INFORMATION TECHNOLOGIES FOR SHIFT TO RAIL

D1.7 – Proof-of-Concept Packaged Resolvers Additional Features

Due date of deliverable: 31/10/2017

Actual submission date: 14/05/2018

Leader/Responsible of this Deliverable: LEONARDO

Reviewed: Y

Document status		
Revision	Date	Description
1	27/07/2017	First draft of the deliverable
1.1	22/02/2017	Added a paragraph describing the operational environment in which the Interoperability Framework is deployed.
2	26/02/2018	Final version for TMC approval (first version)
3	27/04/2018	Final version for TMC approval
4	14/05/2018	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 Additional Release (AREL) of IT2Rail. Most of descriptions have been provided following the same approach used during the Core Release (CREL) as that release represented the building block for later enhancements planned both in the Additional and Final Release (FREL).

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 (AREL) 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 of the design, based on the it2rail rdf framework, which provides semantic data discovery, query an aggregation over linked, distributed triple stores.
3. The **Location Resolver** service of the design, 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.

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.

All terms and acronyms are defined in the IT2Rails glossary.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

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
	D1.6 Proof of Concept Packaged Resolvers Core Features		

2. PRINCIPLES AND PURPOSE

In order to make rail travel across Europe attractive the rail system must be perceived and used by Customers 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 organisation, 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 centralised 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 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. 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 […].[1]"

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.

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

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 Provider** 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 transportation 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.

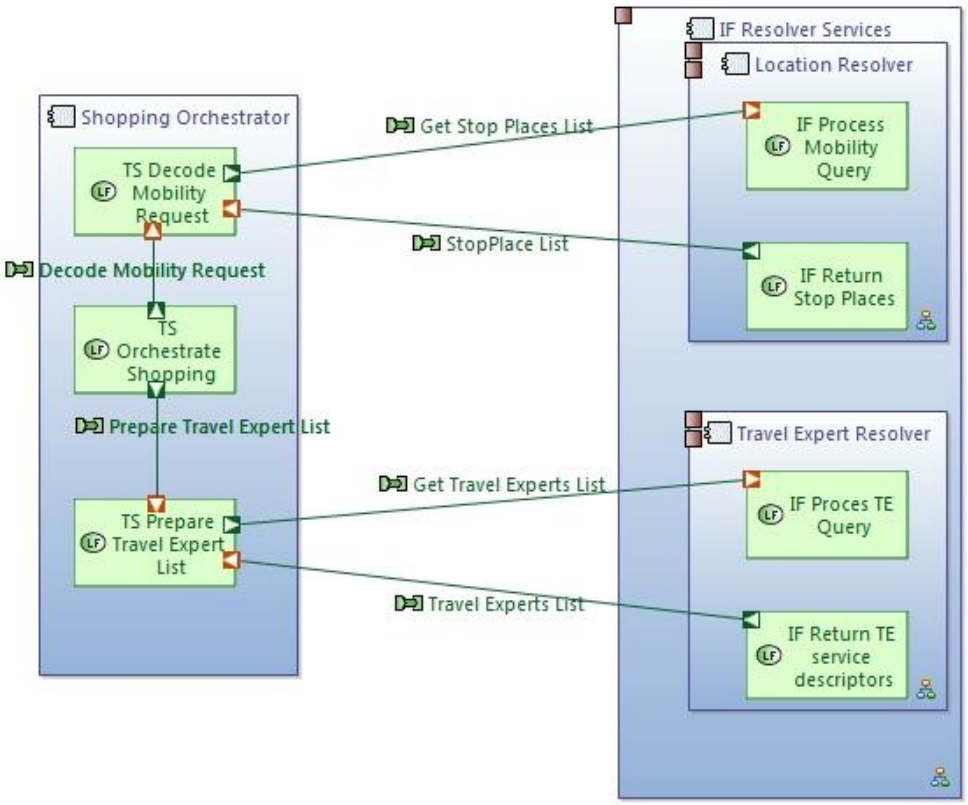


Figure 1: Resolver Services in relationship with Travel Shopping

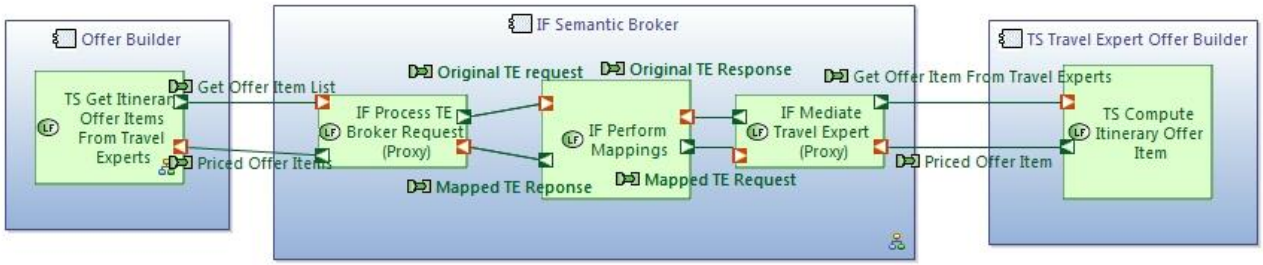


Figure 2: Semantic Broker in relationship with Travel Shopping

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

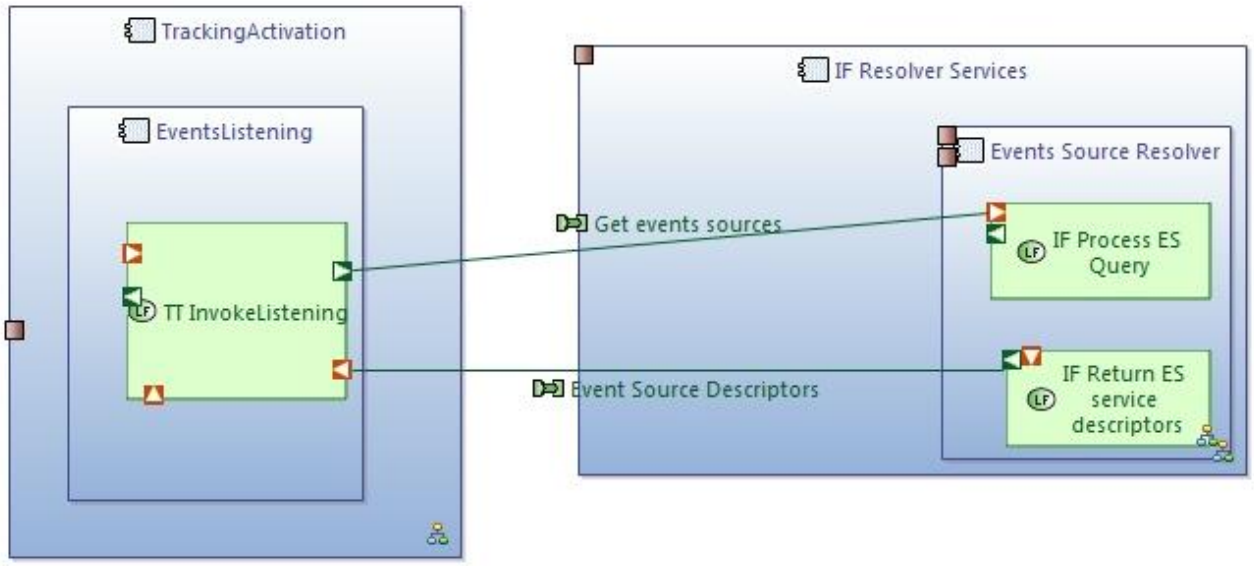


Figure 3: Semantic Broker in relationship with Trip Tracker

3.1 USE CASES

3.1.1 IF in Manage Mobility and travel rights delivery Use Case

The Interoperability Framework provides capabilities included in the WP2 Shop- Provide Itinerary Offers and Shop – Provide itinerary offer items capabilities of the “Manage Mobility and travel rights delivery “Use Case.

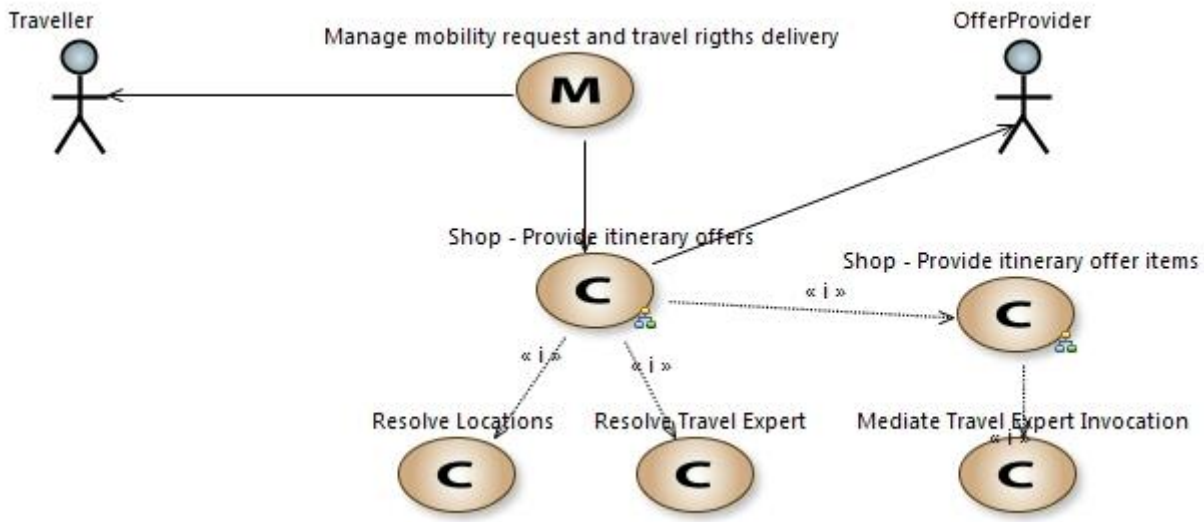


Figure 4: Manage Mobility and travel rights delivery Use Case

The Interoperability Framework participates in the Use Case by providing the Resolve Locations, Resolver Travel Expert and Mediate Travel Expert Invocation capabilities to the Use Case. These capabilities are described in the “Capabilities” chapter of this document.

3.1.2 IF in Track Itinerary Use Case

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

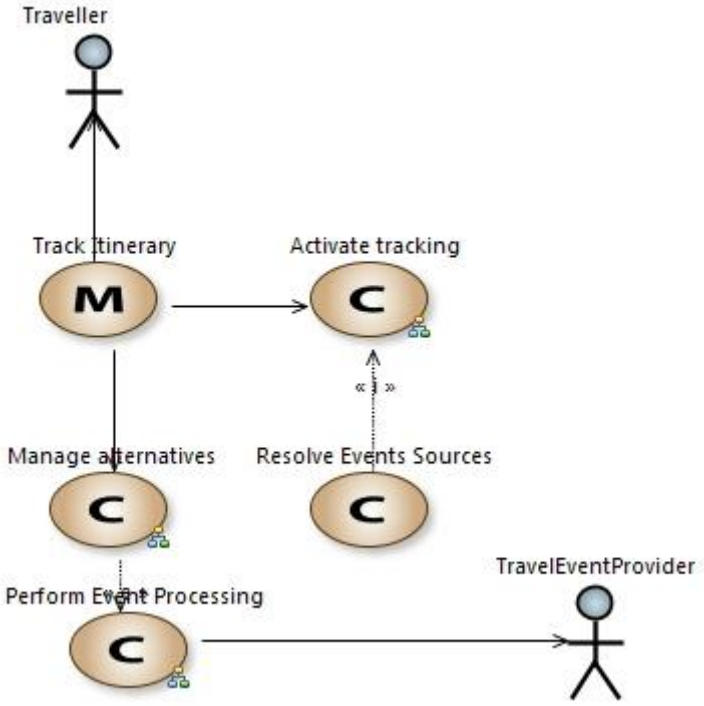


Figure 5: Track Itinerary Use Case

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.

3.1.3 Manage IF Assets and Publish Resources

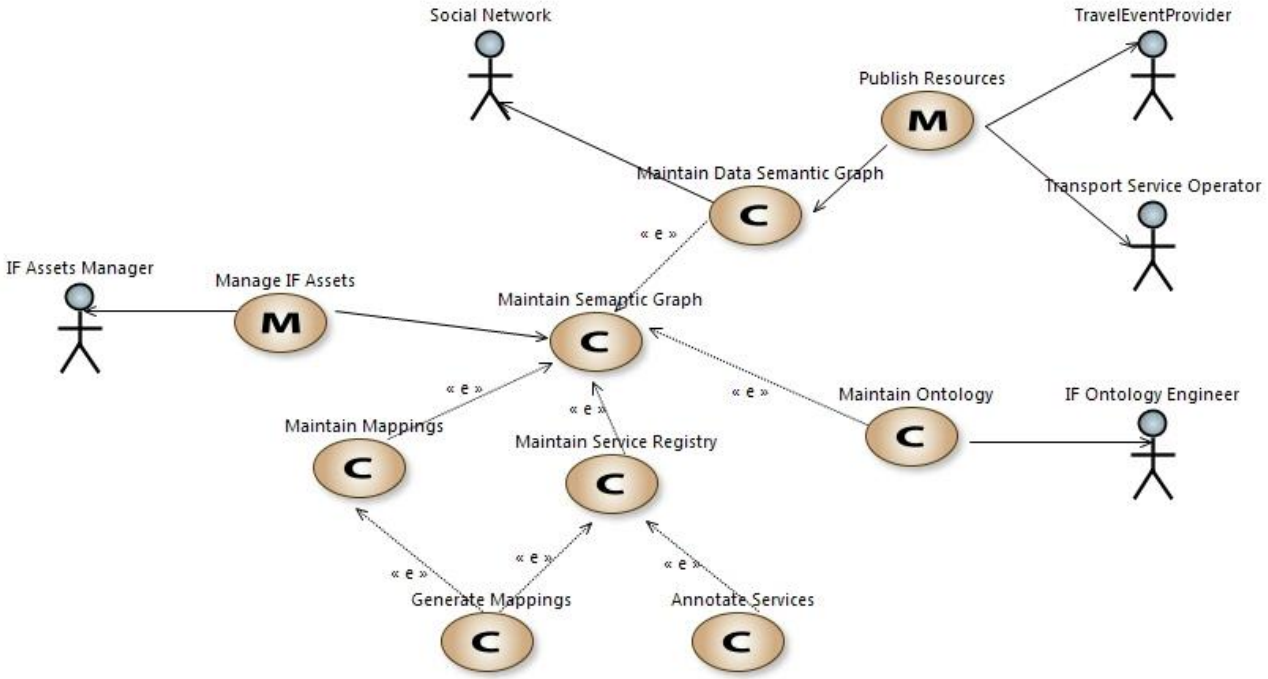


Figure 6: Manage IF Assets and Publish Resources Use Cases

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

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. CAPABILITIES

4.1 MANAGE IF ASSETS

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.

