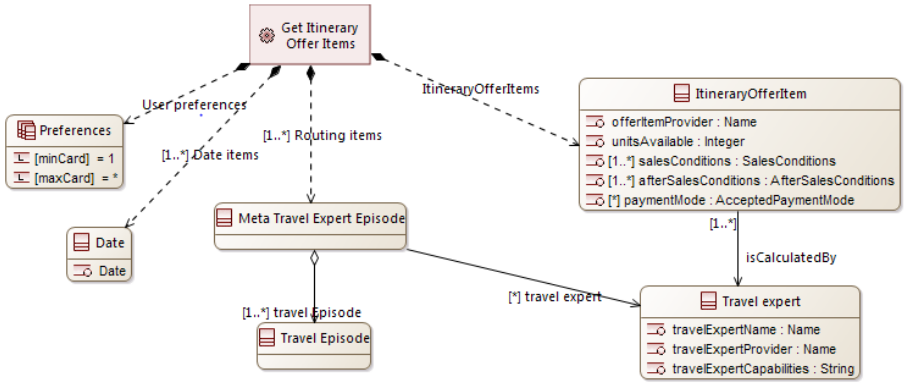
Interface ID:	**Travel Expert Broker**
Interface Name:	
Purpose of the Interface	Provide a proxy to external, heterogeneous Travel Experts
Requestor:	Offer Builder (WP2 Component)
Provider	Semantic Broker
Description:	Return ItineraryOfferItems for Meta Travel Expert Episodes from external, heterogeneous Travel Experts
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Travel Expert ontology and mappings loaded in Broker Ontology Manager
Postconditions:	Zero, one or more ItineraryOfferItem returned
Exchange Items	
Exceptions:	Missing mappings or schemas, unreachable or unavailable external Travel Experts
Notes and Issues:	

Table 17: Booking Engine Broker interface

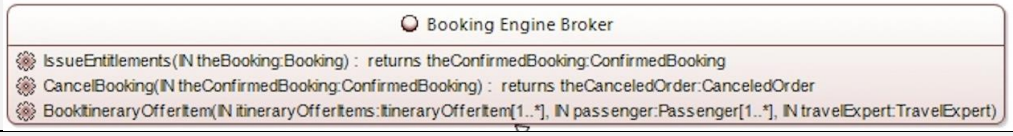
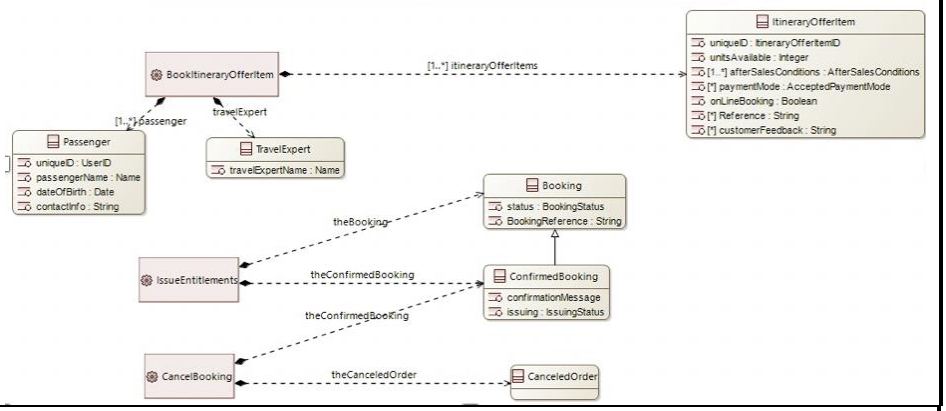
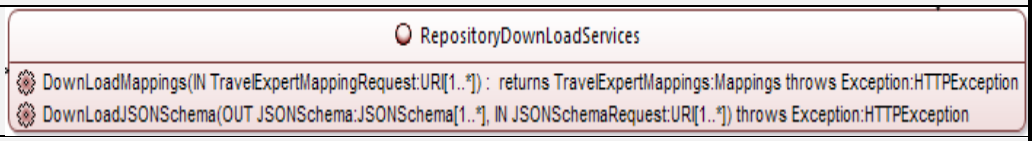
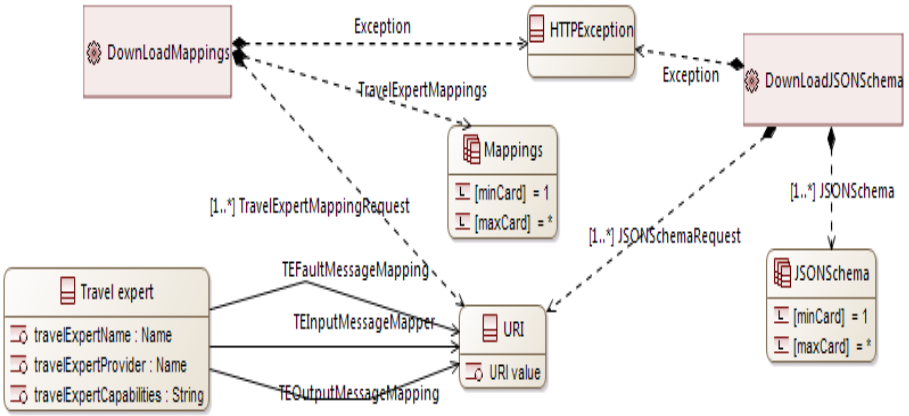
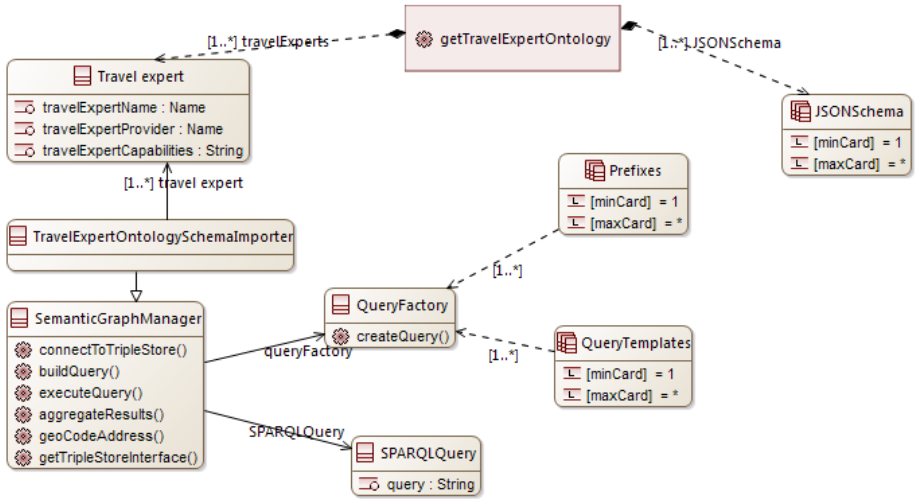
Interface ID:	**Booking Engine Broker**
Interface Name:	
Purpose of the Interface	Provide a proxy to external, heterogeneous Booking Engines
Requestor:	Booking and Ticketing (WP3 component
Provider	Semantic Broker
Description:	Return Booking (inventory lock), Confirmed Booking with Entitlements or Booking Cancelation
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Travel Expert ontology and mappings loaded in Broker Ontology Manager
Postconditions:	Zero, one or more Booking, Confirmed Booking or Cancelled Booking returned
Exchange Items	
Exceptions:	Missing mappings or schemas, unreachable or unavailable external Travel Experts
Notes and Issues:	

Table 18: Repository Download Services interface

Interface ID:	**Repository Download Services**
Interface Name:	
Purpose of the Interface	Download Travel Expert schemas and mappings from Asset Manager to Broker Ontology Manager
Requestor:	Broker Ontology Manager
Provider	Schema and Mappings repository
Description:	Download Travel Expert schemas and mappings from Asset Manager to Broker Ontology Manager
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Schema and mappings web services running
Postconditions:	Schemas and mappings successfully downloaded and installed in Broker Ontology Manager
Exchange Items	
Exceptions:	HTTP exception, unreachable or unavailable schemas and mappings web server
Notes and Issues:	

7.3.2 Internal Interfaces

Table 19: Travel Expert JSON Ontology interface

Interface ID:	**Travel Expert JSON Ontology**
Interface Name:	GetTravelExpertJSONOntology getTravelExpertOntology(IN travelExperts:Travel expert[1..*]) : returns JSONSchema:JSONSchema
Purpose of the Interface	Retrieve a Travel Expert JSON Schema to be used in processing a Travel Expert Broker request
Requestor:	Semantic Broker
Provider	Broker Ontology Manager
Description:	Retrieve a Travel Expert JSON Schema to be used in processing a Travel Expert Broker request
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Valid request
Postconditions:	Zero or one JSON schema retrieved
Exchange Items	
Exceptions:	Invalid request
Notes and Issues:	This interface available in SOFIA implementation of the semantic broker

8. TECHNICAL ARCHITECTURE

The design of the Interoperability Framework is based on the provision of a set of semantic interoperability services that can be deployed in multiple configurations, described below, and that do not mandate a specific set of communication protocols or frameworks, leaving the choice of deployment strategies to partners that may opt to re-use a shared enterprise service bus, perhaps on a virtual private network protected by specific security and authentication protocols, or decide to engage in pure peer-to-peer exchanges over the public world wide web, or a mixture of these or other options. This is also important to allow operators, including yet unknown companies who are not partners in the IT2Rail project, to choose their own roadmap for adoption of the 'native' IT2Rail semantic language for their exchanges, using or discontinuing the transformation services according to their own timeline.
The demonstration scenario of IT2Rail will be based on a concrete Use Case deployed on European-wide multi-modal transportation 'corridor' using actual data and services made available by partners in IT2Rail. This will entail a specific concrete deployment strategy for the deployment of the Interoperability Framework depending on the run-time environment of participant companies, thus validating the actual latitude of possible deployment options.

8.1 DEPLOYMENT SCENARIOS

The Interoperability Framework is designed to provide a set of idempotent stateless services using assets, i.e. ontology, web service descriptors, mapping and data triples managed in shareable repository for maximum latitude in the implementation of exchange scenarios. This section describes available deployment and exchange scenarios between a service consumer and a service provider, multiple of which may occur simultaneously depending on choices made by the participants.
The following symbols provide a legend for the diagrams illustrating the options:

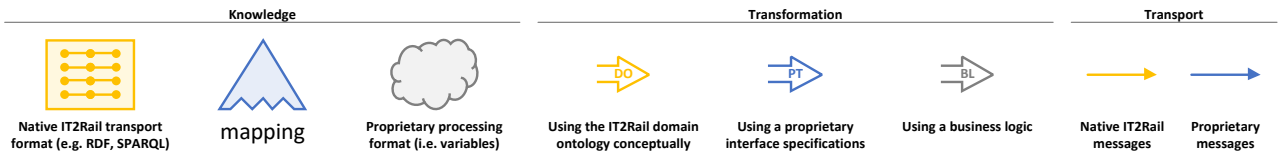


Figure 29: Decoding of symbols used in Fig 30, Fig 31 and Fig 32

In all diagrams the following is assumed:

1. The web service provider's web service descriptor has been obtained using the Travel Expert, Events Listener or Analytics resolvers as described in the preceding sections
2. Mapping is obtained through download from the shared Mappings Library

The diagrams illustrate cases in which the Service Requestor is expressing requests in the IT2Rail ontology language. Mirror cases in which the roles are reversed, i.e. the Service

Provider is a native IT2Rail ‘speaker’ are also possible with the same benefits and issues, but they are not shown for clarity.