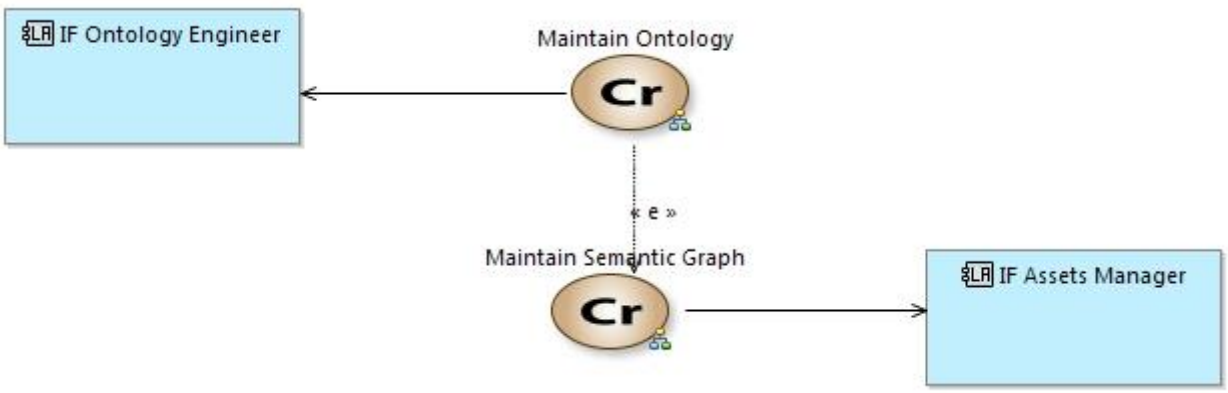


Figure 7: IF asset management capability - Maintain Ontology

4.1.3 Maintain Service registry

Extends Manage Semantic Graph to support 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.

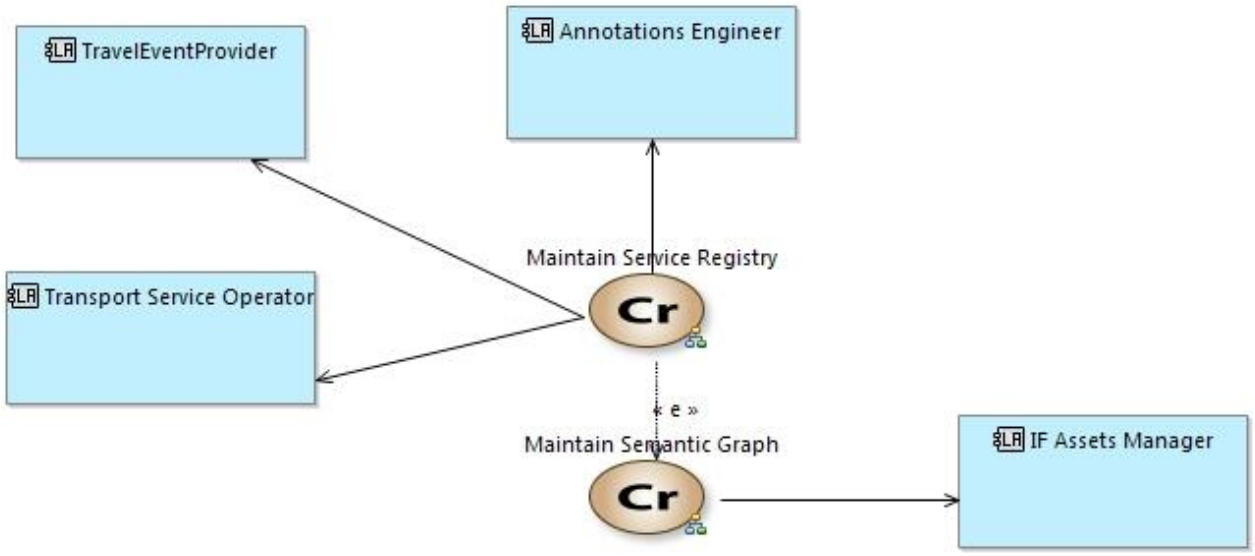


Figure 8: IF asset management capability – Maintain Service Registry

4.1.4 Maintain Data Semantic Graph

Extends Manage Semantic Graph to support import and rdfyication by the Annotations Engineer Actor of external data sources provided by Transport Service Operator, Travel Event Provider and Social Network external Actors.

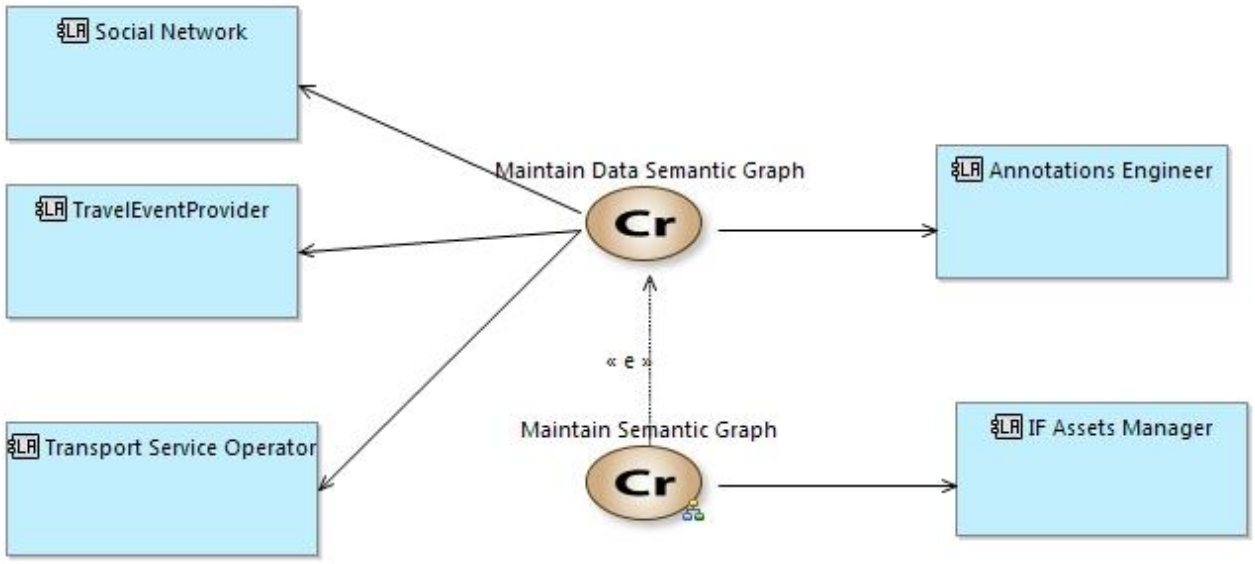


Figure 9: IF asset management capability – Maintain Data Semantic Graph

4.2 IF RESOLVER SERVICES

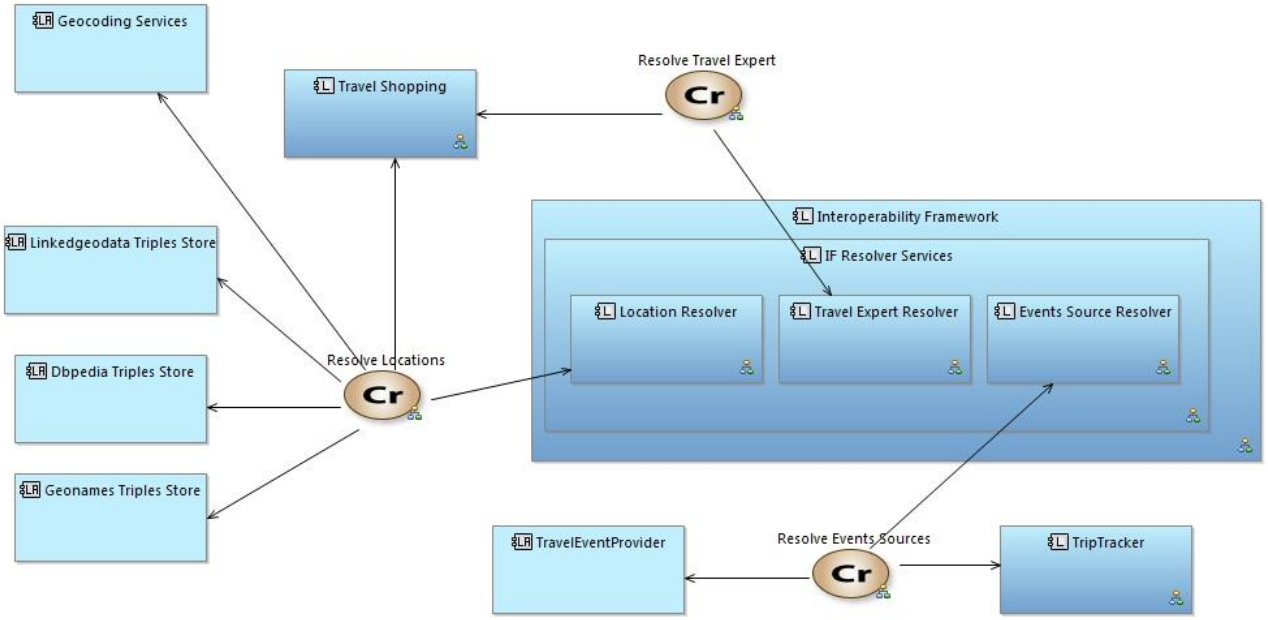


Figure 10: IF Resolver Services

4.2.1 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 mode 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.

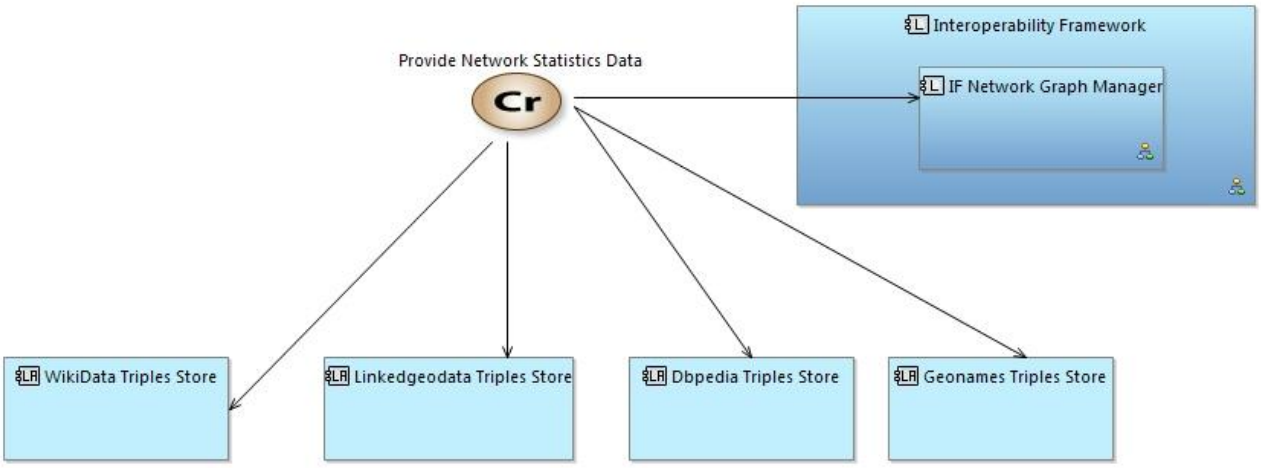


Figure 11: Provide Network Statistics Data capability

4.2.2 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

4.2.3 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

4.2.4 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

4.3 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.