8.2 NATIVE IT2RAIL

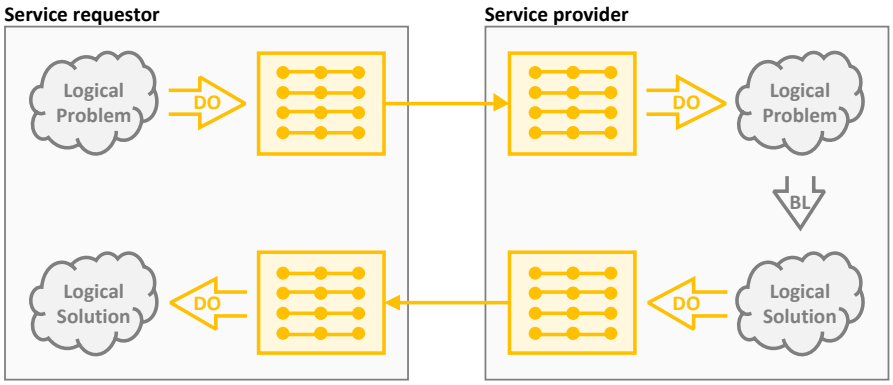


Figure 30: Native IT2Rail exchange scenario

This Scenario depicts the peer to peer communication between an arbitrary service requestor and a service defined and developed in IT²Rail. All participants are capable of using IT²Rail semantic technologies for communication. This is the desired exchange scenario for any newly IT2Rail developed services.
Benefits involve: Lightweight approach in terms of workflow. All data/knowledge is encoded in the same language. Least error prone since no mapping is required.

Issues involve: Only suitable for newly developed services.

8.3 WRAPPED LEGACY SERVICE

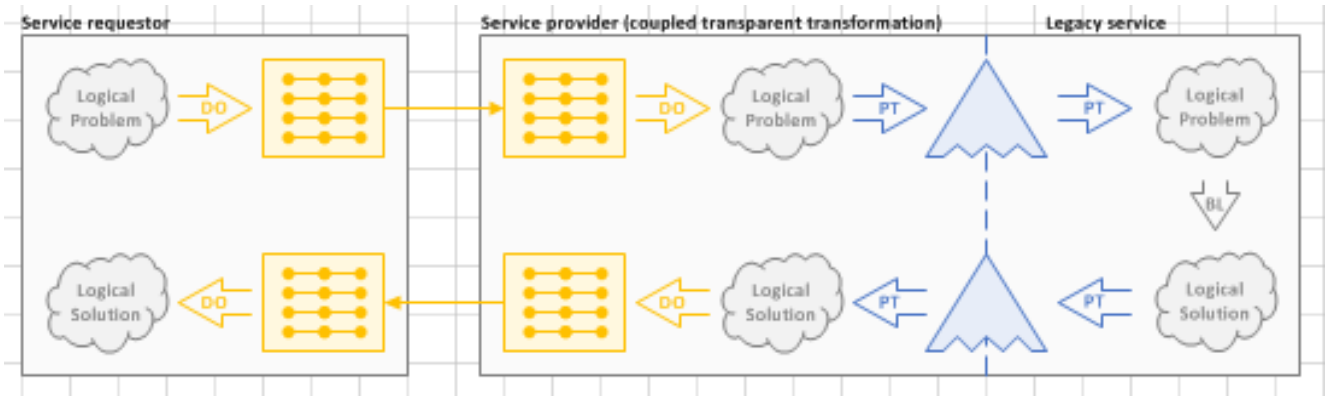


Figure 31: Wrapped legacy service exchange scenario

This is the preferred standard scenario for integration of legacy services into IT2Rail. The transformation from native IT2Rail semantic technologies to legacy data representation shall be conducted by the entity providing the legacy service. From IT²Rail perspective the legacy service can be considered as completely wrapped.
Benefits involve: Legacy services can be connected to IT2Rail. Mappings provided and performed by legacy service provider. Legacy data schema is known, hence transformation is less error prone. The workflow is transparent for the service requestor.

Issues involve: Overall Workflow is slightly more complex compared to Native IT2Rail. Changes to Legacy Service may require changes to mapping. However changes of the Legacy service are expected to consist in adopting the native IT2Rail ontology language, thus reducing the case to the native IT2Rail scenario.

8.4 BROKER – ENTERPRISE SERVICE BUS

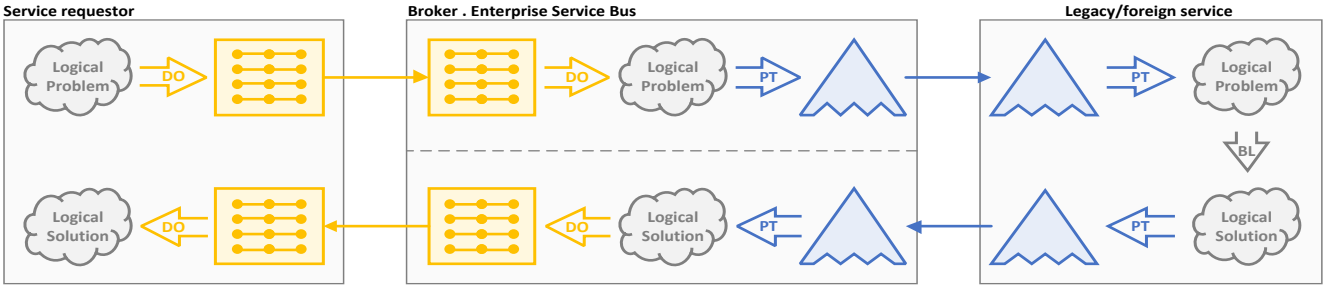


Figure 32: Enterprise Service Bus / Broker scenario

In this scenario exchanges are mediated through a broker or standard Enterprise Service Bus in which assets obtained from the WP1 Assets Manager, i.e., the ontology and the mappings are deployed.
Unlike the previous scenarios, in this case the exchange between Service Requestor and Provider is not peer-to-peer. Secure and reliable communications must therefore be provided by the selected Enterprise Service Bus platform.

8.5 OPERATIONAL ENVIRONMENT

The Interoperability Framework has been designed to provide a set of enriched services to all modules developed by the other work packages of the IT2Rail environment. The Interoperability Framework provides cost-effective technical means that allow participant devices, systems and applications to interoperate both syntactically and semantic modes.
The inherent nature of the Interoperability Framework is based on modularity, to this end, the provision of semantic-annotated services is guaranteed independently from the architecture and deployment strategy.
The Interoperability Framework doesn't mandate a specific hardware and software configuration, it can be deployed both within traditional virtual machines and innovative software container solutions as it follows the principles of decomposition of the micro-services approach. For the IT2RAIL project purposes, a deploy strategy based on virtual machine has been chosen, CentOS 7 64bit was installed along with Java Development Kit 1.8.
The virtual machine which ships the Interoperability Framework has the following features:

- Memory - Ram 8GB.
- Cpu 2 core.
- Hdd 25 Gb provisioning to 130.

In addition, most of the software modules belonging to the Interoperability Framework are supposed to run within an application server. Tomcat has been chosen as application server where to deploy the Interoperability Framework modules. Tomcat is an open source implementation of several Java EE specifications including Java Servlet, JavaServer Pages (JSP), Java EL, and WebSocket, and provides a "pure Java" HTTP web server environment in which Java code can run. Tomcat offers the most basic functionality necessary to run a server, meaning it provides relatively quick load and redeploy times compared to many of its peers.

9. TECHNICAL DEMONSTRATOR FREL IMPLEMENTATION

This chapter documents the main implementation objects of the “final release” (FREL) features of the design as a technical demonstrator of the Interoperability Framework concept.
Multiple implementation choices are possible: the fundamental selection criterion for the technical demonstrator in the IT2Rail project is the minimisation of development and testing effort on *non*-research elements of the solution while concentrating the attention on its innovation contents. For this reason open source frameworks and tools have been employed whenever possible, as documented in deliverable D 1.5 Interoperability Framework Integrated Development Environment, and only basic considerations have been given to non-functional specifications such as performance, scalability or security, mainly to validate that such quality aspects are *not* constrained by the design while they can be addressed by a specific design of the underlying computing infrastructure. The design of this infrastructure is also an implementation choice.

9.1 TECHNICAL DEMONSTRATOR PACKAGING FOR DELIVERY

In order to allow for the multiple deployment options described in the previous section Deployment Scenarios, and to facilitate the further development of the Interoperability Framework, the Interoperability Framework has been developed as a set of interdependent eclipse MAVEN projects[4], each encapsulating specialised functionality. The following figure shows the projects and their dependencies:

[4] Apache Maven (https://maven.apache.org/) is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

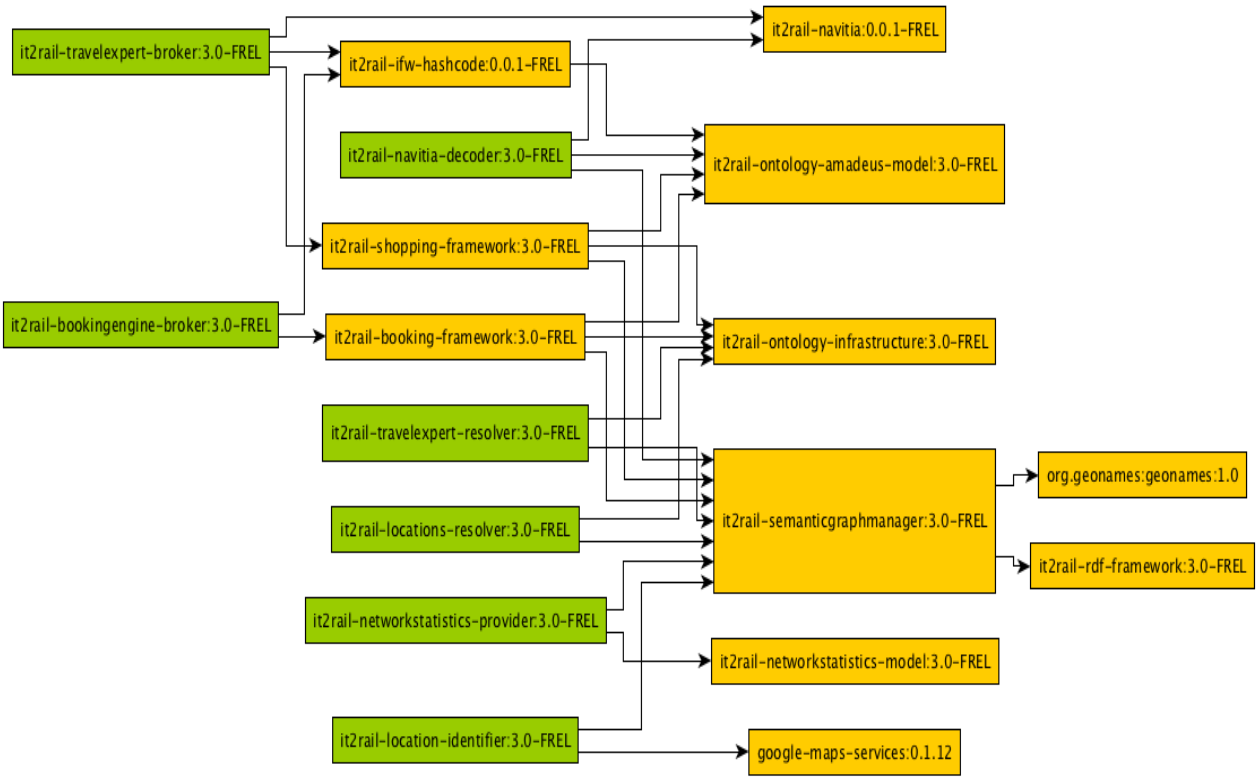


Figure 33: Interoperability Framework packaging

Items in green are Web Application Resource (WAR) files designed to be deployed in a standard web application server such as Apache Tomcat 8.5. They implement the web service “external” interfaces used by all other components of the IT2Rail project. These are the Interoperability Framework “packaged resolvers” implementations.