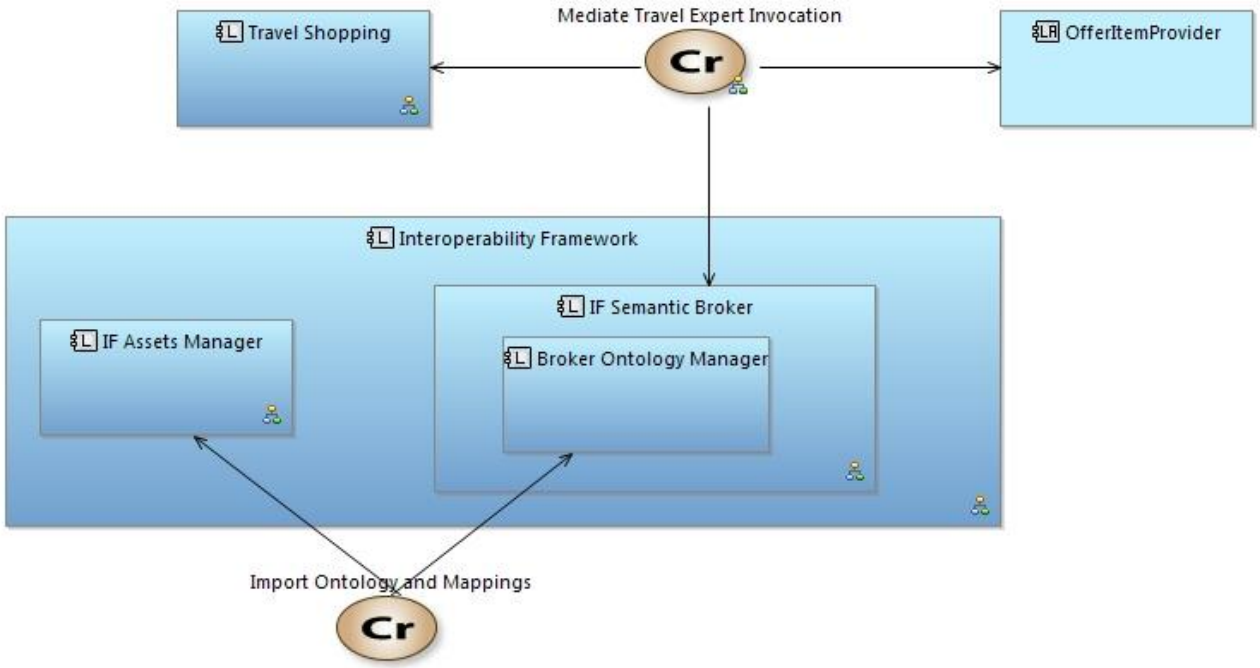


Figure 12: Mediate Travel Expert Invocation

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

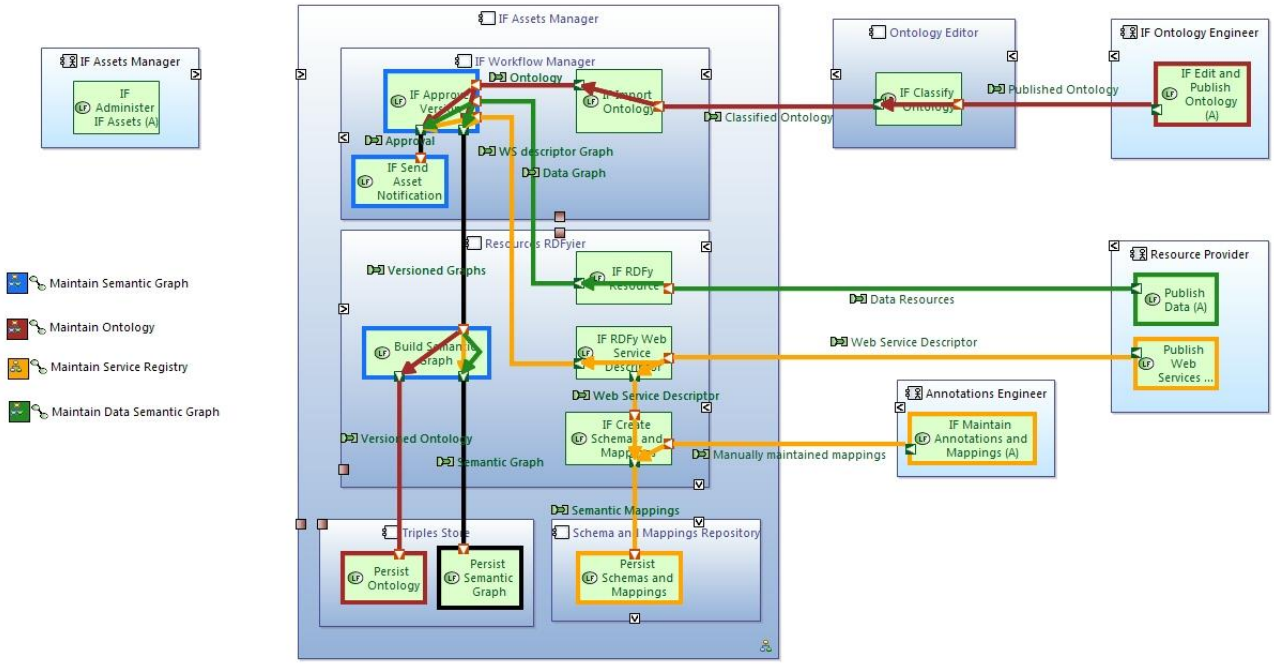


Figure 13: Manage IF Assets activity diagram

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 IF RESOLVER SERVICES

5.2.1 Provide Network Statistics Data

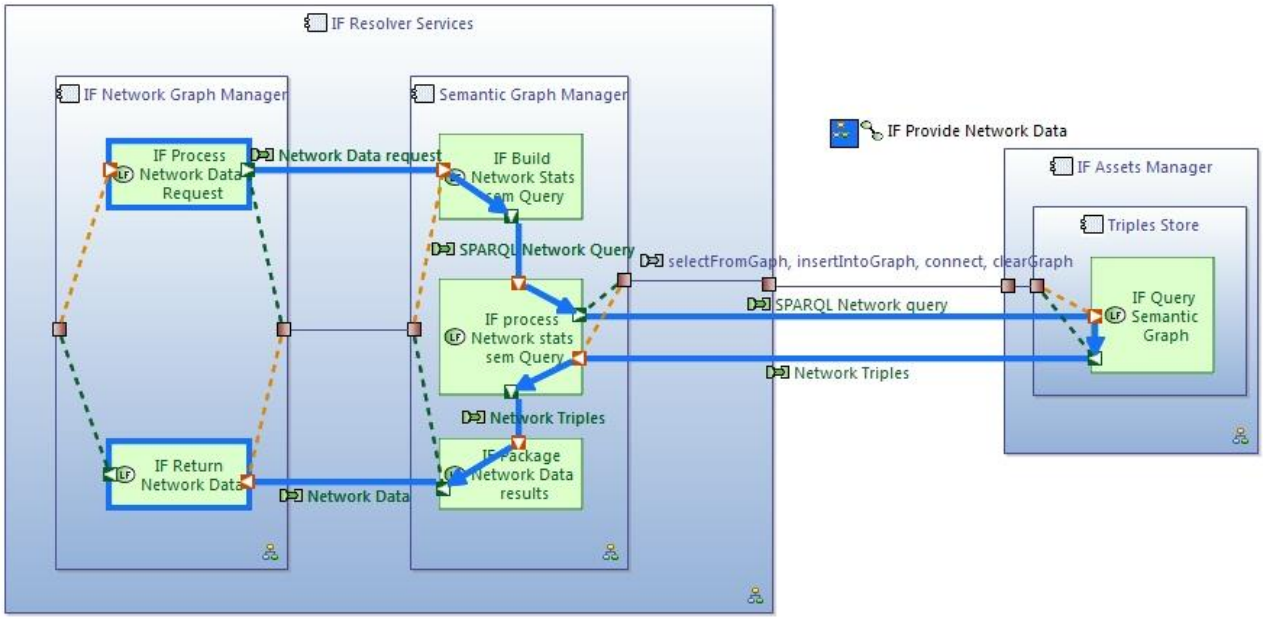


Figure 14: Provide Network Statistics Data activity diagram

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

5.2.2 Resolve Locations

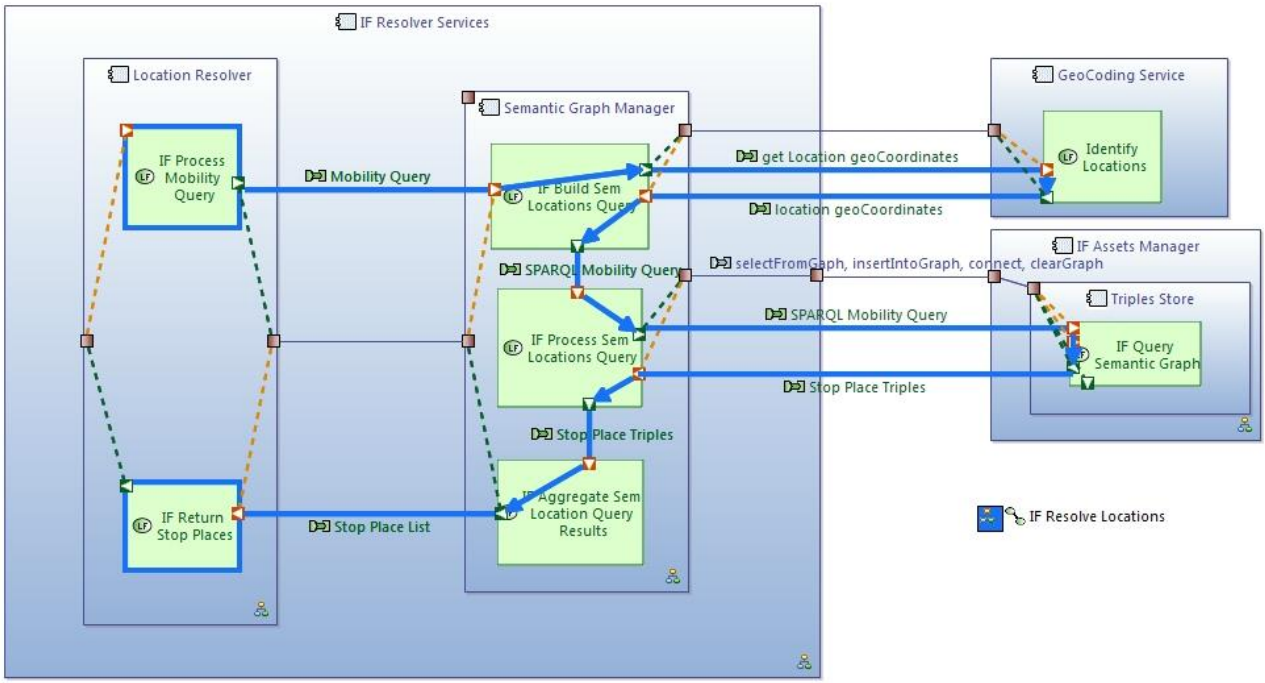


Figure 15: Resolve Locations activity diagram

Functions in Resolve Locations capability realisation

Logical Function	Description	Component
IF Process Mobility Query	Receives and validates mobility request	Location Resolver
Identify Locations	Gets geoCoordinates 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

5.2.3 Resolve Travel Expert

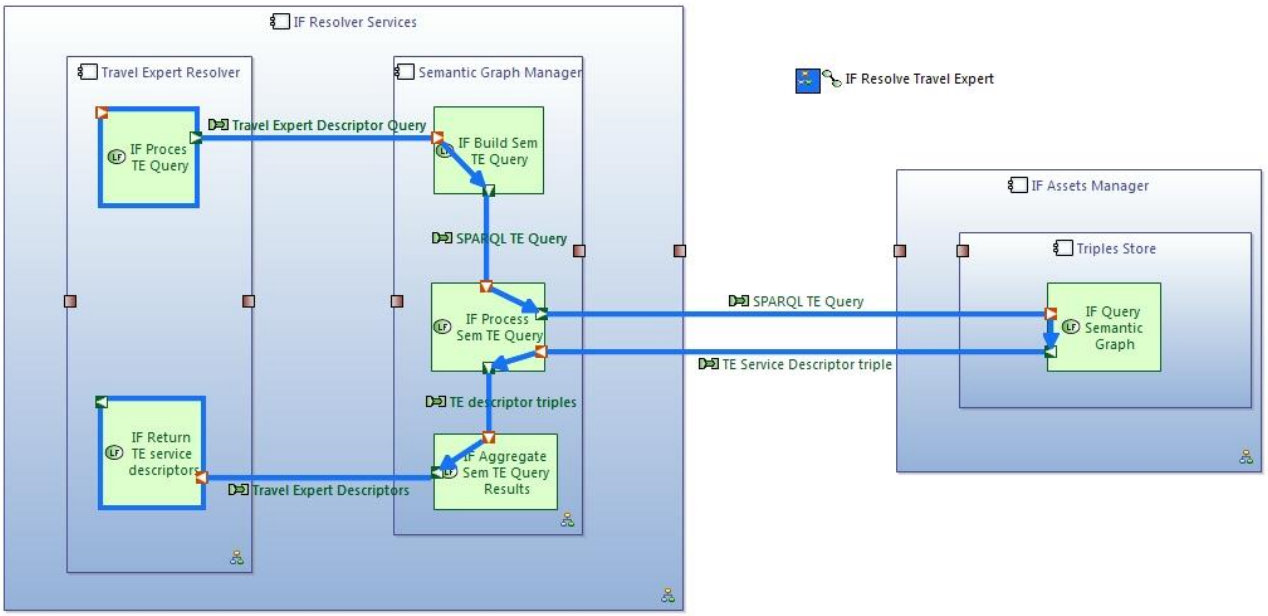


Figure 16: Resolve Travel Expert activity diagram

Functions in Resolve Travel Expert capability realisation

Logical Function	Description	Component
IF Proces 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

5.2.4 Resolve Events Source

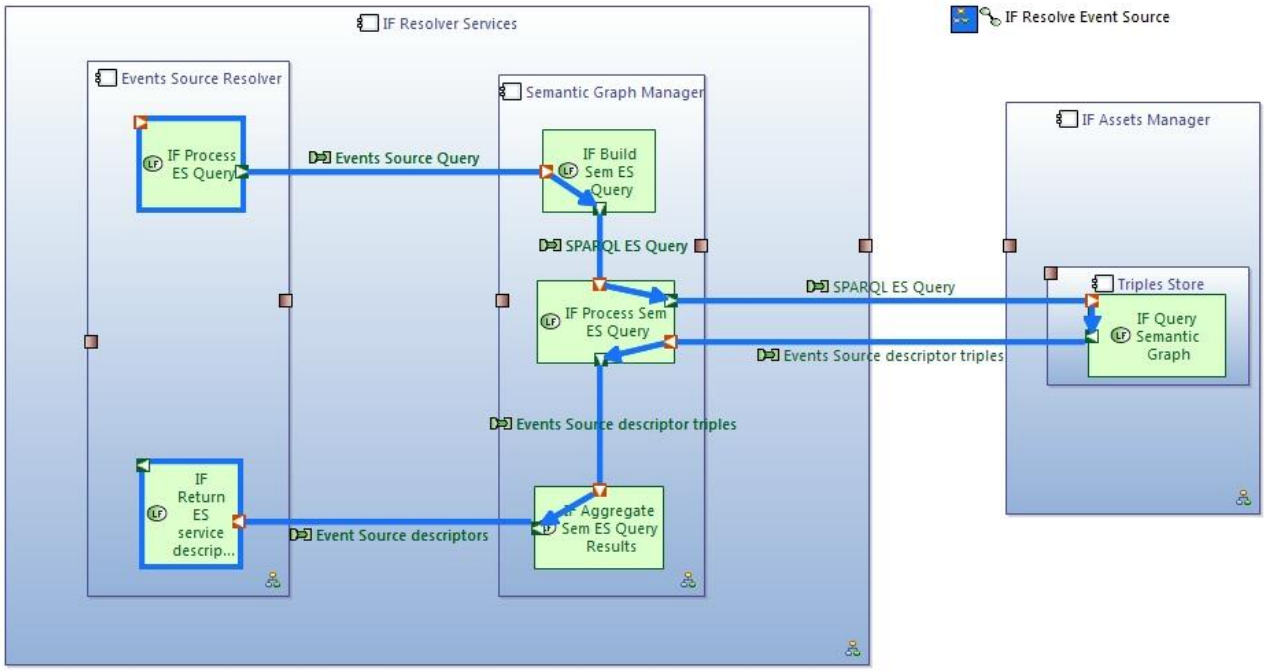


Figure 17: Resolve Events Source activity diagram

Functions in Resolve Events Source capability realisation

Logical Function	Description	Component
IF Process ES Query	Receives and validates Events Source request	Events Source Resolver
IF Build Sem ES Query	Transforms events source request and prepares events source semantic query	Semantic Graph Manager
IF Process Sem ES Query	Connects to Triple Stores and submits events source semantic query for processing	Semantic Graph Manager

IF Query Semantic Graph	Queries semantic graph and retrieves result triples	Triple Store
IF Aggregate Sem ES Query Results	Processes triples results set and generates events source descriptor results	Semantic Graph Manager
IF Return ES service descriptors	Generates list of Events Source descriptor response	Events Source Resolver

5.3 IF SEMANTIC BROKER

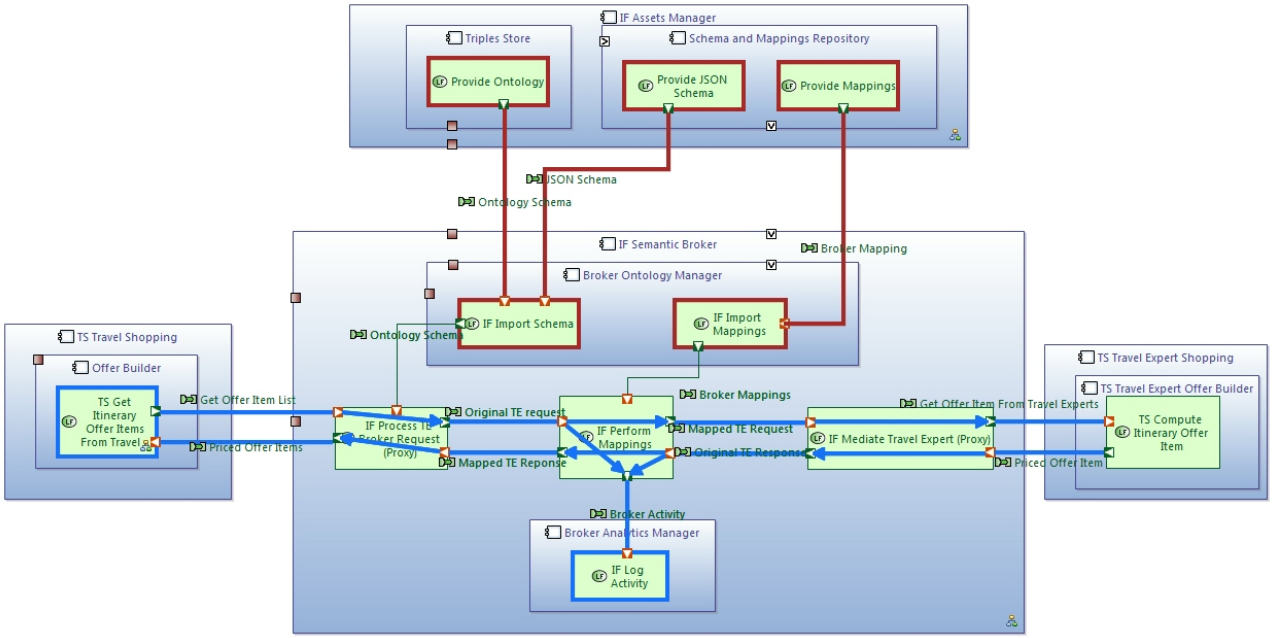


Figure 18: IF Semantic Broker activity diagram

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), 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	Semantic Broker
IF Perform Mappings	Applies transformation of response fromTravel 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)	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

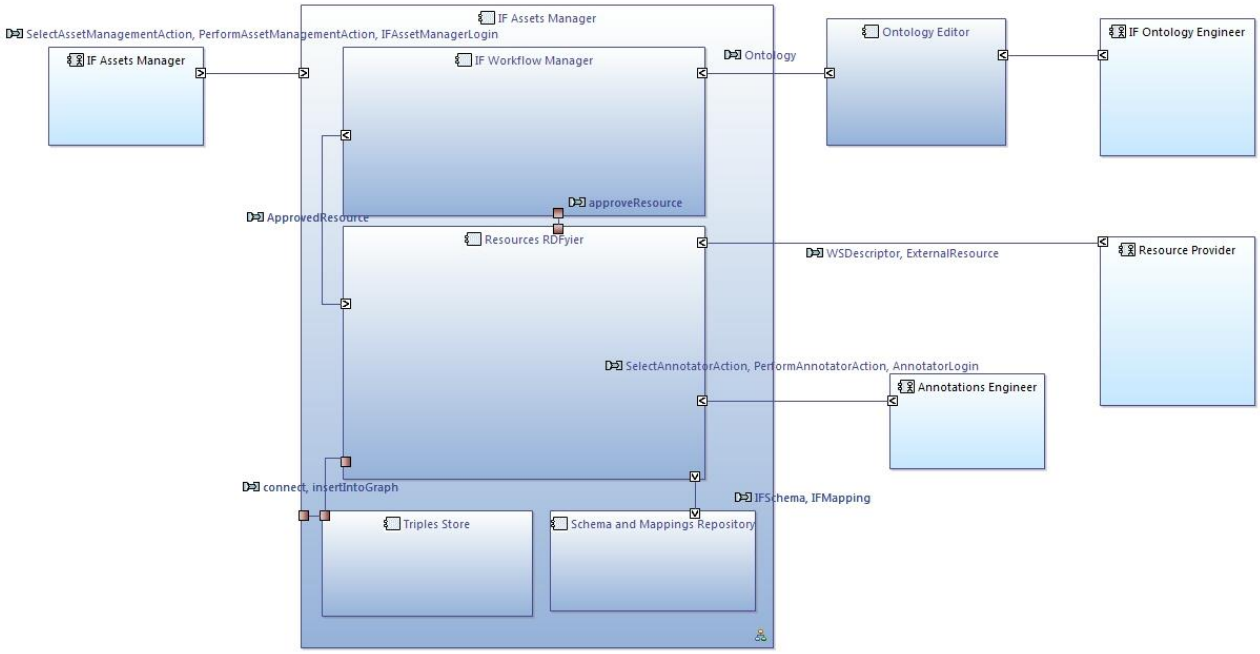


Figure 19: Exchanges across components of the IF Asset Manager

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

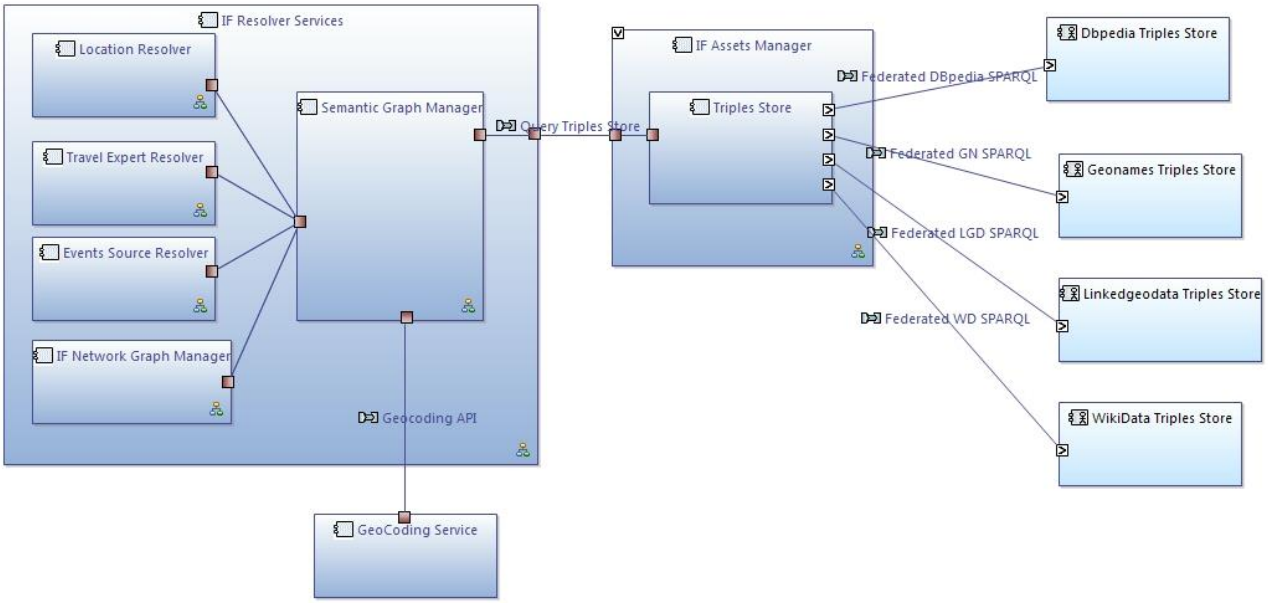


Figure 20: Exchanges across components of the IF Resolver Services

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.

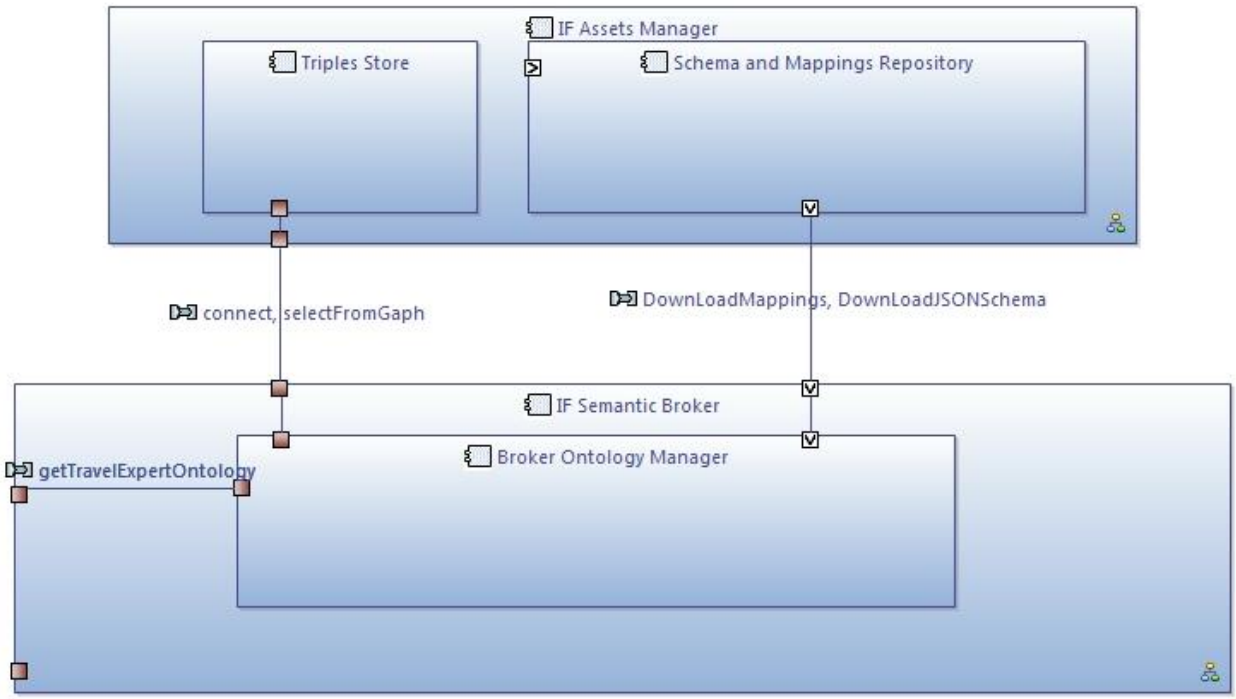


Figure 21: Exchanges across components of the IF Semantic Broker

7. INTERFACES

7.1 IF ASSETS MANAGER

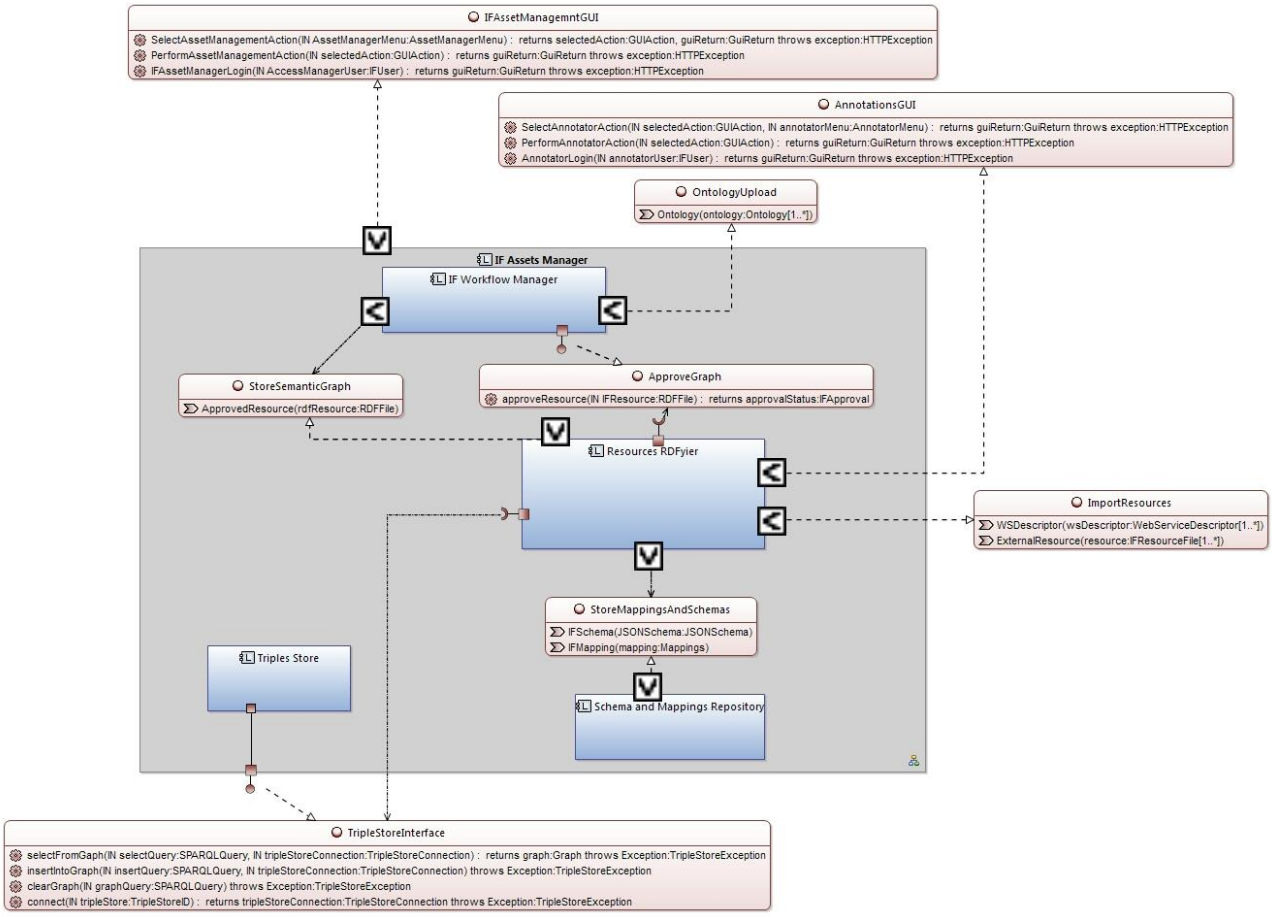


Figure 22: IF Assets Manager interfaces

7.1.1 External Interfaces

Table 2: Asset Management GUI interface

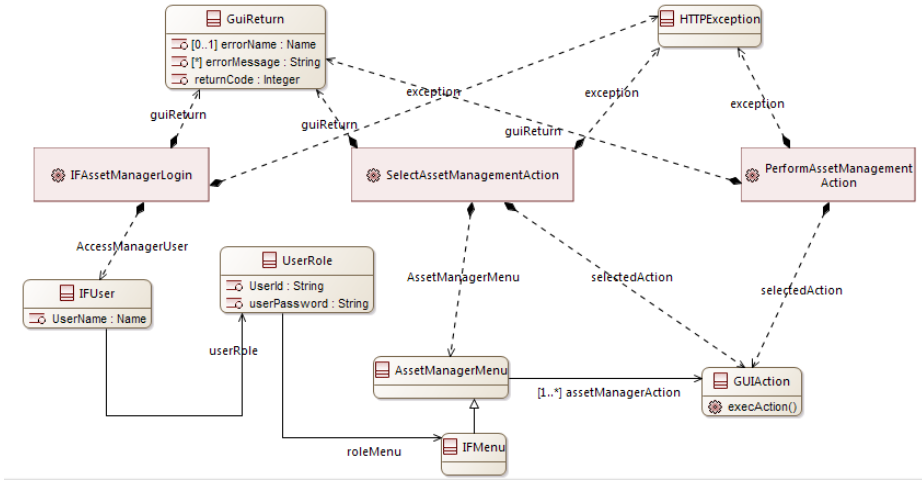
Interface ID:	**Asset Management GUI**
Interface Name:	IFAssetManagemntGUI SelectAssetManagementAction(IN AssetManagerMenu:AssetManagerMenu) : returns selectedAction:GUIAction, guiReturn:GuiReturn throws exception:HTTPException PerformAssetManagementAction(IN selectedAction:GUIAction) : returns guiReturn:GuiReturn 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:	Inexisting 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

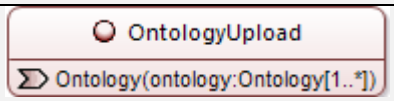
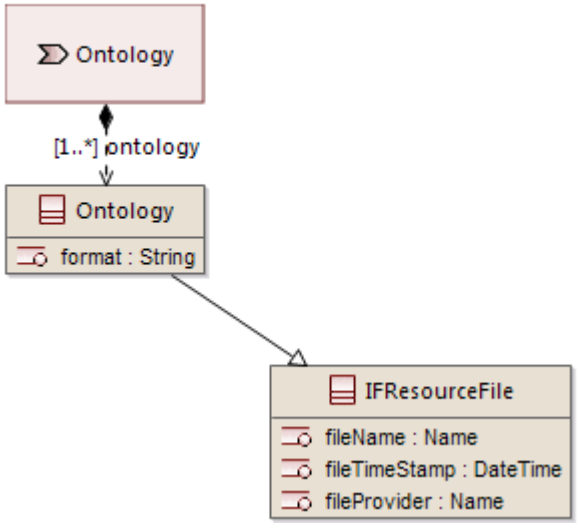
Interface ID:	**Ontology Upload**
Interface Name:	
Purpose of the Interface	Upload validated ontology files edited by external Ontology Editor for approval in the approval workflow process
Requestor:	Ontology Editor
Provider	Workflow Manager
Description:	Upload validated ontology files edited by external 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

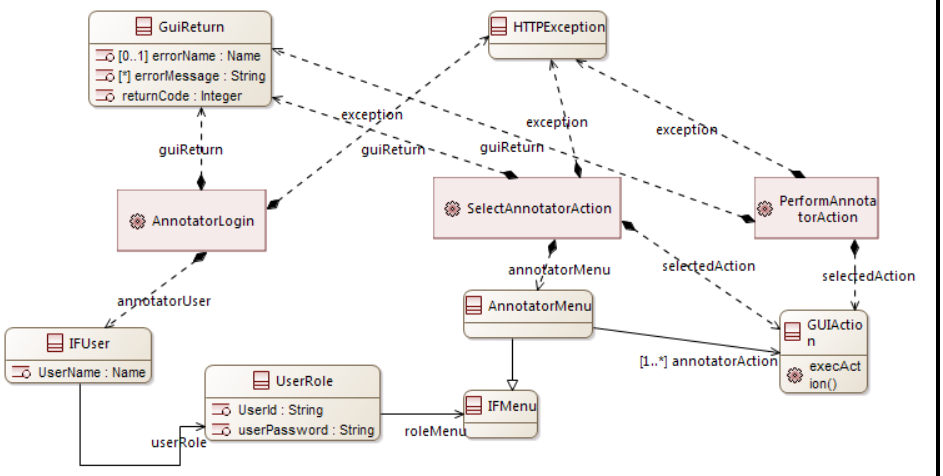
Interface ID:	**Annotations GUI**
Interface Name:	AnnotationsGUI SelectAnnotatorAction(IN selectedAction:GUIAction, IN annotatorMenu:AnnotatorMenu) : returns guiReturn:GuiReturn throws exception:HTTPException PerformAnnotatorAction(IN selectedAction:GUIAction) : returns guiReturn:GuiReturn throws exception:HTTPException 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:	Inexisting account, account not authorised
Notes and Issues:	Standard, open source rdfyzer software to be used for implementation