Items in orange are Java Archive (JAR) files containing java code and other resources, such as configuration files, that implement “internal” interfaces and handle lower level tasks for the packaged resolvers.

WAR files typically include the dependent JARs and are therefore self-contained: this allows them to be deployed simultaneously on multiple web application servers for horizontal scalability.

JARs are libraries that can be used as dependencies in alternate implementations of the packaged resolvers, for example in a “wrapped legacy service” deployment scenario, whereby the JARs are installed in the class path of a legacy shopping or booking application at the client or at the server side of an exchange thus avoiding the need to deploy the brokers on a remote web application server.

1. **it2rail-navitia:0.0.1-FREL** is a library that implements access to the Navitia platform on https://www.navitia.io/.
2. **it2rail-ifw-hashcode:0.0.1-FREL** is a library that implements a hashing function used to electronically sign valid offeritems, and recognises the signature when they are submitted to booking.

3. **it2rail-ontology-amadeus-model:3.0-FREL** is a library of java classes that implement JAXB bindings for XML schemas provided by the Amadeus company for interfacing with Travel Shopping and Booking/Ticketing applications. The classes are annotated with terms from the It2rail ontology.
4. **it2rail-shopping-framework:3.0-FREL** is a library that implements helper classes used by the Travel Expert Broker to validate requests and handle exceptions.
5. **it2rail-booking-framework:3.0-FREL** is a library that implements helper classes used by the Booking Engine Broker to validate requests and handle exceptions.
6. **it2rail-ontology-infrastructure:3.0-FREL** is a library with semantically annotated classes used by the shopping and booking frameworks, and by the Locations and Travel Expert Resolvers to convert the locations semantic graph representation to XML.
7. **it2rail-semanticgraphmanager:3.0-FREL** is a library that encapsulates semantic interoperability functionality on behalf of all packaged resolvers. It handles the interfacing with various triple stores distributed across the world wide web.
8. **it2rail-rdf-framework:3.0-FREL** is a library that implements serialisation of java classes to the RDF format and vice-versa for low level access to triple stores.
9. **org.geonames:geonames:1.0** is a library that implements access to the geonames triple store and georeferencing services.
10. **it2rail-networkstatistics-model:3.0-FREL** is a library that implements JAXB bindings for the network statistics XML schema.
11. **google-maps-services:0.1.12** is an externally provided library that implements access to google maps services. It is used by the Location Identification service.

9.2 THE IT2RAIL RDF FRAMEWORK

The IT2Rail RDF framework provides Java Persistence API Architecture (JPA)-like capabilities extending it with the ability to operate on triple stores, i.e. on database systems designed for storage and retrieval of RDF statements (“triples”) through semantic queries.
Similarly with how in JPA annotations on java classes provide automatic object-relational mappings from these classes to relational database schemas and vice-versa, the RDF framework provides automatic mappings to/from RDF statements. This enables the developer to delegate the mechanics of connection, input/output and storage/retrieval of the data across the network to the framework, while concentrating on the manipulation in pure java of objects and relationships which, unlike the JPA entities, represent logical statements connected to the domain’s axiomatic description of the domain’s ontology. For example, triples generated by a SPARQL query as a result of logical inference, i.e. new logical statements inferred based on the axioms and the existing data, are automatically instantiated as java objects and relationships and are therefore available for “ordinary” programming.
The IT2Rail RDF framework is itself an extension and a merging of the two open source frameworks Empire (https://github.com/mhgrove/Empire) and Pinto (https://github.com/stardog-union/pinto), both licensed under the Apache License 2.0. The extensions consist in:

1. Porting to the Eclipse RDF4J framework (http://rdf4j.org/).
2. Merging of Empire and Pinto RDF mapping functionality.
3. It2Rail-specific extensions, including multiple additional datatype conversions

The it2rail RDF framework for the FREL release is built and installed in the IT2Rail project's MAVEN repository for use in higher level development.

9.2.1 Empire Configuration

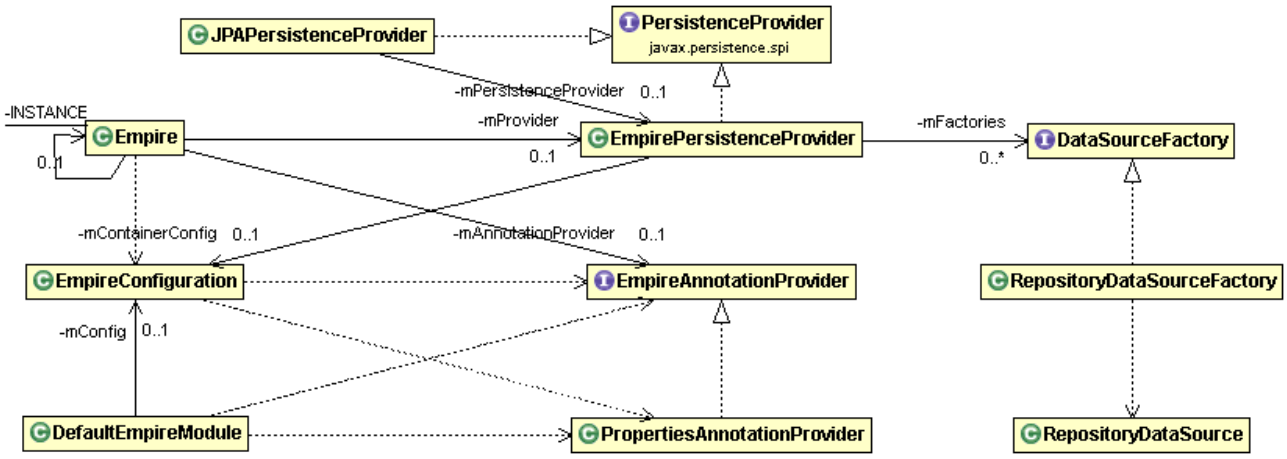


Figure 34: The Empire Configuration

The Empire class is a container that holds the RDF framework's configuration instantiated at start up time. The configuration consists principally of three items:

1. The EmpireConfiguration containing a list of persistence unit descriptors and the name of a file of annotated java classes.
2. The Namespace and NamedQueries annotations used by the RdfGenerator and RdfQueryFactories classes, described below, to perform SPARQL queries and automatic mapping to/from RDF. This configuration is generated at initialisation time by an implementation of the EmpireAnnotationProvider interface.
3. The DataSourceFactory to be used by the EntityManager, described below, to create access to one or more triple store providers for storing and retrieving triples. This configuration is generated at initialisation time by an implementation of the JPA PersistenceProvider interface.
4. Concrete implementations of the EmpireAnnotationProvider and EmpirePersistenceProviders are injected at initialisation time through the Guice framework5, i.e. by defining bindings in a Guice Module. The figure above displays the providers injected by default through the DefaultEmpireModule (not shown in the figure): EmpirePersistenceProvider implements the JPA PersistenceProvider while PropertiesAnnotationProvider implements EmpireAnnotationProvider.

Default Empire Configuration

The following default configuration is created at initialisation time by the call
Empire.init(new OpenRdfModule)

[5] https://github.com/google/guice/wiki/ExternalDocumentation

under the control of Guice modules:

1. Empire Configuration persistence unit descriptors are read from the configuration file whose name is associated with the System Property “empire.configuration.file”. The configuration file must be accessible by the classLoader.
2. Namespace and NamedQueries are read through EmpireAnnotationProvider from the annotations of java classes listed in a file whose name is the value of “annotation.index” in the configuration file. The file listing classes and named queries must be accessible by the classLoader.
3. The EmpirePersistenceProvider implementation of the JPA PersistenceProvider is injected.
4. The concrete implementation RepositoryDataSourceFactory of the DataSourceFactory interface is injected from the binding specified in the Guice module OpenRdfModule

The following is a sample empire configuration file with two persistence unit descriptors:

```
annotation.index = locations.resolver.empire.annotation.config


0.name=it2rail
0.factory= rdf4j
0.url =http://192.168.150.139:7200
0.repo=IT2RAIL

1.name=wikidata
1.factory = rdf4j
1.sparql_endpoint = https://query.wikidata.org/
```

The property annotation.index points to the file name “locations.resolver.empire.annotation.config” which has the following contents:

```
RdfsClass= org.it2rail.ontology.infrastructure.StopPlace, <additional classes>
```

9.2.2 Entity Manager

Once an Empire configuration is set up the NameSpace and NamedQueries annotations, the persistence unit descriptors and a concrete DataSourceFactory are available to create instantiate one or more Entity Managers used to perform storage and retrieval of RDF statements and conversions of java classes to/from RDF triples.
The IT2Rail RDF framework’s Entity Manager is an implementation of the JPA EntityManager interface, i.e. it provides the same methods to persist, merge, remove and find java entities to/from triple stores as those provided for relational data bases.

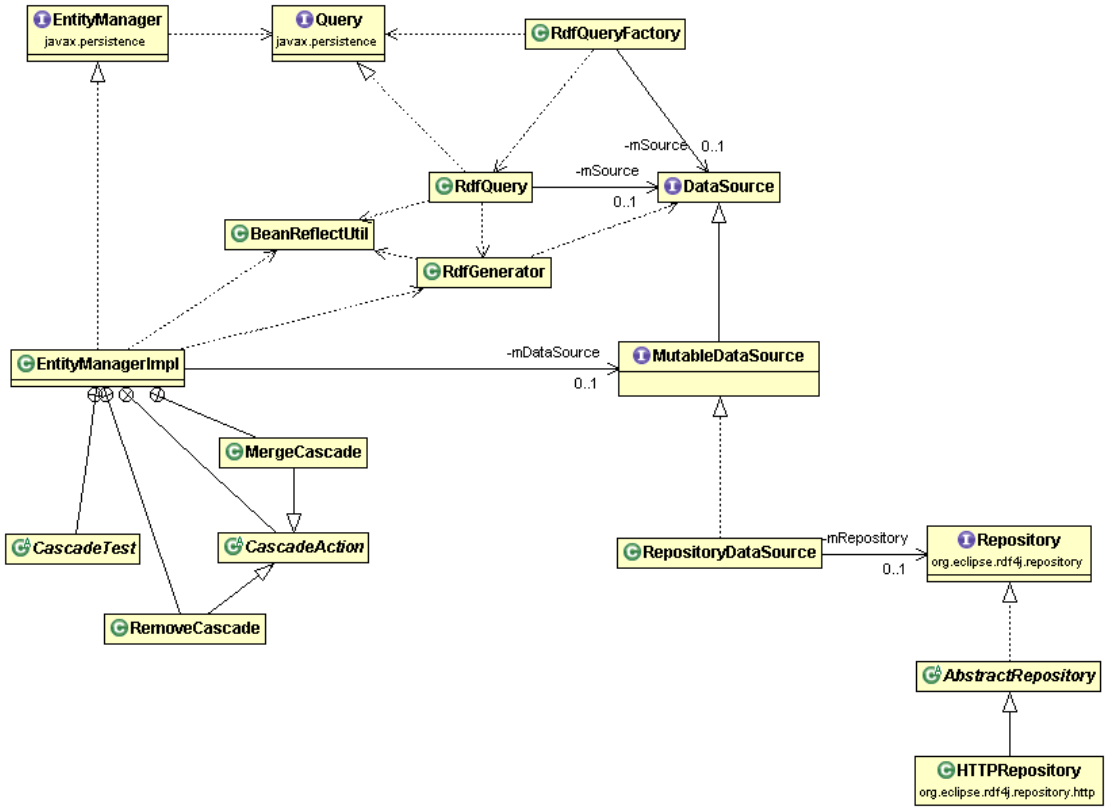


Figure 35: Entity Manager

In the figure above the EntityManager, on the left, handles the usual JPA operations, including managing cascading as determined by normal JPA CascadeType values. Objects are persisted/retrieved from the concrete DataSource implementation created by the RepositoryDataSourceFactory injected at Empire configuration time. In the figure, to the right, the DataSource interface implementation created by the RepositoryDataSourceFactory is the class RepositoryDataSource, which itself has a reference to a HTTPRepository of the Eclipse RDF4J framework to access a remote triple store. All persistence operated commanded by the EntityManager are executed as method invocations on the actual interface to the remote triple store.
Between the Entity Manager and the Repository the RDFGenerator performs the serialisation of java objects to/from RDF triples.
In analogy with the JPA API Architecture, which provides capabilities to generate custom SQL queries from java classes, the IT2rail RDF framework includes the RdfQueryFactory utility class to generate and validate SPARQL queries against the target triple store concrete Repository interface implementation.
An Entity Manager is instantiated through the call

```
EntityManager aManager = Persistence.createEntityManagerFactory(persistenceUnit)
                                                    .createEntityManager();
```