Table 5: Import External Resources interface

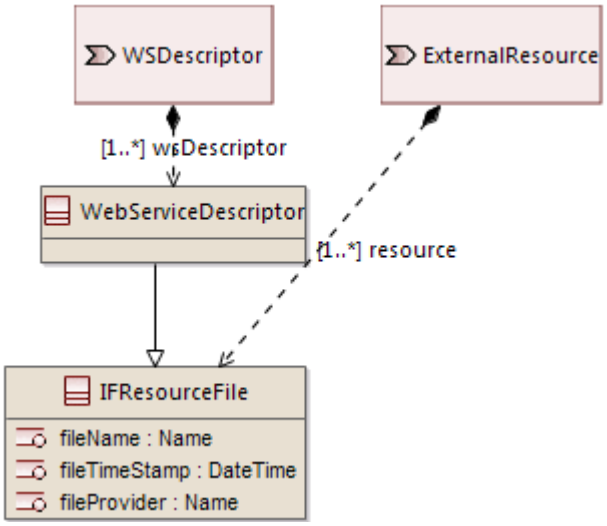
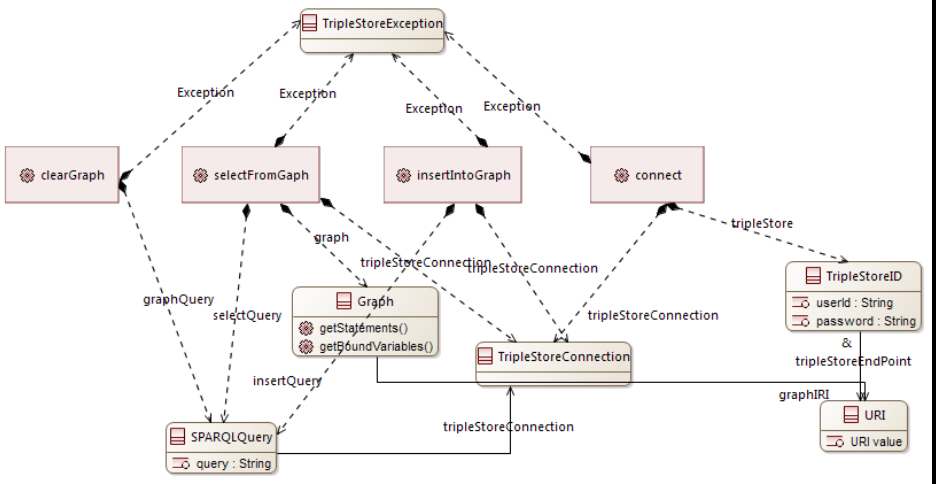
Interface ID:	**Import External Resources**
Interface Name:	
Purpose of the Interface	Import external resources, e.g. data, web service descriptors, for semantic annotation and conversion into semantic graph
Requestor:	Resource Provider (external Actor)
Provider	Resource RDFyer
Description:	Import external 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

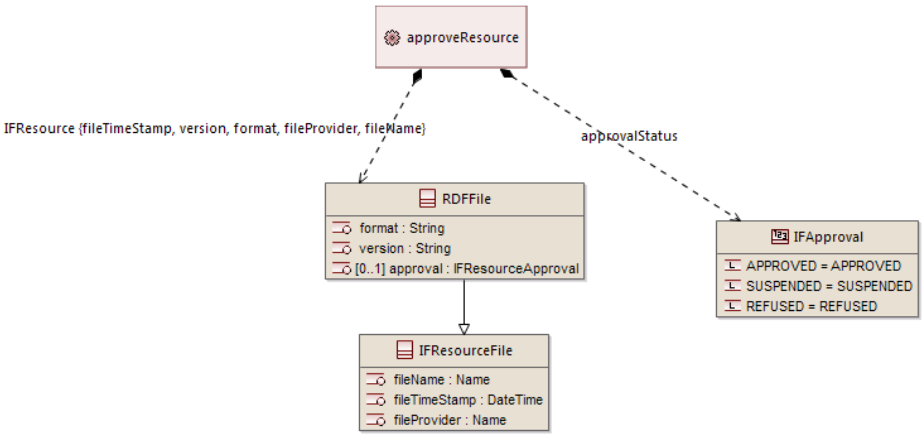
Interface ID:	**Approve Graph**
Interface Name:	
Purpose of the Interface	Submit resource to approval workflow process
Requestor:	Resource RDFyer
Provider	Workflow Manager
Description:	Submit resource to approval workflow process
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
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

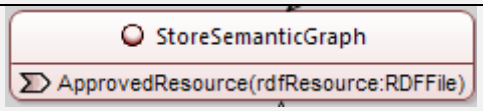
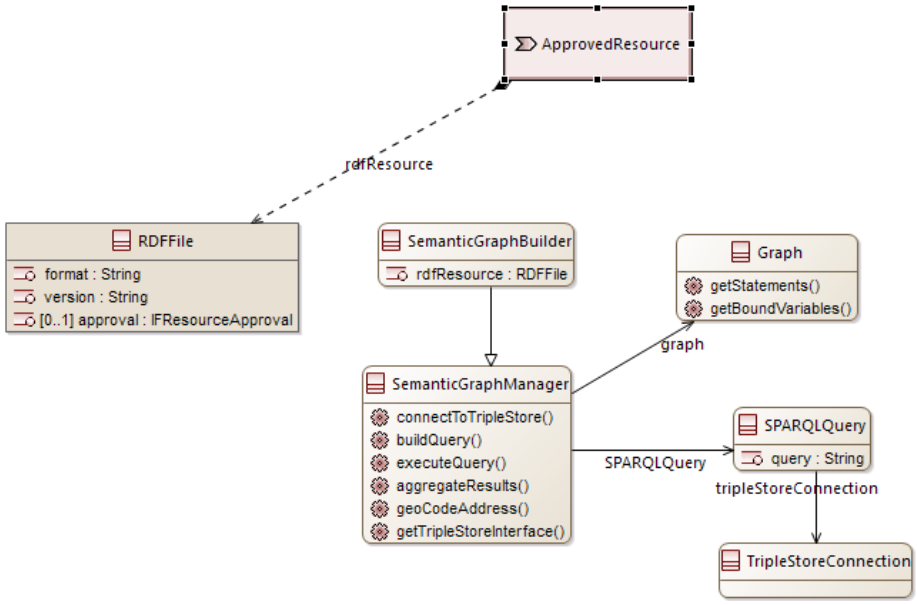
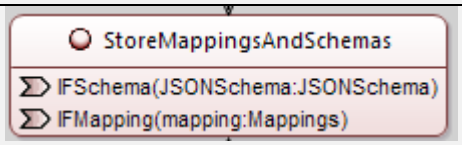
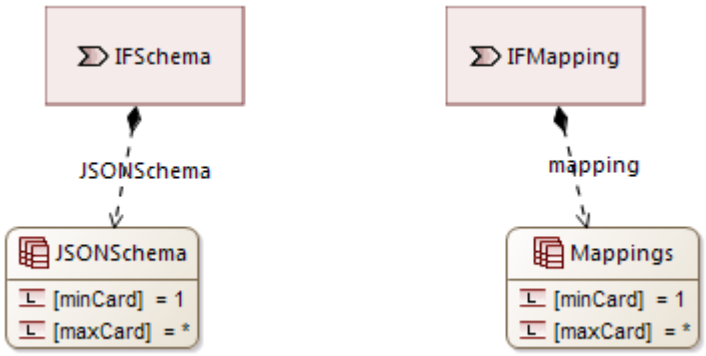
Interface ID:	**Store Semantic Graph**
Interface Name:	
Purpose of the Interface	Store approved RDF resource as semantic graph in triple store
Requestor:	WorkFlow Manager
Provider	Resource RDFyer
Description:	Store approved RDF resource as semantic graph in triple store
Impact to CREL	Complete
Impact 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

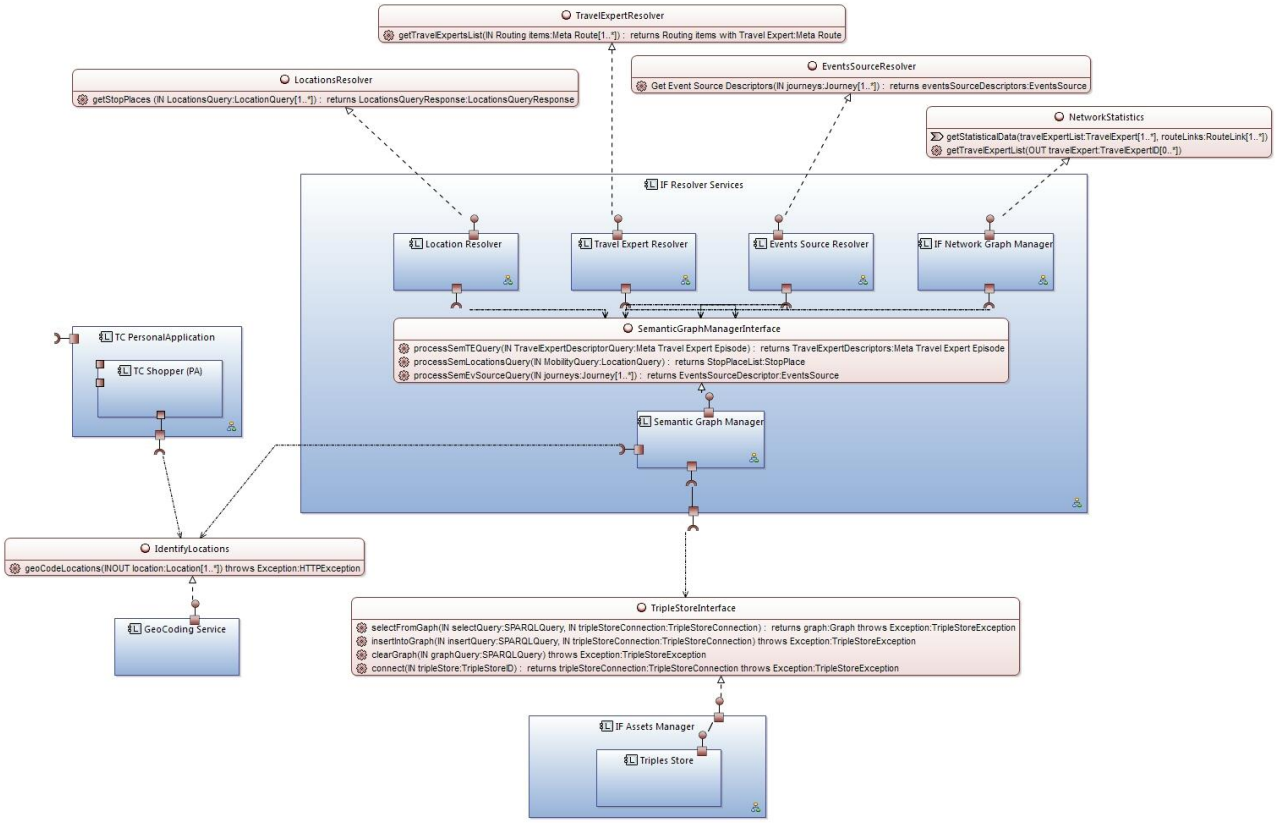


Figure 23: IF Resolver Services interfaces

7.2.1 External Interfaces

Table 10: IdentifyLocations interface

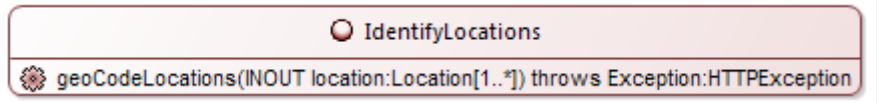
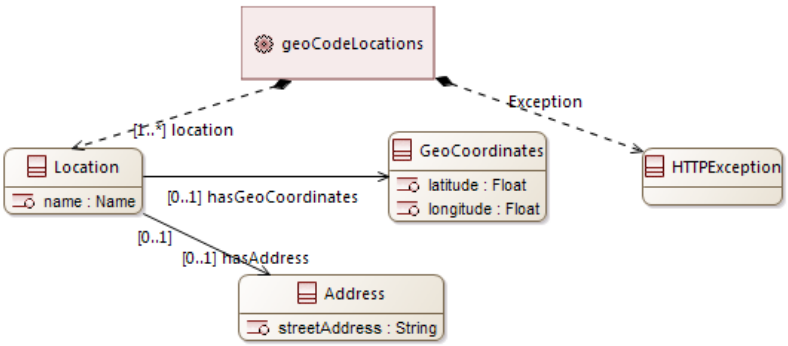
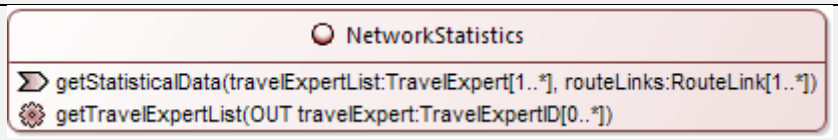
Interface ID:	**IdentifyLocations**
Interface Name:	
Purpose of the Interface	Provide list of Locations with geo coordinates
Requestor:	Travel Companion (WP5 component)
Provider	GeoCoding service
Description:	Provide list of Locations with geo coordinates
Impact to CREL	Complete
Impact to AREL	Complete
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

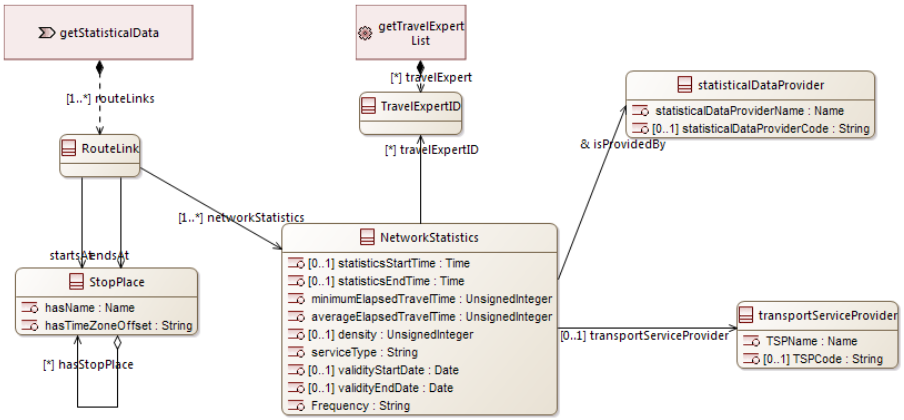
Exchange Items	
Exceptions:	Invalid request
Notes and Issues:	