Where persistenceUnit is the name of a persistence unit descriptor created at Empire configuration time, i.e. “it2rail” or “wikidata” in the example configuration file shown above.

9.2.3 RDF Generator

The RDFGenerator provides utility classes and methods to execute serialisation of java classes to/from RDF triples using annotations on the class.

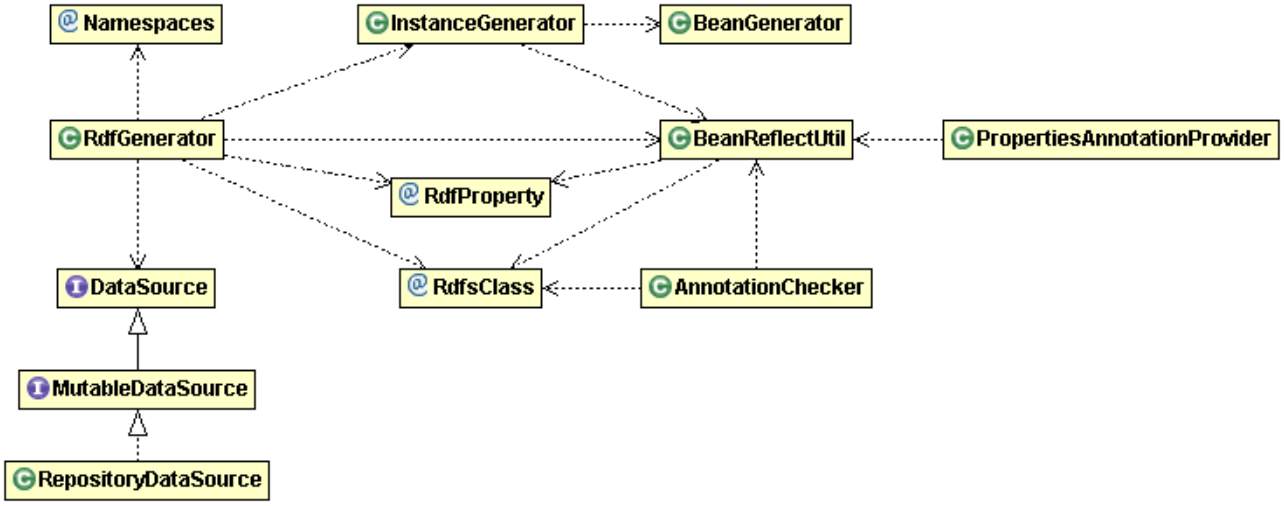


Figure 36: RDF Generator

The fundamental mechanism of the serialisation is the following:

1. Serialisation **to** RDF ("lifting" from the specific data format to the 'ontology' language)
    a. The class @RdfsClass annotation determines the rdf:type property of an instance of this class.
    b. All getter methods of a class that contain an @RdfProperty annotation are used to read property values using reflection. The value of the @RdfProperty annotation becomes a predicate, and the property value obtained from the getter becomes the object of the triple.
2. Serialisation **from** RDF ("lowering" from the 'ontology' language to the specific data format)
    a. The object of the rdf:type predicate in a triple determines the instantiated java class.
    b. For all predicates in the triple the setter methods in the class that have a matching @RdfProperty annotation are invoked to set the class' corresponding value to the object of the triple's predicate. The setter methods can be 'inferred' from the annotated getter methods if they conform to the usual naming convention for java getters/setters.

9.3 The Semantic Graph Manager

The Semantic Graph Manager is built on top of the It2Rail RDF framework and provides an implementation of the IT2Rail Semantic Graph Manager component described in chapter 6 of this document to support multiple "packaged resolver" services.
The Semantic Graph Manager for the FREL release is built and installed in the IT2Rail project's MAVEN repository for use in packaged resolver development.

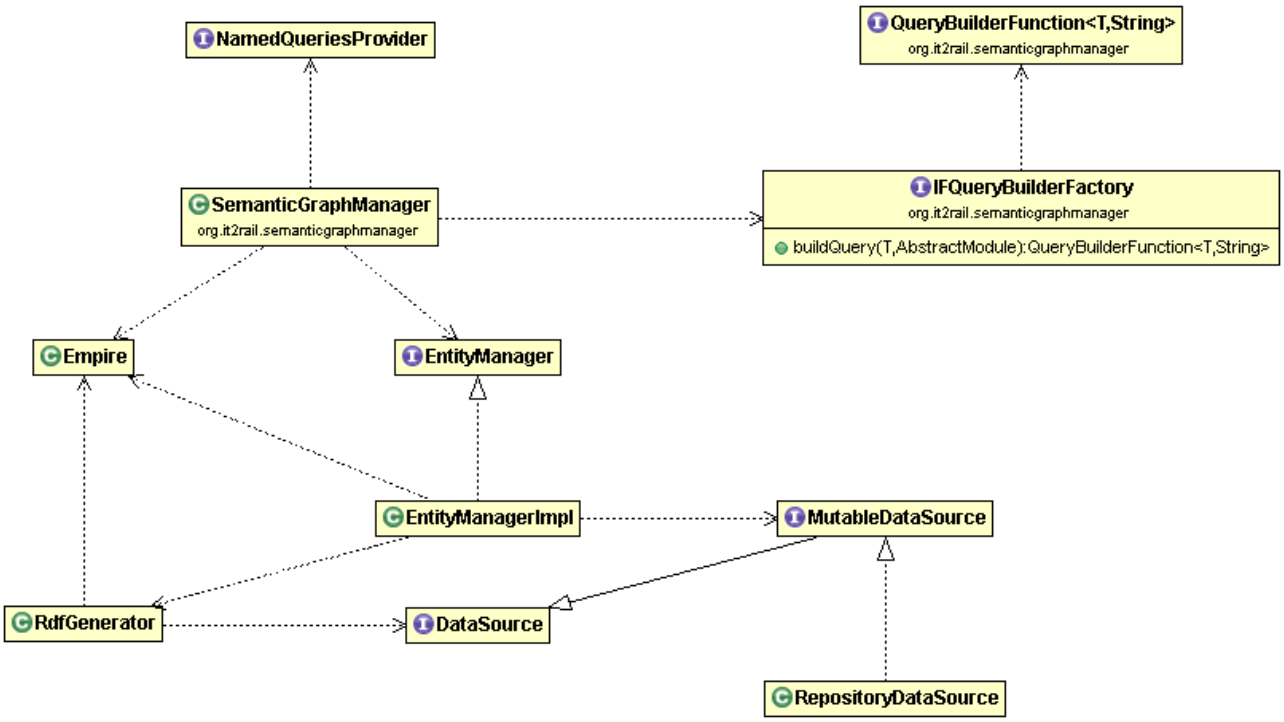


Figure 37 Semantic Graph Manager

As can be seen in the figure above, the Semantic Graph Manager adds to the underlying it2rail RDF framework references to an instantiated EntityManager, to a NamedQueriesProvider interface and to an IFQueryBuildFactory interface.
A particular “packaged resolver” Semantic Graph Manager is obtained by supplying a specific Guice injection module containing the specific bindings requested

1. to the desired persistence unit configuration for which an EntityManager must be created.
2. to the desired DataSourceFactory for the specific packaged resolver.
3. to the desired NamedQueriesProvider interface implementation for the particular resolver.
4. to the desired IFQueryBuilderFactory implementation requested for the particular packaged resolver.

The first two bindings are propagated to the underlying it2rail RDF framework to create the Empire configuration described in chapter 9.1.1, while the third and fourth are specific to the Semantic Graph Manager:

1. a NamedQueryProvider interface implementation provides access to a set of stored SPARQL queries templates from which individual queries can be instantiated. Since semantic queries can represent first order predicate logic ‘rules’, the ability to store them for re-use, versioning, etc. as ‘assets’ in the Interoperability Framework Assets Manager is an important feature of the design and the implementation. Since providers can be injected through the Guice framework it is possible use different sets/versions of named queries, for example for testing and production, and/or to store them in different media, e.g. as files packaged with the Semantic Graph Manager’s JAR archive, on distributed web servers or the triple store itself.
2. An IFQueryBuilderFactory interface implementation, also injected through the Guice framework, provides the means to produce an implementation of the Function<T,String> functional interface available in Java 1.8. The concrete injected

implementation generates such a functional interface that creates a SPARQL query from a request of generic type T. Used in conjunction with the named queries templates, this feature allows for the development of multiple "packaged resolvers" on the same Semantic Graph Manager by supplying it with the appropriate Guice Module.

As an example of the instantiation of a specific packaged resolver, the following statement creates a Location Resolver:

```
SemanticGraphManager sgm = SemanticGraphManager.getInstance(new LocationResolverModule());
```

The instruction above will result in the following instantiated resolver:

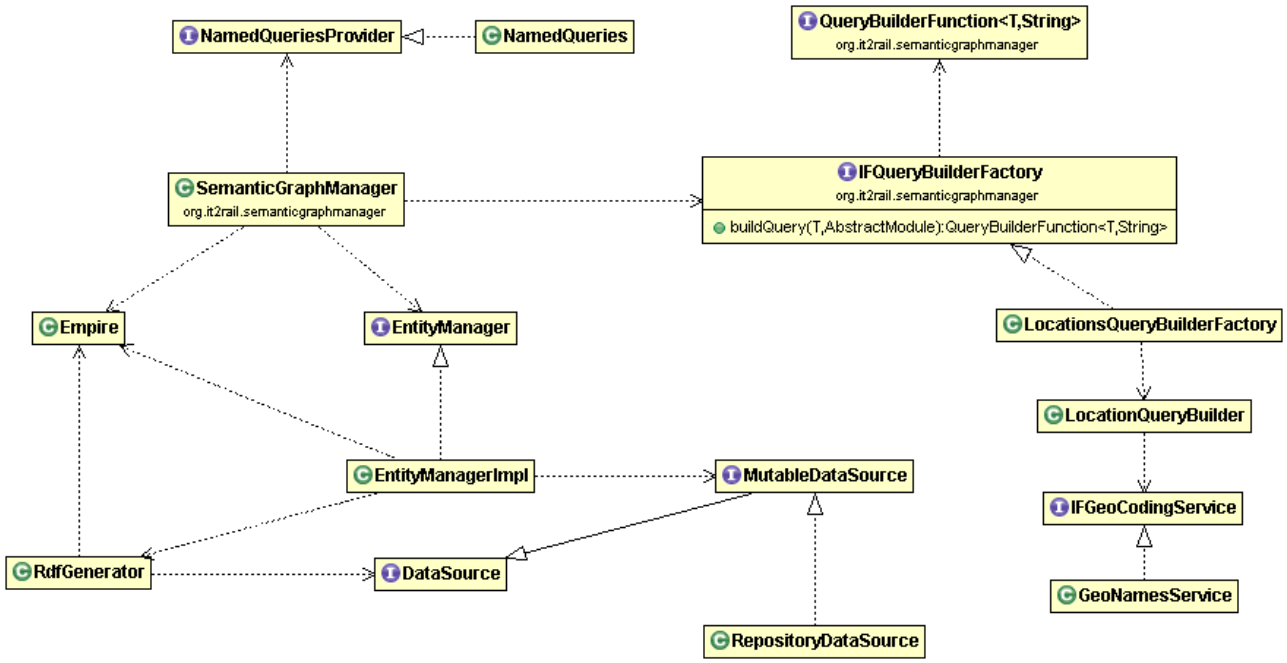


Figure 38: instantiated resolver

9.4 LOCATION RESOLVER SERVICE

The Location Resolver service is implemented as a web service that exposes an interface to the IT2Rail shopping components to generate a list of "infrastructures", e.g. Airports or Rail Stations, located within a given radius of a reference point.

It is a 'packaged resolver' in the sense that it is a *specific configuration*, i.e. "package", of the Semantic Graph Manager dedicated to a particular "resolver" job which involves, additionally, the automatic conversion of a request/response XML message according to a given schema to/from the semantically annotated data stored in the distributed triple store.

For this purpose the following has been developed for FREL:

1. The relevant subset of the IT2Rail ontology, expressed in OWL, has been converted into a set of Java classes annotated with the terms of the ontology for use by the it2rail RDF framework RDFGenerator.
2. The process for conversion of the ontology for use with the it2rail RDF framework is the following:

   a. OWL Classes (concepts) are Java classes with an RdfClass annotation with the same namespace and concept name.
   b. A data or object property in the ontology with Range a given Concept is an RdfsProperty annotated getter of the Java class that represents Concept. The type returned by the getter is the Domain type of the data or object property of the ontology.
3. The external XML request/response message schemas have been converted to Java JAXB using a standard JAXB MAVEN plugin.
4. The resulting JAXB have been annotated with the terms from the ontology through the it2rail RDF framework mechanism establishing semantic correspondence between the JAXB and the ontology Java classes.
5. Template SPARQL queries have been created and stored for use by the Semantic Graph Manager.
6. A specific implementation of the IFQueryBuilderFactory interface has been created which is, in effect, the only Location Resolver specific implementation: the rest of the functionality is provided by the Semantic Graph Manager.

While this “annotation” process, which provides the RDFGenerator with the necessary instructions to perform lifting/lowering to/from syntactic to/from semantic levels, has been performed manually, it is to be noted however that no *java methods* have been written, so that only the “ontology” language has been involved. Additionally as a result of the exercise it has been determined that a MAVEN plugin could be written to automate at least in part the process in the further development of the technology.

The following figure depicts the Location Resolver service instantiated by supplying the Semantic Graph Manager with the specific LocationResolver Guice module.

In particular the LocationResolverQueryBuilder creates the specialised SPARQL queries needed by the implementation of the service, and uses an implementation of the IFGeoCodingInterface, also injected through the Guice Module, to obtain geo-coordinates of requested points of interest to be used in identifying StopPlaces in the distributed semantic graph stored in the triple store. Injection of the geo-coding service in the LocationResolverQueryBuilder allows for choosing any one of the multiple such free or commercial services available from multiple providers.

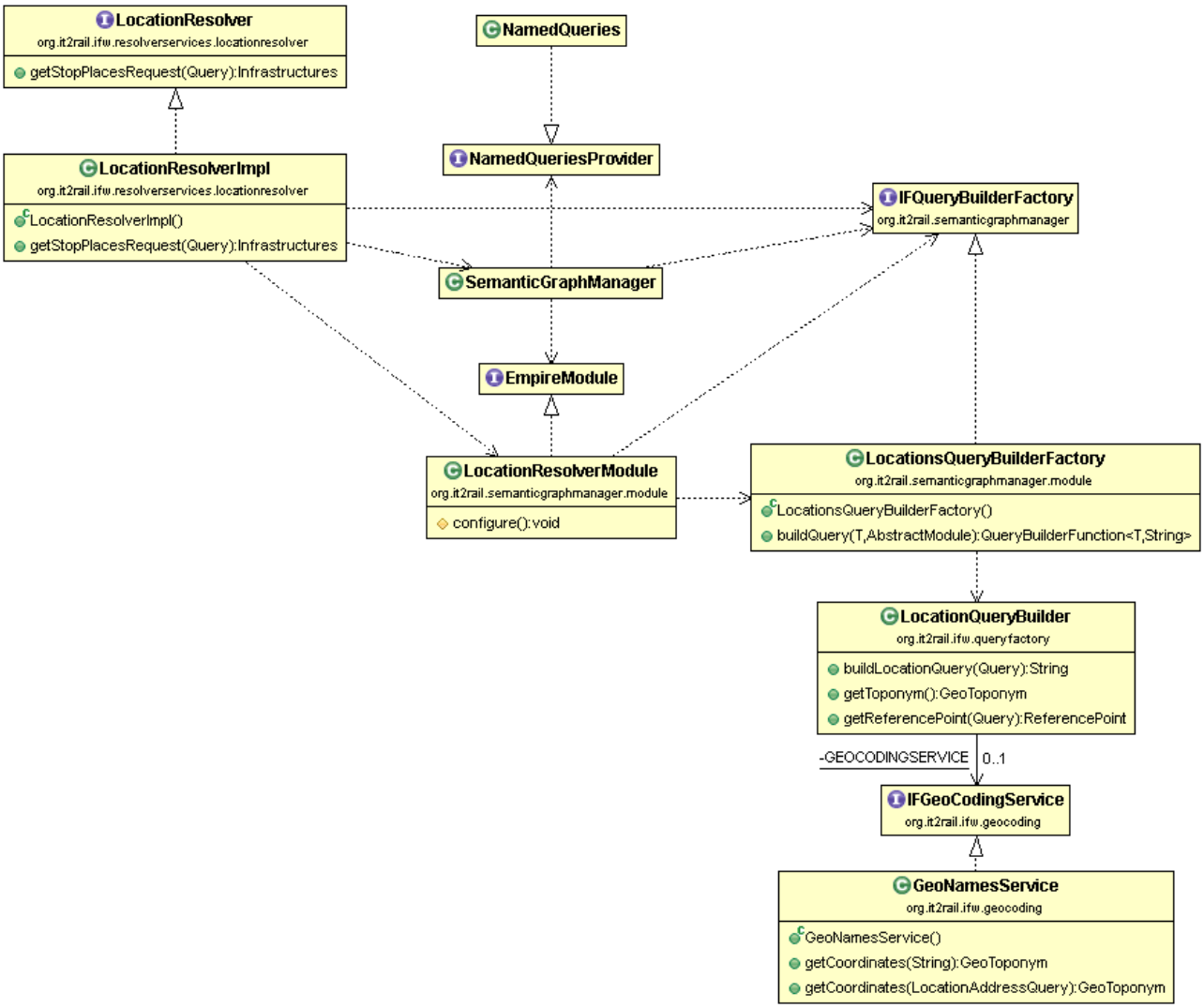


Figure 39 Location Resolver service

9.5 TRAVEL EXPERT RESOLVER SERVICE

The Travel Expert Resolver service is implemented as a web service that exposes an interface to the IT2Rail shopping components to generate a list of descriptors of “Travel Expert” web services that can generate OfferItems for a given “MetaRoute”, i.e. an ordered pair of start and stop StopPlaces.
The Travel Expert resolver is also a “packaged resolver” in the sense described above, i.e. it is a specific “packaging” of the Semantic Graph Manager to perform the defined “resolver” job.
As with the Location Resolver, the “packaging” is obtained for the Travel Expert resolver using the same process of annotations, creation of the query builder functionality and binding to Semantic Graph Manager interfaces as necessary through a specific Guice Module.
The java classes generated from the ontology are of course shared across all packaged resolvers as a JAR archive listed in the specific resolver’s MAVEN POM file.

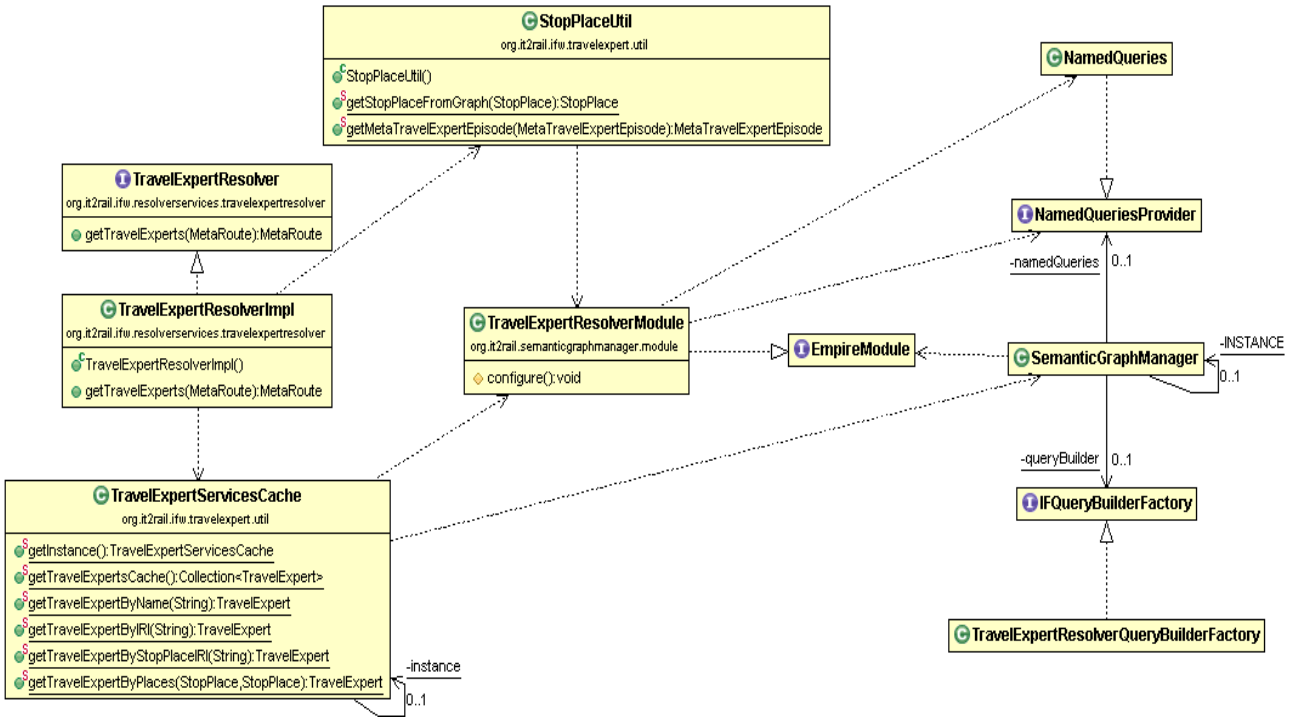


Figure 40: Travel Expert Resolver

The Travel Expert Resolver service conforms to the same principles as the Location Resolver but uses two additional utility classes, StopPlaceUtil and TravelExpertServiceCache, as helpers during the FREL implementation to emulate the presence of Travel Experts that were not available at this stage of the project.