Table 12: Locations Resolver interface

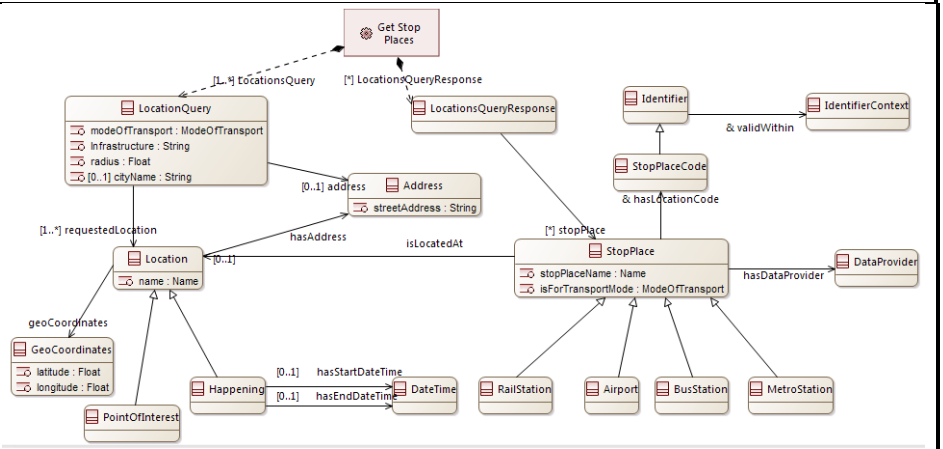
Interface ID:	**Locations Resolver**
Interface Name:	LocationsResolver Get Stop Places (IN LocationsQuery:LocationQuery[1..*]) : returns LocationsQueryResponse:LocationsQueryResponse
Purpose of the Interface	Provide list of Stop Place within a given radius of a Location
Requestor:	Travel Shopping (WP2 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

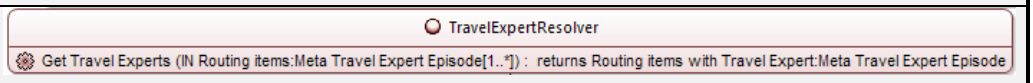
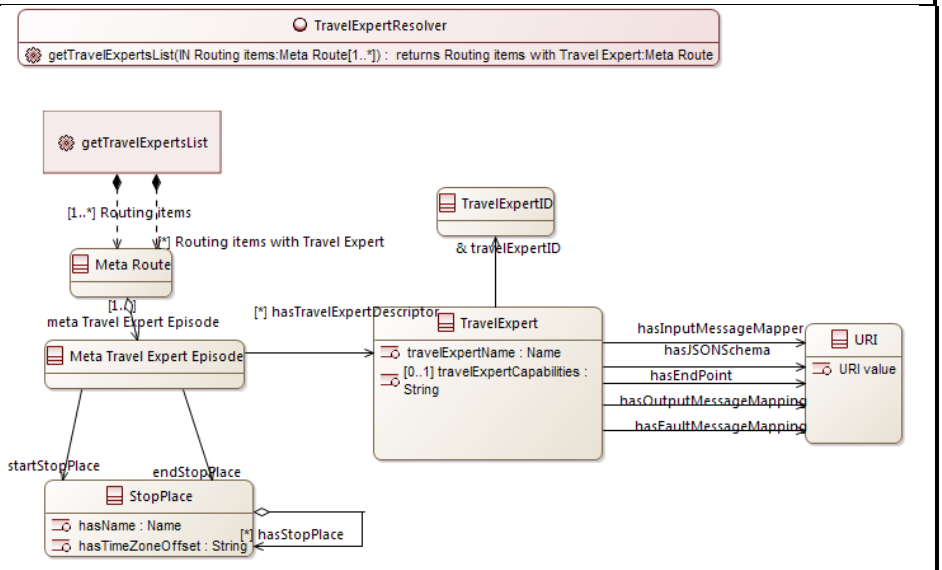
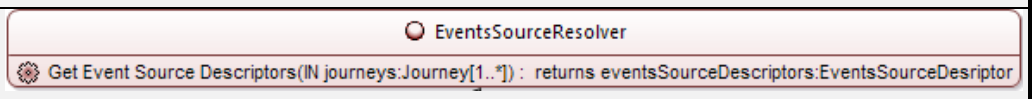
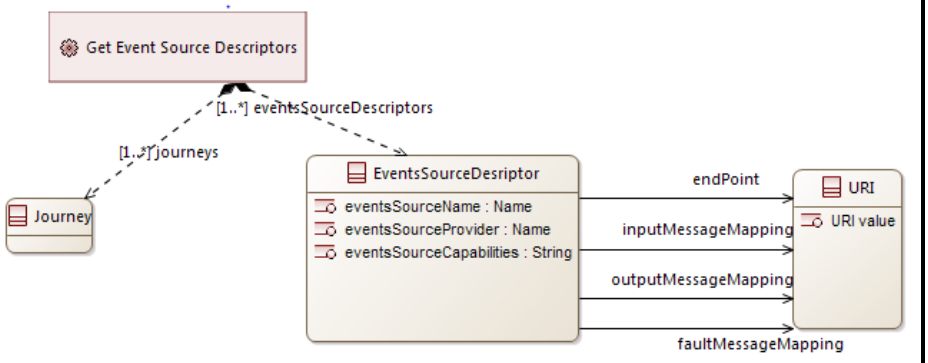
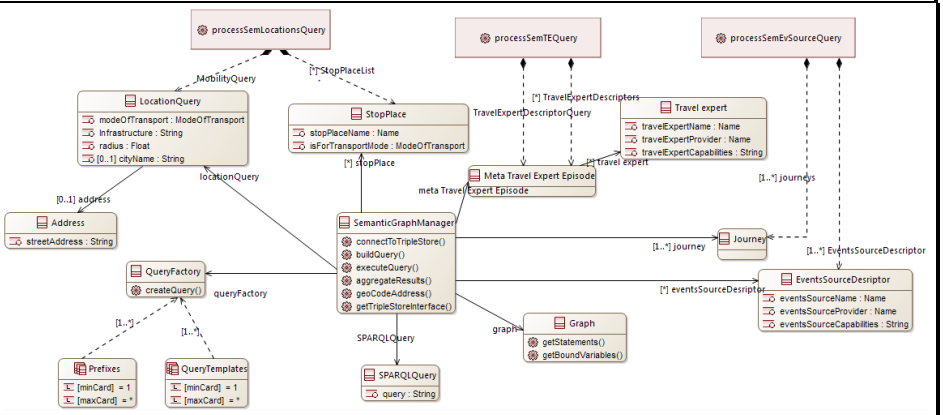
Interface ID:	**Travel Expert Resolver**
Interface Name:	
Purpose of the Interface	Associate Meta Travel Expert Episode with corresponding Travel Expert service descriptors
Requestor:	Travel Shopping (WP2 Components)
Provider	Travel Expert Resolver
Description:	Associate Meta Travel Expert Episode with 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:	Additional notes, issues and comments

Table 14: Events Source Resolver interface

Interface ID:	**Events Source Resolver**
Interface Name:	
Purpose of the Interface	Provide descriptors of services providing tracking information or a set of journeys (event sources)
Requestor:	Events Listening (WP4 component)
Provider	Events Source Resolver
Description:	Provide descriptors of services providing tracking information or a set of journeys (event sources)
Impact to CREL	Complete
Impact to AREL	Complete
Impact to FREL	Complete
Preconditions:	Valid input journeys
Postconditions:	Zero, one or more service descriptors of sources providing tracking events for journey
Exchange Items	
Exceptions:	Invalid request
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

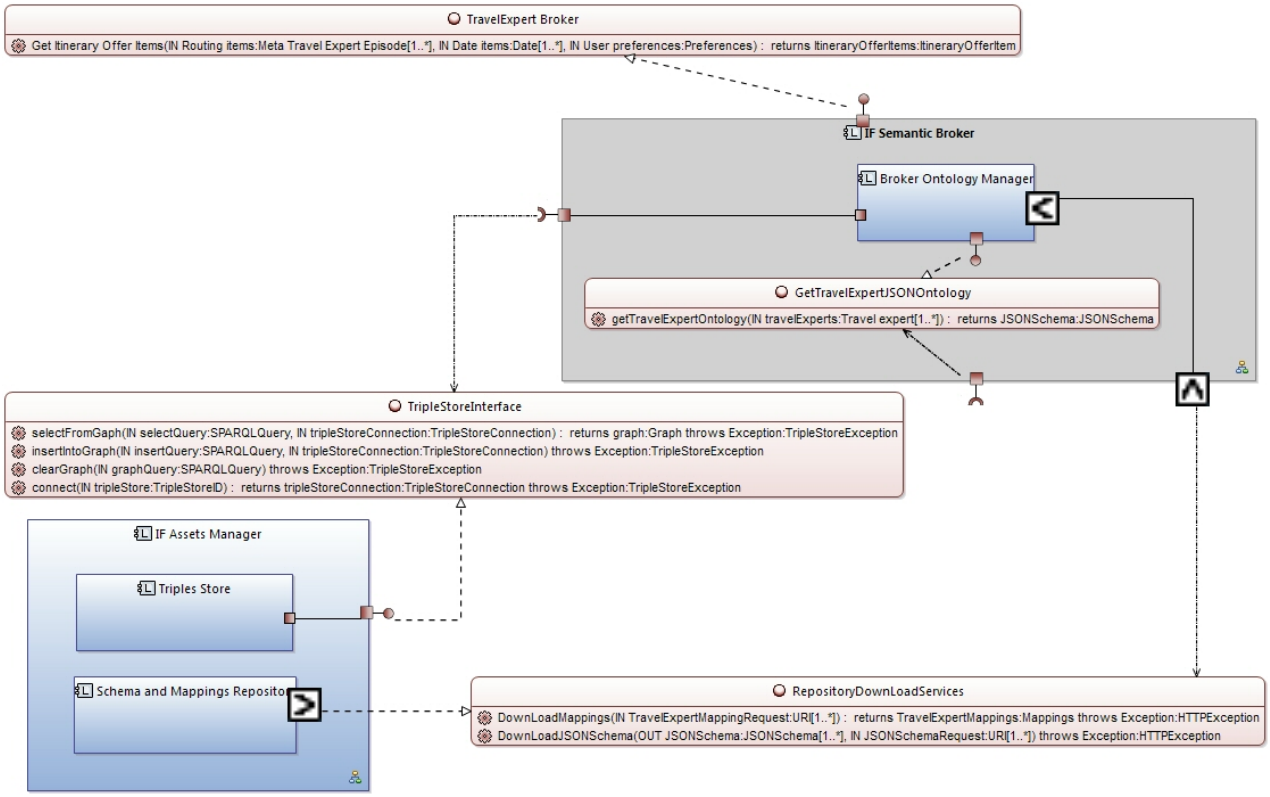


Figure 24: IF Semantic Broker interfaces

7.3.1 External Interfaces

Table 16: Travel Expert Broker interface

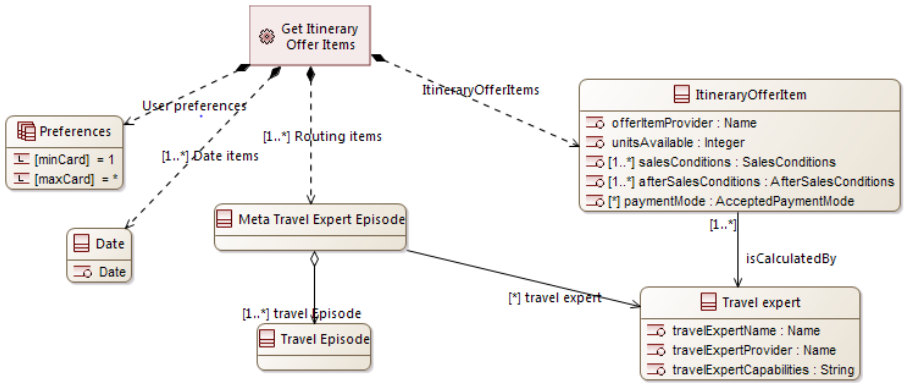
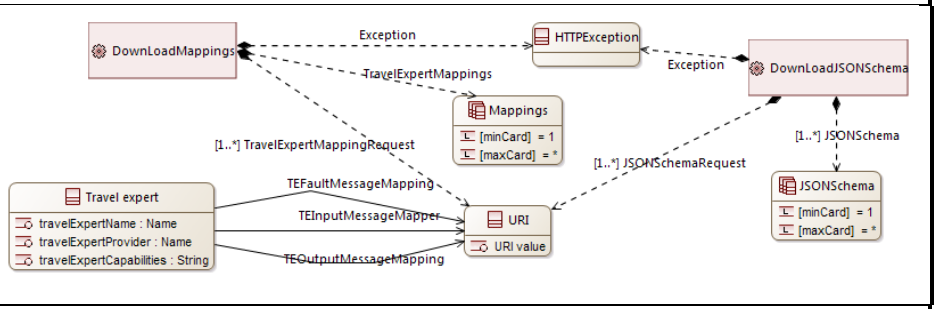
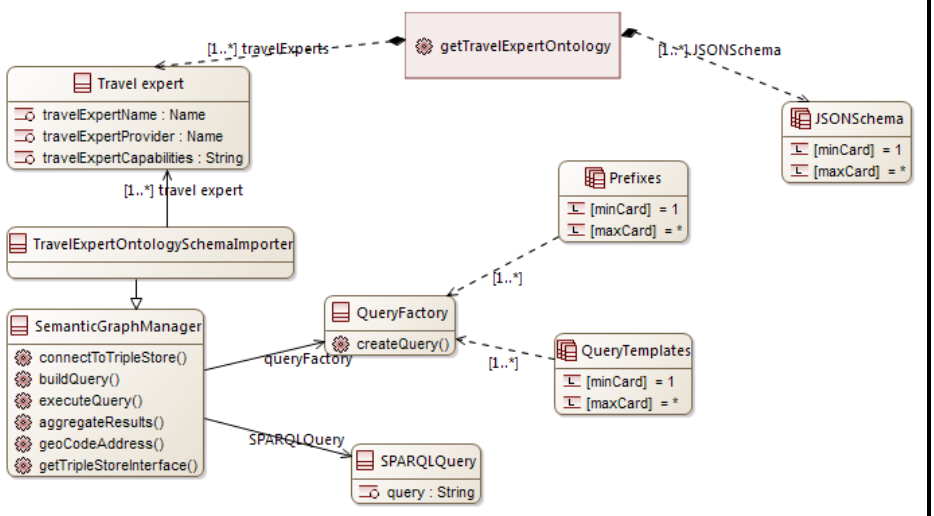
Interface ID:	**Travel Expert Broker**
Interface Name:	TravelExpert Broker Get Itinerary Offer Items(IN Routing items:Meta Travel Expert Episode[1..*], IN Date items:Date[1..*], IN User preferences:Preferences) : returns ItineraryOfferItems:ItineraryOfferItem
Purpose of the Interface	Provide a proxy to external, heterogeneous Travel Experts
Requestor:	Offer Builder (WP2 Component)
Provider	Semantic Broker
Description:	Return ItinerayOfferItems 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: Repository Download Services interface

Interface ID:	**Repository Download Services**
Interface Name:	RepositoryDownLoadServices DownLoadMappings(IN TravelExpertMappingRequest:URI[1..*]) : returns TravelExpertMappings:Mappings throws Exception:HTTPException DownLoadJSONSchema(OUT JSONSchema:JSONSchema[1..*], IN JSONSchemaRequest:URI[1..*]) throws Exception:HTTPException
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 18: 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

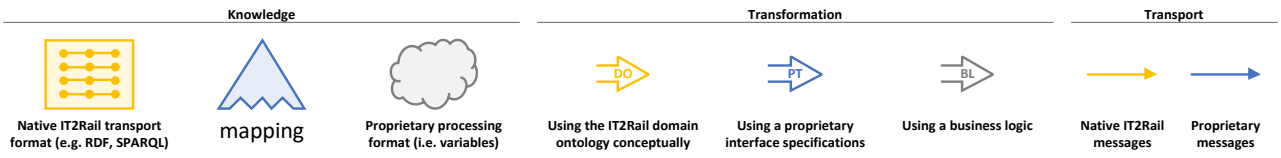
The design of the Interoperability Framework is based on the provision of a set of semantic interoperability services that can be deployed in multiple configurations, described below, and that do not mandate a specific set of communication protocols or frameworks, leaving the choice of deployment strategies 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:



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

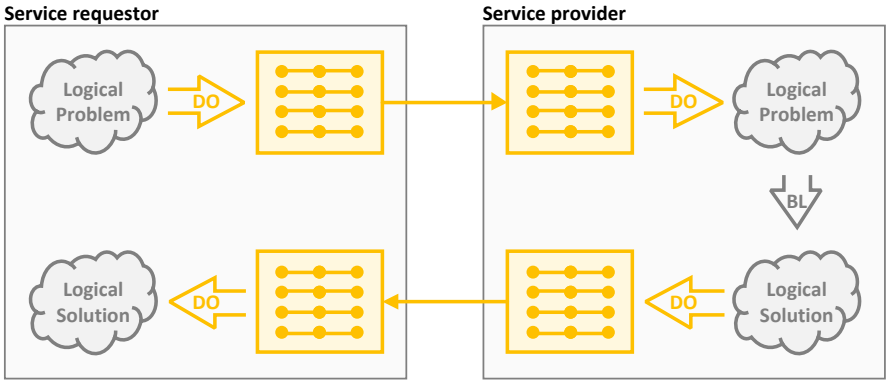


Figure 25 Native IT2Rail exchange scenario

This Scenario depicts the peer to peer communication between an arbitrary service requestor and a services 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

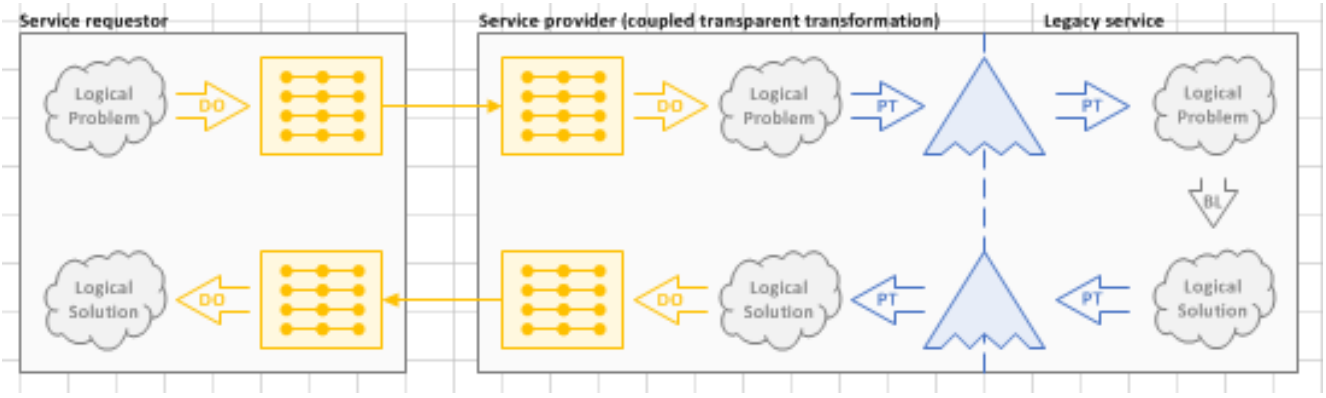


Figure 26 Wrapped legacy service exchange scenario

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

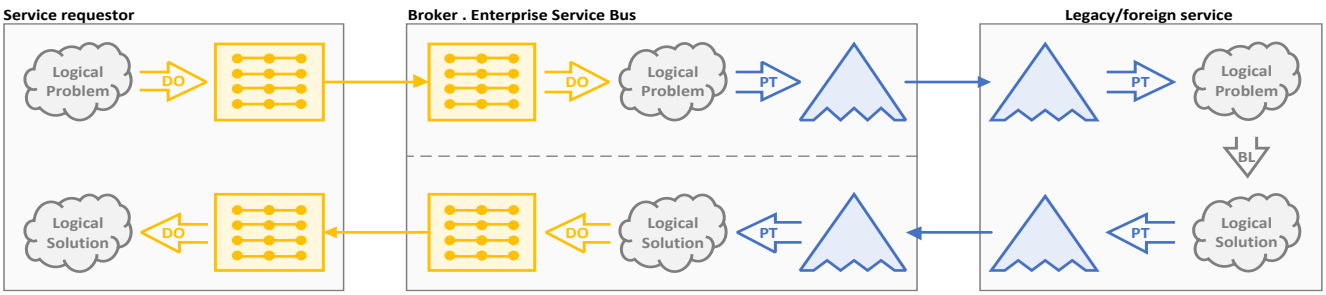


Figure 27 - Enterprise Service Bus / Broker scenario

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.

In the IT2Rail demonstration this scenario may be implemented, using the SOFIA2 platform made available by IT2Rail partner INDRA, along one or both of the others in to test the deployment independence of the Interoperability Framework services and to obtain comparative performance evidence.

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 AREL IMPLEMENTATION

This chapter documents the implementation of the “additional release” (AREL) 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 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 AREL release is built and installed in the IT2Rail project’s maven repository for use in higher level development. The following fragment from the framework’s pom file lists the dependencies:

```
<groupId>it2rail.rdf.framework</groupId>
```

```
 <artifactId>it2rail-rdf-framework</artifactId>
 <version>1.0-AREL</version>

<name>WP1 rdf framework for IT2Rail</name>
<description>A POJO/RDF semantic mapping and persistence framework IT2Rail project</description>
<organization>
        <name>It2Rail project Work Package 1 Interoperability Framework</name>
</organization>
<properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
 </properties>

<dependencies>
                <dependency>
                        <groupId>org.eclipse.rdf4j</groupId>
                        <artifactId>rdf4j-runtime</artifactId>
                        <version>2.0</version>
                         <type>pom</type>

                </dependency>
        <!-- https://mvnrepository.com/artifact/com.google.inject.extensions/guice-multibindings -->
                 <dependency>
                        <groupId>org.ow2.spec.ee</groupId>
                        <artifactId>ow2-jpa-1.0-spec</artifactId>
                        <version>1.0.12</version>
                </dependency>

                <dependency>
                        <groupId>com.google.inject.extensions</groupId>
                        <artifactId>guice-multibindings</artifactId>
                        <version>4.0</version>
                </dependency>
                <!-- https://mvnrepository.com/artifact/com.google.inject.extensions/guice-assistedinject -->
                <dependency>
                        <groupId>com.google.inject.extensions</groupId>
                        <artifactId>guice-assistedinject</artifactId>
                        <version>4.0</version>
                </dependency>
                <!-- https://mvnrepository.com/artifact/org.javassist/javassist -->
                <dependency>
                        <groupId>org.javassist</groupId>
                        <artifactId>javassist</artifactId>
                        <version>3.17.1-GA</version>
                </dependency>
                <dependency>
                        <groupId>org.it2rail.cp-rdf4j-utils</groupId>
                         <artifactId>it2rail.cp-rdf4j-utils</artifactId>
                        <version>1.0.0</version>
                 </dependency>
                 <dependency>
                   <groupId>com.complexible.common.utils</groupId>
                         <artifactId>cp-common-utils</artifactId>
                         <version>5.0.1</version>
                 </dependency>
                <!-- https://mvnrepository.com/artifact/org.geonames/geonames -->
                <dependency>
                        <groupId>com.google.guava</groupId>
                        <artifactId>guava</artifactId>
                        <version>19.0</version>
                </dependency>
```

```
        <!-- https://mvnrepository.com/artifact/commons-beanutils/commons-beanutils -->
        <dependency>
                <groupId>commons-beanutils</groupId>
                <artifactId>commons-beanutils</artifactId>
         <version>1.9.2</version>
         </dependency>
       <!-- https://mvnrepository.com/artifact/junit/junit -->
        <dependency>
                 <groupId>junit</groupId>
                 <artifactId>junit</artifactId>
                 <version>4.12</version>
     </dependency>
    <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->
        <dependency>
                <groupId>org.slf4j</groupId>
                <artifactId>slf4j-api</artifactId>
                <version>1.7.21</version>
        </dependency>
   <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-simple -->
        <dependency>
                 <groupId>org.slf4j</groupId>
                 <artifactId>slf4j-simple</artifactId>
                 <version>1.7.21</version>
        </dependency>
</dependencies>
```

9.1.1 Empire Configuration

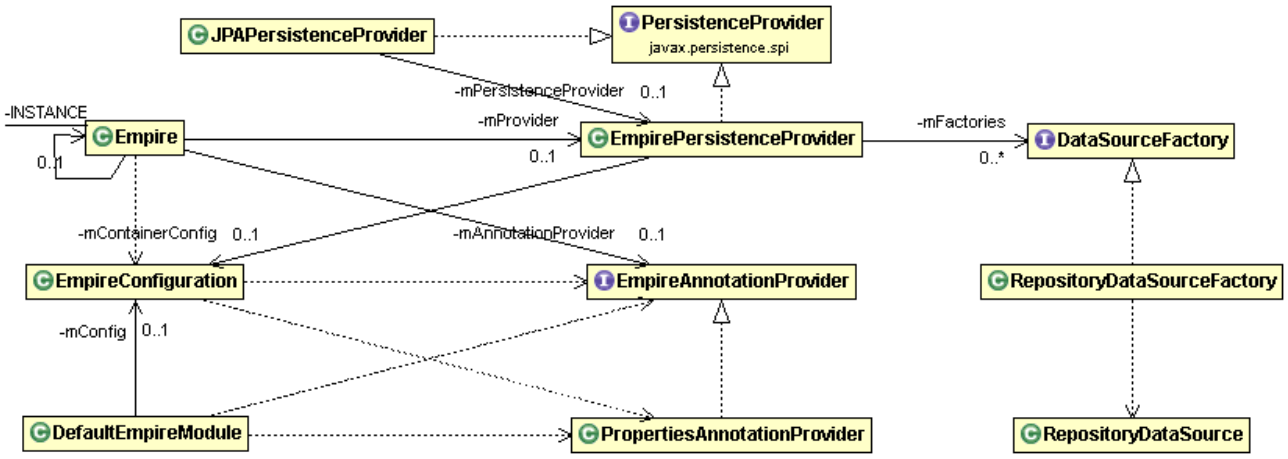


Figure 28: Empire Configuration - diagram showing relationships among Java classes

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.

Concrete implementations of the EmpireAnnotationProvider and EmpirePersistenceProviders are injected at initialisation time through the Guice framework2, 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)

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>
```

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

9.1.2 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 providers for relational data bases.

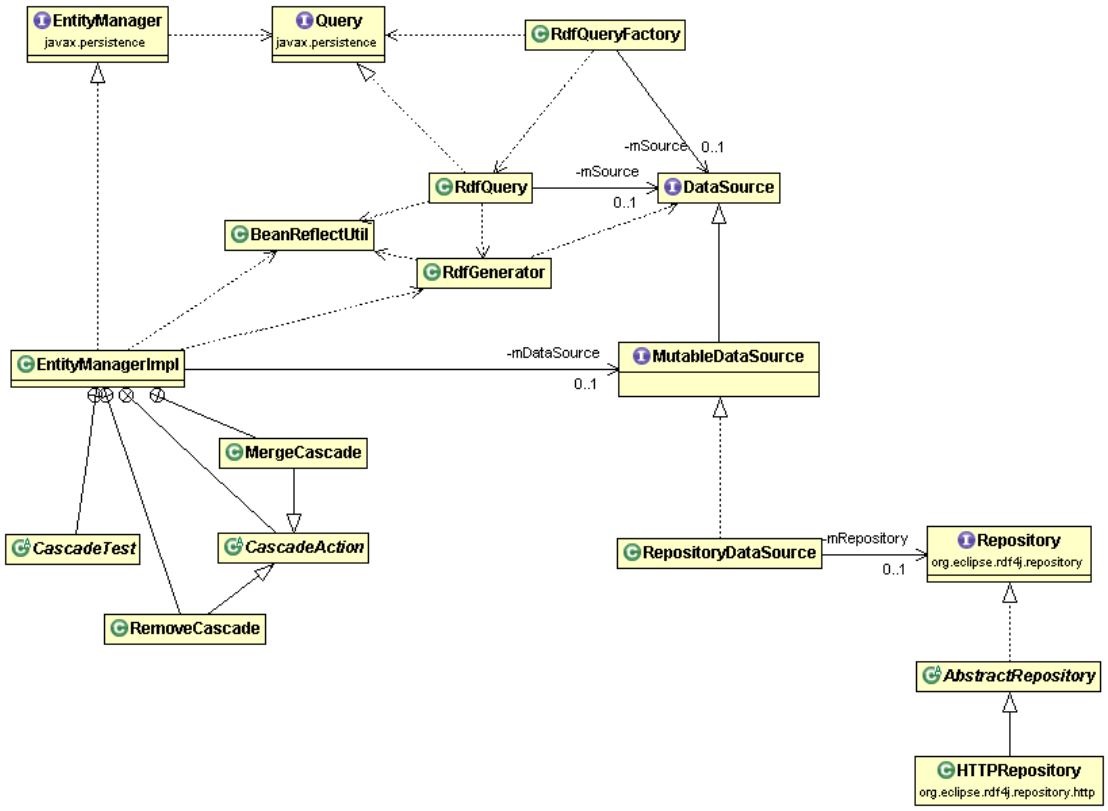


Figure 29: Empire Manager - diagram showing relationships among Java classes

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.

9.1.3 RDF Generator

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

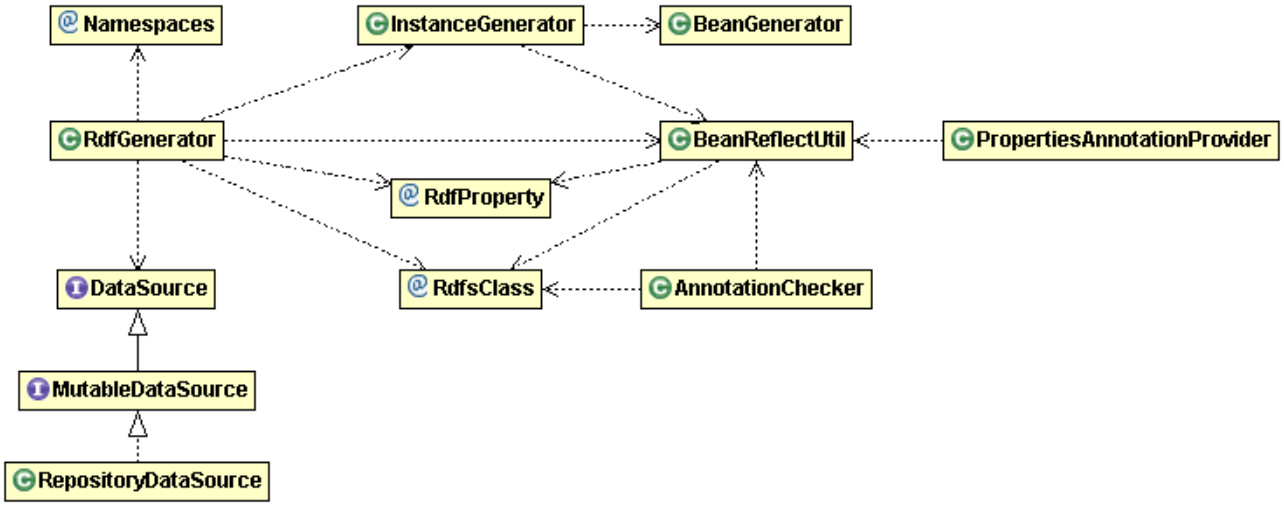


Figure 30: RDF Generator - diagram showing relationships among Java classes

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

9.2 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 AREL release is built and installed in the IT2Rail project's maven repository for use in packaged resolver development. The following fragment from the framework's pom file lists the dependencies:

```
<groupId>it2rail.if.semanticgraphmanager</groupId>
<artifactId>it2rail-semanticgraphmanager</artifactId>
<version>1.0-AREL</version>
<name>WP1 Semantic Graph Manager for IT2Rail</name>
<description>IT2Rail Interoperability Framework   Semantic Graph Manager</description>
<organization>
        <name>It2Rail project Work Package 1 Interoperability Framework</name>
</organization>
<properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
</properties>

<dependencies>
                <dependency>
                        <groupId>it2rail.rdf.framework</groupId>
                        <artifactId>it2rail-rdf-framework</artifactId>
                        <version>1.0-AREL</version>
                </dependency>
                <!-- https://mvnrepository.com/artifact/org.geonames/geonames -->
                <dependency>
                        <groupId>org.geonames</groupId>
                        <artifactId>geonames</artifactId>
                        <version>1.0</version>
                        <scope>test</scope>
                </dependency>
</dependencies>
```

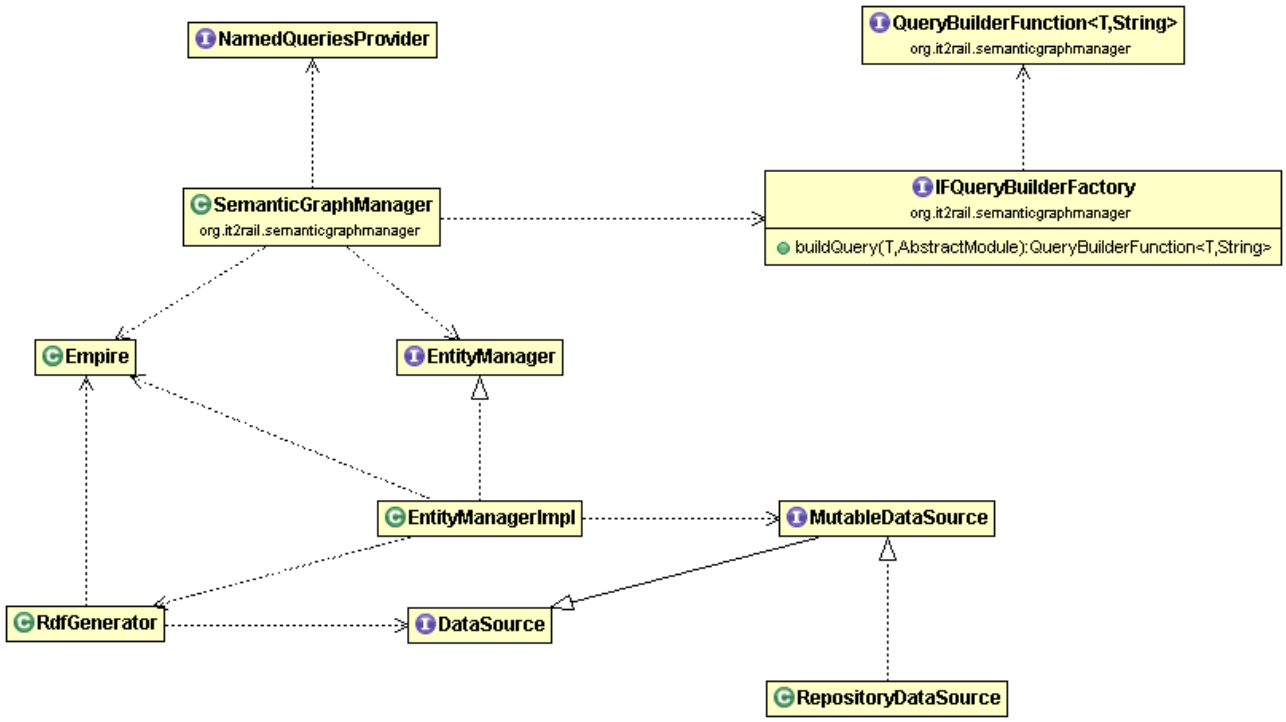


Figure 31: Semantic Graph Manager - diagram showing relationships Java classes

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, ect 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:

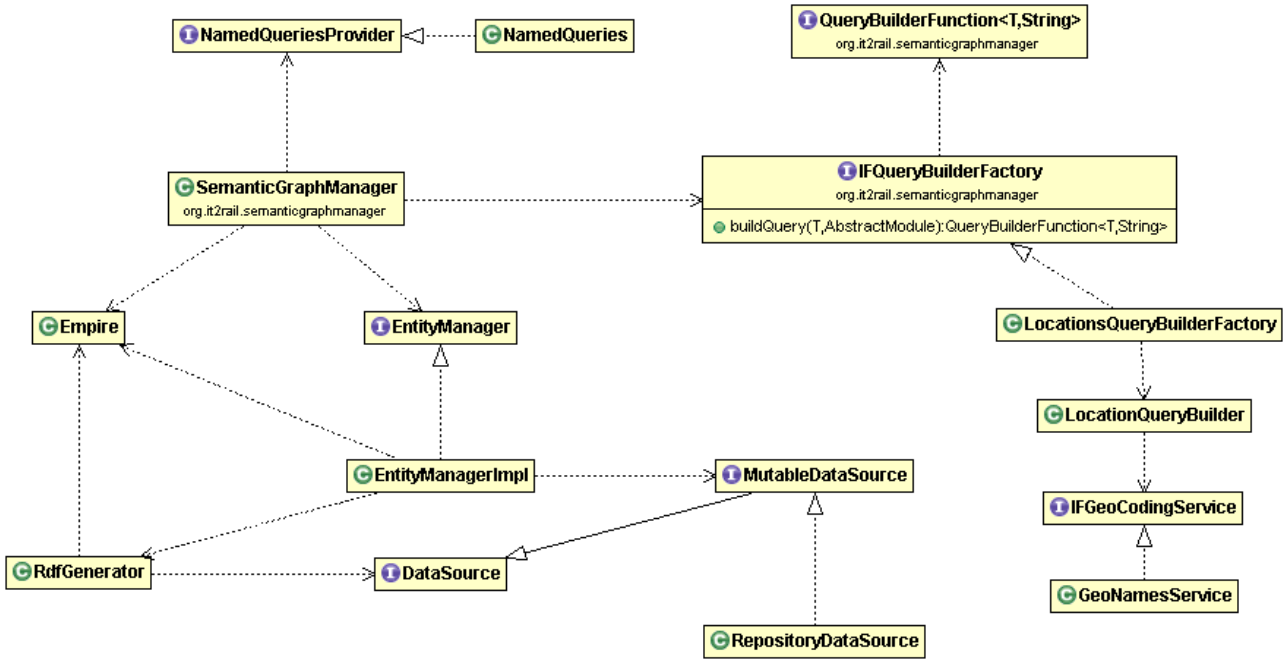


Figure 32: Example of Location Resolver service instantiation

9.3 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 AREL:

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 bindings using a standard jaxb maven plugin.
4. The resulting jaxb bindings have been annotated with the terms from the ontology through the it2rail rdf framework mechanism establishing semantic correspondence between the jaxb bindings 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 fragment of the Location Resolver maven pom lists the dependencies:

```
<groupId>it2rail.if.locationsresolver</groupId>
 <artifactId>it2rail-locations-resolver</artifactId>
 <version>1.0-AREL</version>
 <name>WP1 Locations Resolver for IT2Rail</name>
 <description>IT2Rail Interoperability Framework  Locations Resolver</description>
<organization>
                <name>It2Rail project Work Package 1 Interoperability Framework</name>
</organization>
 <properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
</properties>

<dependencies>
                 <dependency>
                                <groupId>it2rail.if.semanticgraphmanager</groupId>
                                <artifactId>it2rail-semanticgraphmanager</artifactId>
                                <version>1.0-AREL</version>
                 </dependency>
                <dependency>
                                <groupId>it2rail.if.ontology.infrastructure</groupId>
                                <artifactId>it2rail-ontology-infrastructure</artifactId>
                                 <version>1.0-AREL</version>
```

```
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.geonames/geonames -->
        <dependency>
            <groupId>org.geonames</groupId>
            <artifactId>geonames</artifactId>
            <version>1.0</version>
        </dependency>
</dependencies>
```

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 geocoding service in the LocationResolverQueryBuilder allows for choosing any one of the multiple such free or commercial services available from multiple providers.

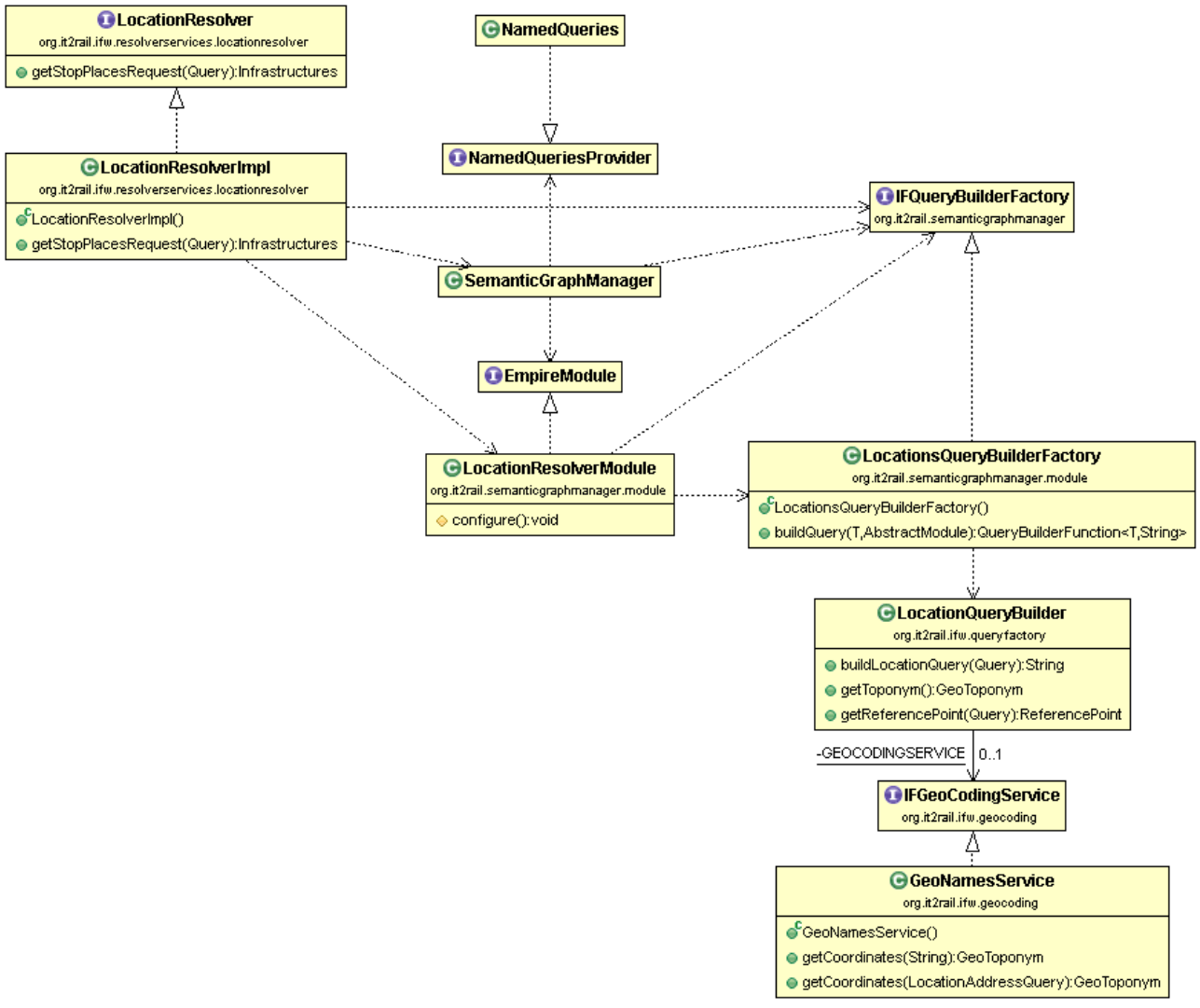


Figure 33: Location Resolver service instantiated by supplying the Semantic Graph Manager with the specific LocationResolver Guice module.

9.4 TRAVEL EXPERT RESOLVER

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, as in the following fragment:

```
<groupId>it2rail.if.travelexpertresolver</groupId>
<artifactId>it2rail-travelexpert-resolver</artifactId>
<version>1.0-AREL</version>
<name>WP1 Travel Expert Resolver for IT2Rail</name>
<description>IT2Rail Interoperability Framework  Travel Expert Resolver</description>
<organization>
                <name>It2Rail project Work Package 1 Interoperability Framework</name>
</organization>
 <properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
</properties>

<dependencies>
                <dependency>
                                <groupId>it2rail.if.semanticgraphmanager</groupId>
                                <artifactId>it2rail-semanticgraphmanager</artifactId>
                                 <version>1.0-AREL</version>
                </dependency>
                <dependency>
                                <groupId>it2rail.if.ontology.infrastructure</groupId>
                                <artifactId>it2rail-ontology-infrastructure</artifactId>
                                <version>1.0-AREL</version>
                </dependency>
</dependencies>
```

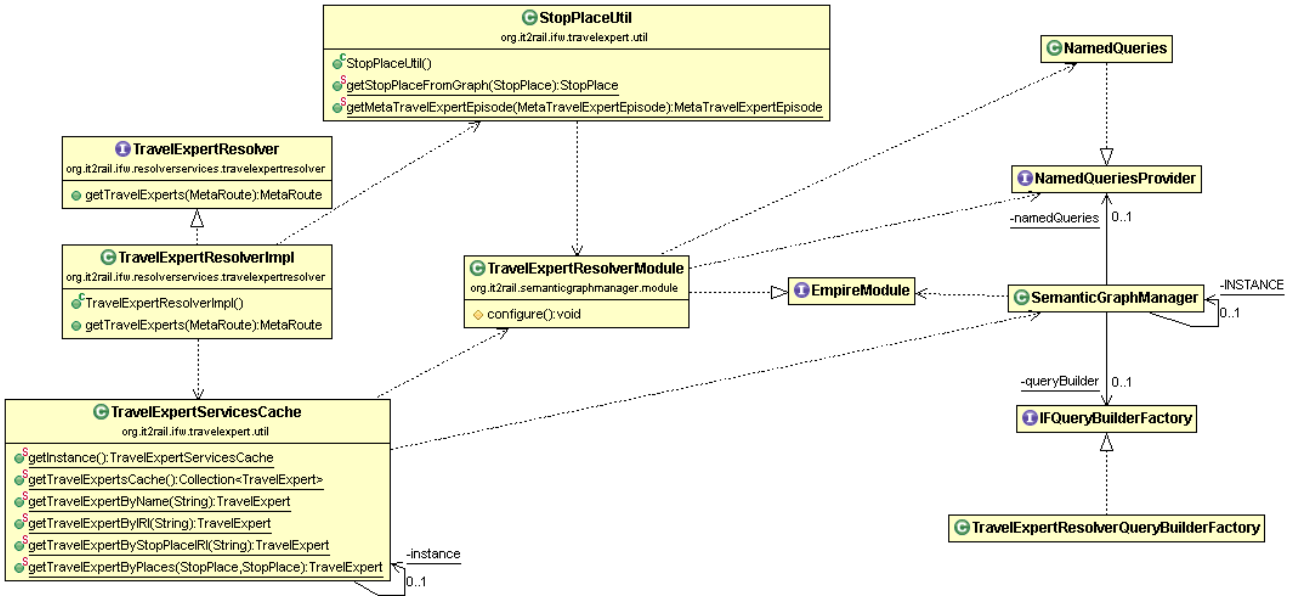


Figure 34: Example of Travel Expert Resolver service instantiation

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 AREL implementation to emulate the presence of Travel Experts that were not available at this stage of the project.

End-of-document