9.6 SEMANTIC BROKER AND PROXY INJECTION

The semantic broker mediates the interfacing of Travel Shopping, Booking and Ticketing applications with a multitude of heterogeneous Travel Experts and Booking Engines. The mediation involves the semantic transformation of messages across multiple formats and protocols. This “packaged resolver” of the Interoperability Framework has been designed according to the following principles:

1. Multiple concurrent deployment options must be available to interacting Partners. In certain cases a dedicated separate “broker” component can host the functionality, but it must be possible to operate the “mediation” within the calling or the receiving application. The mediation functionality must be therefore independent from the software handling communications, security and other accessory operations.
2. If a broker is chosen for deployment of the mediation functionality then Partners must have a choice of the broker implementation, and different Partners must be able to choose different broker environments, such as a standard Enterprise Service Bus supplied by any vendor.
3. If a broker is chosen or developed, then the broker must be unaware of the mediation functionality that it is asked to execute. It must be capable, in fact, of executing multiple such mediations depending on the specifics of each different requirement.

In the IT2Rail demonstrator the choice of a dedicated broker has been made by the developers of the Travel Shopping, Booking and Ticketing applications. This broker has

been developed as the implementation of a TravelExpertBroker and a BookingEngineBroker web service deployed in an Apache Tomcat 8.5 web application server.
In order to comply with the principles listed above, the following implementation has been selected:

1. A shopping and a booking “framework” JAR files implement “accessory” functionality that is common to all mediation functions and neutral with respect to the different Travel Experts and Booking Engines. These frameworks perform validation of the incoming messages, assemble the output and handle exceptions generated by the remote Travel Experts and Booking Engines, or by the broker itself.
2. The broker itself implements the TravelExpertBroker and a BookingEngineBroker web service interfaces and call a single method to perform mediation. The single method is defined in an *abstract* TravelExpertProxy or BookingEngineProxy Java class.
3. The broker accesses a Travel Expert registry that identifies the URL of the JAR file that contains the specific TravelExpertProxy, or BookingEngineProxy, *concrete* implemention of the abstract class for a given target Travel Expert or Booking Engine.
4. The broker then fetches the JAR from the identified URL, adds it to its class loader, and instantiates the concrete TravelExpertProxy, or BookingEngineProxy, class, calling its mediation method.
5. As a result, the specific mediation required is *injected* at runtime into the broker.

In the IT2Rail technical demonstration the JAR files specific to a mediation are stored in the Asset Manager, as depicted in the following figure:

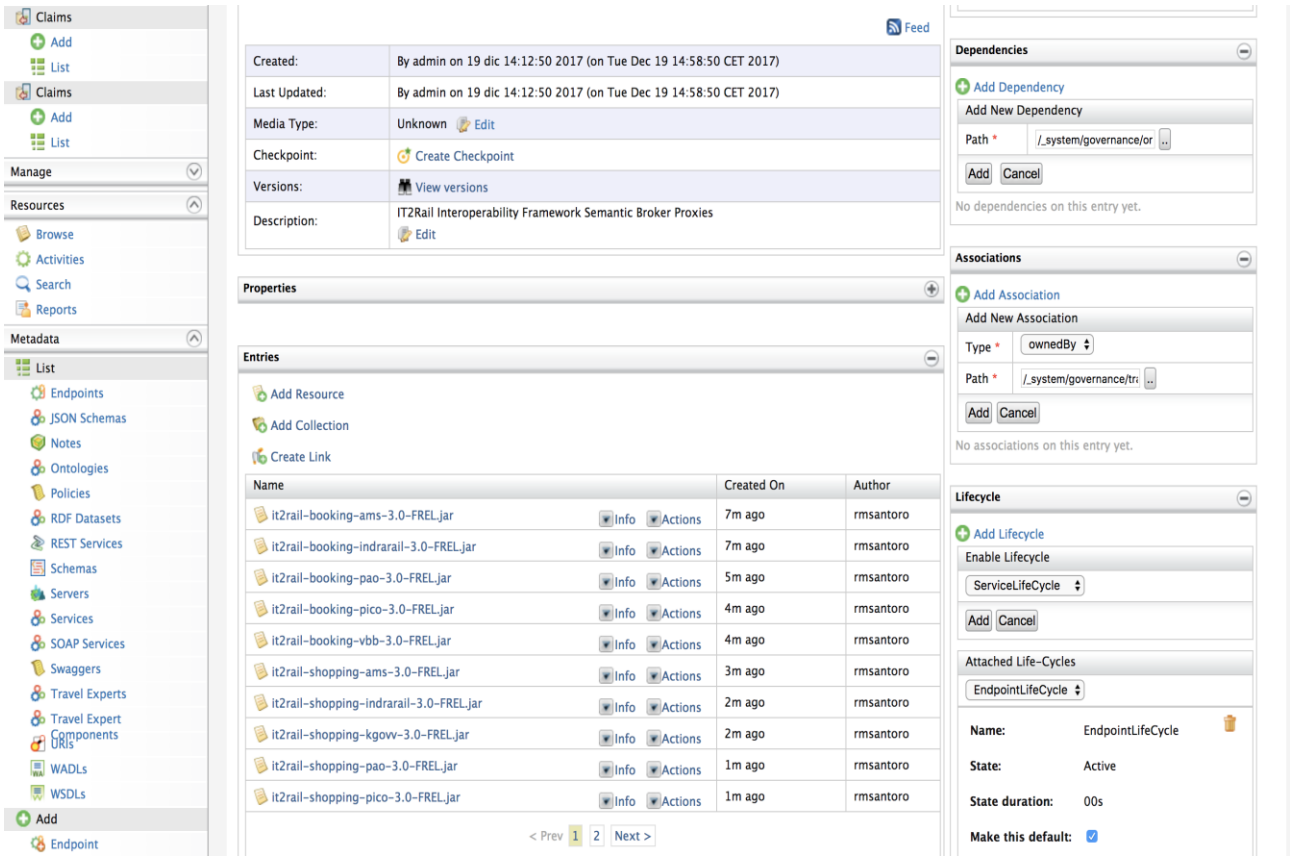


Figure 41: Asset Manager - Semantic Broker Proxies

As can be seen from the picture, JARs are associated with dependencies, associations and Lifecycle objects, providing tooling for implementation of governance procedures.
As a matter of illustration we show in the following picture these associations in the case of the AMS Booking Engine Proxy JAR file

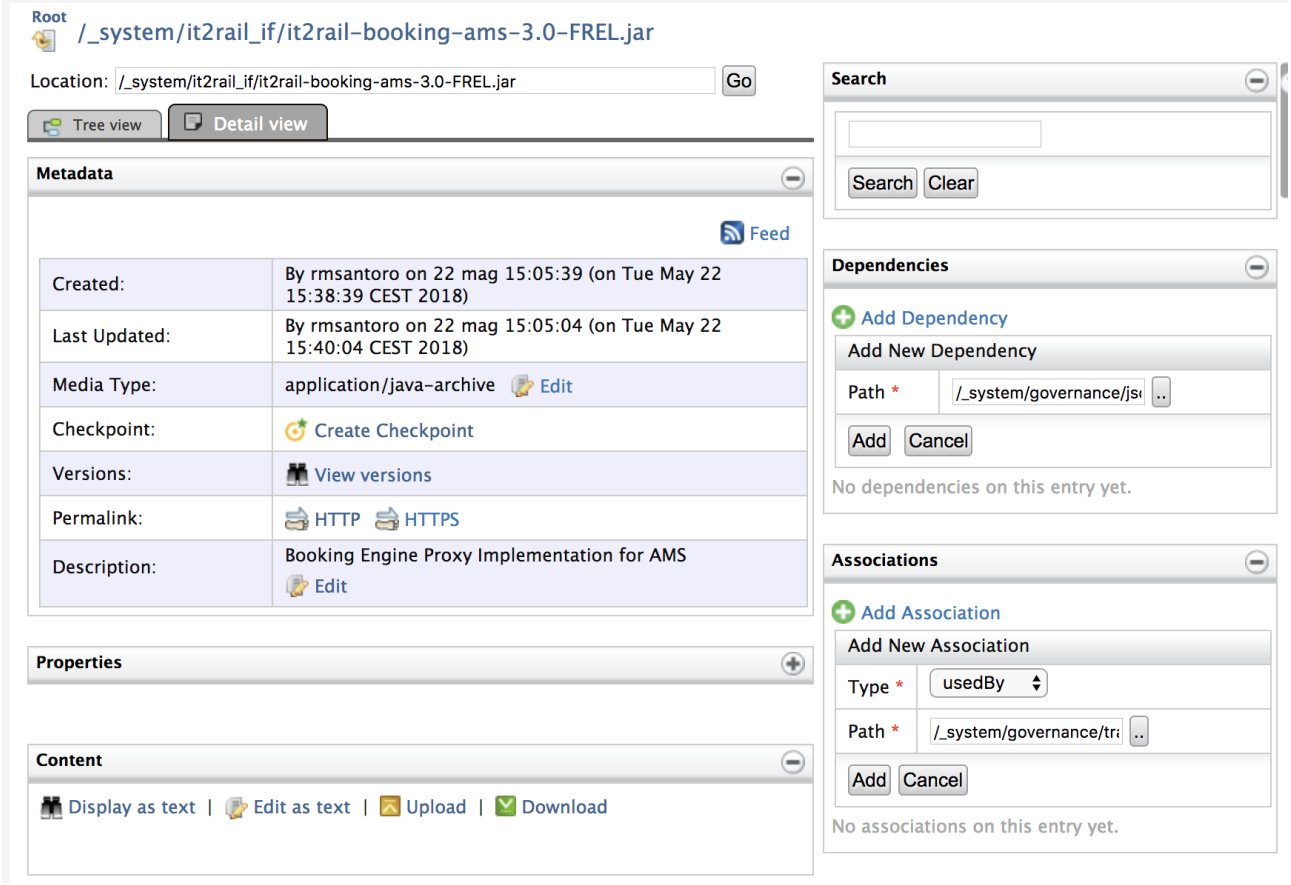


Figure 42 - AMS Booking Engine proxy governance associations

A JAR file stored in the Asset Manager under governance workflows typically packages the concrete implementation of the Proxy, the messages/data structures for the specific Travel Expert or Booking Engine semantically annotated with terms from the ontology, and any other resources that may be needed for the specific mediation task, such as inference rules. As an example, the following figure displays the content of the it2rail-booking-indrarail-3.0-FREL JAR file used to mediate the execution of booking, issuance and cancelations for the IndraRail Booking Engine:

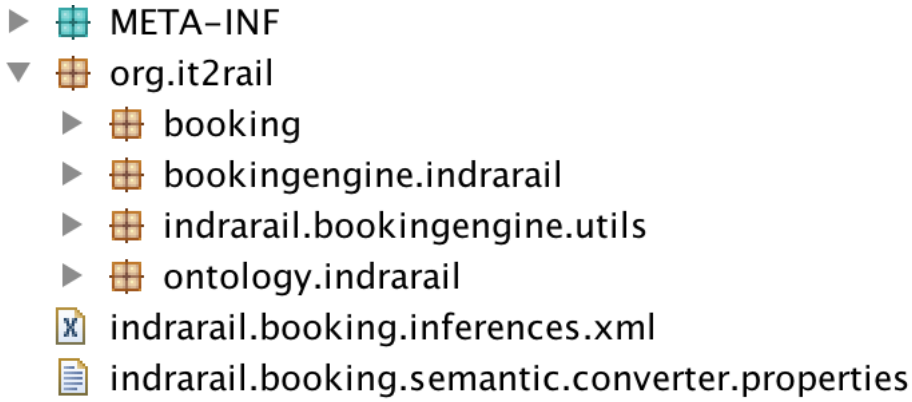


Figure 43: Example "mediation" JAR file contents

9.7 NeTEX Producer Service

In order to exercise the semantic interoperability concept and design an additional service has been developed and deployed, the NeTEx Producer service, which in fact is an alternate implementation of the Interoperability Framework's Location Resolver that uses the NeTEX 1.03 data schema for Stop Places. The service implements the GetNetex operatiom of the NeTEx_wsProducer-Document.wsdl web service interface. It is packaged according to the general technique used throughout the implementation of the Interoperability Framework. The following picture depicts the MAVEN project dependencies:

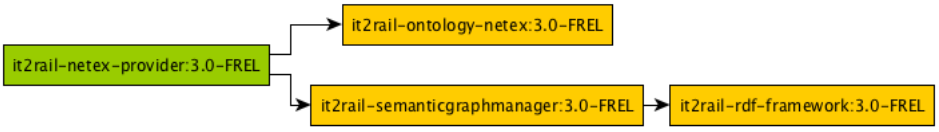


Figure 44: NeTEX Producer service packaging

As shown in the figure, the it2rail-netex-provider-3.0-FREL WAR file implements the interface and includes JAXB binding classes for NeTEX 1.3 schemas annotated with terms from the it2rail ontology. The service re-uses the Semantic Graph Manager and RDF Framework, like all others in the Interoperability Framework, to operate on the locations semantic graph.

The following figure shows a sample request for Stop Places in a bounding box corresponding to Barcelona, Spain:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:sir="http
   <soapenv:Header/>
   <soapenv:Body>
      <sir:GetNeTexService>
         <Request>
            <siri:ServiceRequest>
               <siri:ServiceRequestContext>
                    <siri:RequestTimeout>P0DT0H30M0.000S</siri:RequestTimeout>
                    <siri:DeliveryMethod>direct</siri:DeliveryMethod>
               </siri:ServiceRequestContext>
               <siri:RequestTimestamp>2004-12-17T09:30:47-05:00</siri:RequestTimestamp>
               <siri:RequestorRef>NADER</siri:RequestorRef>
               <siri:MessageIdentifier>mymsg122</siri:MessageIdentifier>

               <netex:DataObjectRequest>
                    <siri:RequestTimestamp>2001-12-17T09:30:47.0Z</siri:RequestTimestamp>
                    <siri:MessageIdentifier>mymsg12201</siri:MessageIdentifier>
                    <netex:topics>
                         <netex:NetworkFrameTopic>
                              <netex:Current/>
                              <netex:SiteFrameRef ref="abc:SiteFrame:1234">REQUEST</netex:S
                              <netex:NetworkFilterByValue>
                                <!-- Barcelona, Spain -->
                                <netex:BoundingBox>
                                        <netex:UpperLeft>
                                                <netex:Longitude>2.0719</netex:Longitude>
                                                <netex:Latitude>41.4686</netex:Latitude>
                                        </netex:UpperLeft>
                                        <netex:LowerRight>
                                                <netex:Longitude>2.2273</netex:Longitude>
                                                <netex:Latitude>41.32115</netex:Latitude>
                                        </netex:LowerRight>
                                </netex:BoundingBox>
                             </netex:NetworkFilterByValue>
                         </netex:NetworkFrameTopic>
                    </netex:topics>
                    <netex:Policy>
                         <Language>en</Language>
                    </netex:Policy>
               </netex:DataObjectRequest>
            </siri:ServiceRequest>
         </Request>
      </sir:GetNeTexService>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 45: Sample NeTEX Producer request

On processing the request, the service returns the following response (fragment shown):

```
<ns5:GetNeTexServiceResponse xmlns:ns2="http://www.siri.org.uk/siri" xmlns:ns3="http://www.netex.org.uk
   <Answer>
      <ns2:ServiceDelivery>
         <ns2:AbstractFunctionalServiceDelivery xsi:type="ns3:DataObjectDeliveryStructure" xmlns:xsi="h
            <ns3:dataObjects>
               <ns3:CompositeFrame version="3.0-FREL" id="it2rail">
                  <ns3:codespaces>
                     <ns3:Codespace id="it2rail:infrastructure:codespace">
                        <ns3:Xmlns>it2rail</ns3:Xmlns>
                        <ns3:XmlnsUrl>http://it2rail.org/ontology/infrastructure#</ns3:XmlnsUrl>
                     </ns3:Codespace>
                  </ns3:codespaces>
                  <ns3:FrameDefaults>
                     <ns3:DefaultCodespaceRef ref="it2rail:infrastructure:codespace"/>
                  </ns3:FrameDefaults>
                  <ns3:frames>
                     <ns3:SiteFrame>
                        <ns3:Name lang="en">100 StopPlaces in the requested bounding box</ns3:Name>
                        <ns3:stopPlaces>
                           <ns3:StopPlace id="http://it2rail.org/infrastructure/tmb/TMB:1.1138-S">
                              <ns3:keyList>
                                 <ns3:KeyValue>
                                    <ns3:Key>TMB</ns3:Key>
                                    <ns3:Value>TMB:1.1138</ns3:Value>
                                 </ns3:KeyValue>
                              </ns3:keyList>
                              <ns3:Name>Torre Baró | Vallbona</ns3:Name>
                              <ns3:Centroid>
                                 <ns3:Location>
                                    <ns3:Longitude>2.179884</ns3:Longitude>
                                    <ns3:Latitude>41.459194</ns3:Latitude>
                                 </ns3:Location>
                              </ns3:Centroid>
                              <ns3:PublicCode>TMB:1.1138</ns3:PublicCode>
                              <ns3:TransportMode>metro</ns3:TransportMode>
                              <ns3:StopPlaceType>metroStation</ns3:StopPlaceType>
                           </ns3:StopPlace>
```

Figure 46: Fragment of NeTEX Producer service response

10. TECHNICAL DEMONSTRATION TEST ENVIRONMENT AND PROCEDURE

10.1 TECHNICAL DEMONSTRATION INSTALLATION

The Interoperability Framework technical demonstration executes in the environment described in section 8.5 Operational Environment of this document. The environment is equipped with

- An installation of the open source Apache Tomcat web application server[6].
- An installation of the open source Apache WSO2 Carbon middleware[7].
- An installation of the open source Ontotext GraphDB semantic graph database[8].

all using the Java Development Kit 1.8 and runtime.

[6] http://tomcat.apache.org/

[7] https://wso2.com/products/carbon/

[8] https://ontotext.com/products/graphdb/

The following pictures show details of the three installations:

Server Information						
Tomcat Version	**JVM Version**	**JVM Vendor**	**OS Name**	**OS Version**	**OS Architecture**	**Hostname**
Apache Tomcat/7.0.69	1.8.0_111-b14	Oracle Corporation	Linux	3.10.0-514.2.2.el7.x86_64	amd64	trenitalia.leonardo

Figure 47 - Apache Tomcat installation

Operating System	
OS Name	Linux
OS Version	3.10.0-514.2.2.el7.x86_64

Operating System User	
Country	IT
Home	/home/cefriel
Name	cefriel
Timezone	Europe/Rome

Java VM	
Java Home	/usr/java/jdk1.8.0_111/jre
Java Runtime Name	Java(TM) SE Runtime Environment
Java Version	1.8.0_111
Java Vendor	Oracle Corporation
Java VM Version	25.111-b14

Registry	
DBMS	H2
DBMS Version	1.3.175 (2014-01-18)
DBMS Driver	H2 JDBC Driver
DBMS Driver Version	1.3.175 (2014-01-18)
DBMS URL	jdbc:h2:repository/database/WSO2CARBON_DB

Figure 48 - Apache WSO2 Carbon installation

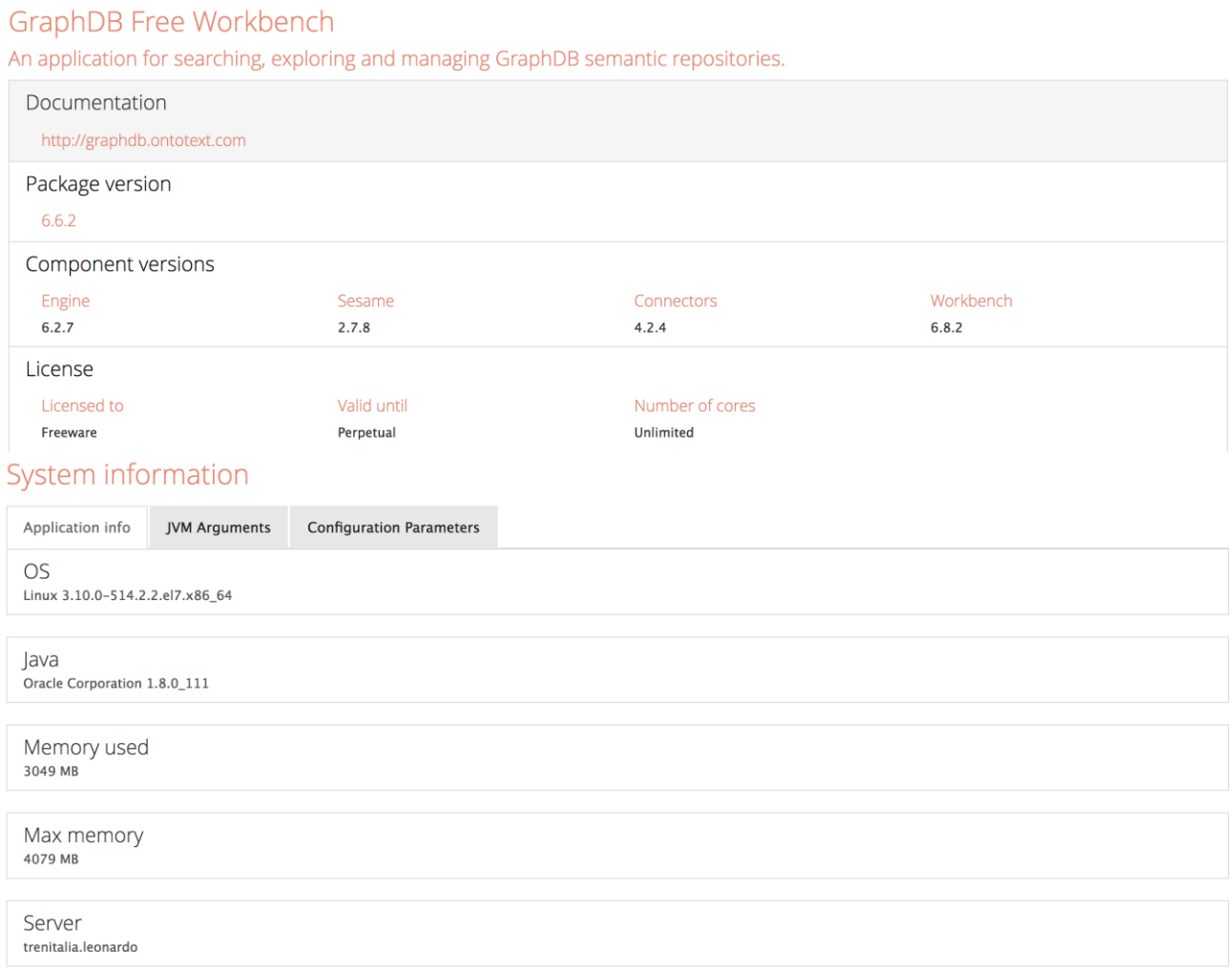


Figure 49 - Ontotext GraphDB Semantic Graph database

The Interoperability Framework services are implemented as Web Archive (WAR) files as described in Figure 32 and deployed on the Apache Tomcat web application server. The following figure shows the deployed services on the Apache Tomcat management console:

Applications			
Path	**Version**	**Display Name**	**Running**
/	*None specified*	Welcome to Tomcat	true
/it2rail-bookingengine-broker-0.0.1-FREL	*None specified*	Booking Engine Broker	true
/it2rail-location-identifier-0.0.1-FREL	*None specified*	Location Identification	true
/it2rail-locations-resolver-1.0-AREL	*None specified*	Location Resolver	true
/it2rail-navitia-decoder-0.0.1-FREL	*None specified*	Event Source Resolver - Navitia Decoder	true
/it2rail-netex-provider-3.0-FREL	*None specified*	NeTEX provider	true
/it2rail-travelexpert-broker-0.0.1-FREL	*None specified*	Travel Expert Broker	true
/it2rail-travelexpert-resolver-1.0-AREL	*None specified*	Travel Expert Resolver	true
/manager	*None specified*	Tomcat Manager Application	true

Figure 50 - Interoperability Framework deployed services

Together with the assets installed in the Asset Manager, as described in Section 9.6 of this document, these services provide IT2Rail Shopping, Booking/Ticketing and Trip Tracking applications the semantic interoperability they need to interact with externally provided Travel Experts and Booking Engines.

10.2 External Travel Expert and Booking Engine services

The following external Travel Expert services have been annotated for use in the technical demonstration:

- SNCF (mainline French Rail) PAO services:
  a) <endpoint>/it2r/sales/searchSolutions.

- AMS (long distance Coach operators, Czech Republic) eshopcv services:
  a) <endpoint>/v1/Connection.
  b) <endpoint>/v1/ConnectionInfo.

- Trenitalia (mainline Italian Rail) PICO Services:
  a) <endpoint>/Sale/SaleProcess/SolutionEngine/TravelSolution/search.
  b) <endpoint>/Sale/SaleProcess/SaleCoordinator/searchBase.

- RENFE (mainline Spanish Rail), Indra Rail services:
  a) <endpoint>/Rail_TSP/NewTSP2/GetItineraries.
  b) <endpoint>/Rail_TSP/NewTSP2/Availability.
  c) <endpoint>/Rail_TSP/NewTSP2/Trains.

- KGOVV (Austrian Public Transport), HaCon services:
  a) <endpoint>/openapi/vao/restproxy/trip.

- TMB (Madrid, Barcelona Public Transport) Indra Rail services:
  a) <endpoint>/HMI2_APP/service/otp/getRoute.

- VBB (Belin / Brandenburg Public Transport), HaCon services:
  a) <endpoint>/restproxy/trip.

The following external Booking Engine services have been annotated for use in the technical demonstration:

- SNCF (main line Rail operator), PAO services:
  a. <endpoint>/it2r/sales/bookProposals.
  b. <endpoint>/it2r/sales/createSalesContract.
  c. <endpoint>/it2r/sales/cancelBooking.
  d. <endpoint>/it2r/sales/cancelTickets.

- AMS (long distance coach services), eshopcv services:
  a. <endpoint>/v1/SeatBlock/.
  b. <endpoint>/v1/Ticket/.

- Trenitalia (main line Rail operator), PICO Services:
  a. <endpoint>/Sale/SolutionEngine/CatalogReservation.
  b. <endpoint>/Sale/SaleProcess/OrderProcess.

- RENFE (mainline Rail operator) Indra Rail services:
  a. <endpoint>/Rail_TSP/NewTSP2/LockInventory.
  b. <endpoint>/Rail_TSP/NewTSP2/IssueToken.
  c. <endpopint>/Rail_TSP/NewTSP2/BookingInfo.
  d. <endpoint>/Rail_TSP/NewTSP2/ReleaseInventory.

- VBB (Public Transport Berlin/Brandenburg) HaCon services:
  a. <endpoint>/shopping/ShoppingMessages/VBB/purchaseRequest .
  b. <endpoint>/ shopping/ShoppingMessages/VBB/retrieveRequest.

10.3 TECHNICAL DEMONSTRATION TEST

For both unit and integration testing of the It2Rail pilot demonstration the SoapUI[9] tool has been used extensively to conduct both manual and automated testing campaigns.

Additionally, logging instructions have been inserted in the code using the Apache Log4j framework controlled by configuration parameters stored in a logging configuration file.

[9] https://www.soapui.org/

The following figure shows a snapshot of the SoapUI tool configured for testing of the main Interoperability Framework services:

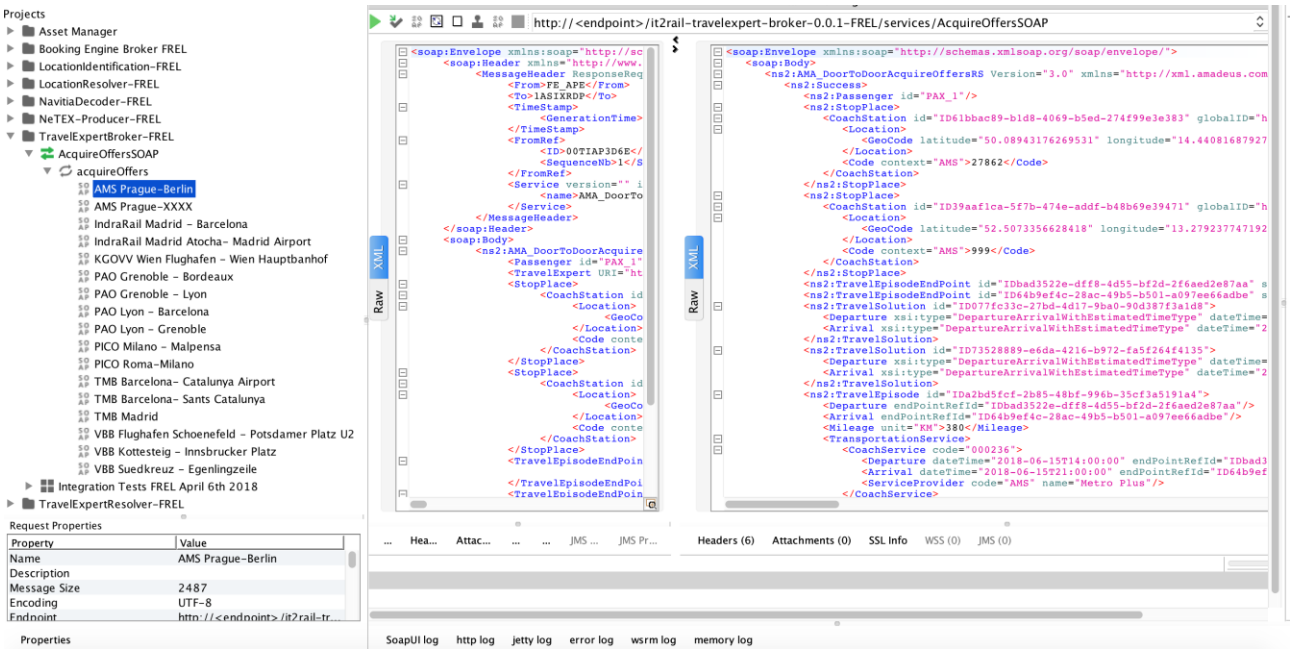


Figure 51 - SoapUI configuration for testing campaign

The figure expands, for illustration purposes, the Travel Expert Broker service with a set of test requests for the acquireOffers operation (on the left), and the request and response messages obtained for the AMS Prague Berlin test instance.

Similar test requests have been prepared and executed for each of the additional services listed under “projects” in the leftmost panel of the SoapUI screen.

The next figure displays a screenshot of an automated test campaign of the Travel Expert Broker service conducted on April 6th 2018:

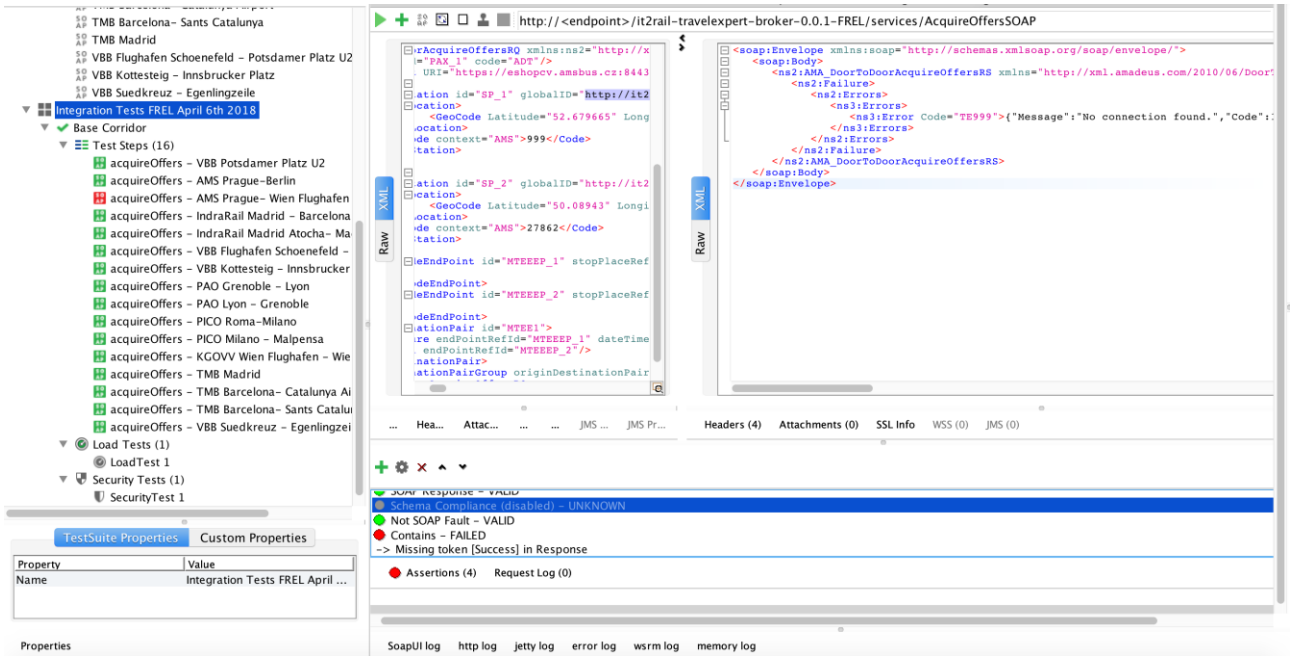


Figure 52 - Travel Expert Broker automated test

The “Base Corridor” test suite displayed on the left pane was performed with all test steps passed (in green), except one (in red) for an AMS Prague to Wien itinerary. The failure consisted in the remote AMS Travel Expert returning a “No connection found” message, indicating that no solutions were available at the Travel Service provider for the specified itinerary and date.

Similar campaigns where performed for all other Interoperability Frameworks.