Contract No. H2020 – 826172

SEMANTICS FOR PERFORMANT AND SCALABLE INTEROPERABILITY OF MULTIMODAL TRANSPORT

D3.1 - Analysis of the state-of-the-art and best practices in architecting systems processing semantic data

Due date of deliverable: 30/04/2019

Actual submission date: 20/05/2019

Leader/Responsible of this Deliverable: POLIMI

Reviewed: Y

Document status		
Revision	**Date**	**Description**
0.7	19/04/2019	First complete draft
0.9	30/4/2019	Revised version of complete draft
0.99	07/05/2019	First version
1.0	09/05/2019	Consolidated version for TMC approval
1.1	17/05/2019	Final Version after TMC approval and Quality check

Project funded from the European Union's Horizon 2020 research and innovation programme		
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: 01/12/2018 Duration: 25 months

EXECUTIVE SUMMARY

This deliverable analyzes the state of the art that is relevant for the future developments (in terms of improved architecture) of the Shift2Rail Interoperability Framework. The deliverable first provides an overview of the Interoperability Framework, to foster a common understanding of its goals and main functions. Then, it studies a few architectural approaches on which the new reference architecture of the Interoperability Framework that will be defined in the course of the SPRINT project could be based. It also surveys how the interoperability problem has been tackled in other domains that have characteristics similar to the transportation domain. Then, it provides an assessment of the developments related to the Interoperability Framework carried out in past Shift2Rail projects. Finally, it concludes with a discussion pointing out the approaches and techniques that seem the most suitable for the future developments of the Interoperability Framework.

ABBREVIATIONS AND ACRONYMS

Abbreviation	Description
API	Application Programming Interface
BPMN	Business Process Model and Notation
EIP	Enterprise Integration Pattern
EU	European Union
GA	Grant Agreement
GTFS	General Transit Feed Specification
H2020	Horizon 2020 framework programme
HATEOAS	Hypermedia as the Engine of Application State
IaaS	Infrastructure as a Service
IF	Interoperability Framework
IoT	Internet of Things
JAR	Java ARchive
JSON	JavaScript Object Notation
JU	Shift2Rail Joint Undertaking
LSM	Linked Stream Middleware
OS	Operating System
P2P	Peer to Peer
PaaS	Platform as a Service
RASIC	Reference Architecture for Semantically Interoperable Clouds
RDF	Resource Description Framework
REST	Representational State Transfer
S2R	Shift2Rail
SaaS	Software as a Service
SSN	Semantic Sensor Network
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier

VM	Virtual Machine
XML	eXtensible Markup Language
WAR	Web Application Resource
W3C	World Wide Web Consortium
WSDL	Web Services Description Language

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This deliverable discusses the state of the art that is relevant for the definition and the development of the reference architecture for the Shift2Rail (S2R) Interoperability Framework (IF). The analysis carried out in this deliverable will be used in Task 3.4 “Design and comparison of alternative architectural solutions for the IF” of the SPRINT project to identify the best architectural solutions to build an IF meeting the requirements that will be identified in Task 3.2 “Elicitation of performance and scalability requirements for the IF”.

To provide a common understanding of what are the purposes and main functions of the S2R IF, Section 2 first gives an overview of the IF. This overview is not intended to provide an early definition of the architecture of the IF. Rather, its goal is to summarize the current conceptual status of the IF: what it aims to be, what are its main elements and the very high-level functions they provide.

Then, sections 3, 4 and 5 carry out the main analysis of the relevant state of the art, which follows different directions.

Section 3 presents the architectural approaches and patterns that are the best candidates to be the basis for the future reference architecture of the IF. In particular, it focuses on the service-oriented approach (Section 3.1), on cloud-based architectures (Section 3.2), and on peer-to-peer systems (Section 3.3).

Section 4 analyzes how the interoperability problem has been tackled in other domains that have similar characteristics as the transport domain (heterogeneity of standards, high number of stakeholders, etc.); in particular, it focuses on the Internet of Things and Cloud systems domains.

Section 5, instead, focuses on the Shift2Rail context, and assesses the results of a pair of previous projects, IT2Rail [1] and ST4RT [2] that have developed technology that is part of the S2R IF.

Finally, Section 6 summarizes the findings of the analysis of the previous sections, and highlights which are the most promising approaches for the future developments of the S2R IF.

2. OVERVIEW OF THE INTEROPERABILITY FRAMEWORK

The provision of truly customer-centric mobility services across the Single European Transport Area requires advanced ICT applications that can discover, access and combine mobility solutions from multiple Service Provider Companies. From a digital system engineering point of view, these applications must be able to coordinate the execution of complex computational tasks that are inherently distributed on multiple heterogeneous “nodes” of an open network with no centralized control.

The IF is designed to provide these applications with a semantically consistent abstraction of ICT resources offered by Service Provider Companies, insulating them from the “mechanics” of operating remotely over networks and across multiple communications protocols and/or data formats. Conversely, it is designed to allow Service Provider Companies to leverage without expensive adaptations their own native ICT computing environment and resources as elements of an end-to-end intermodal mobility solution, insulating them from the specifics of the customer front-end applications.

As such, the Interoperability Framework provides “distribution transparencies”, as defined by the “ITU-T Rec. X.903 | ISO/IEC 10746-3: Architecture” standard for open distributed processing [3], enabling complexities associated with system distribution to be hidden from applications where they are irrelevant to their purpose. These include:

- access transparency, which masks differences of data representation and invocation mechanisms for services between systems;
- location transparency, which masks the need for an application to have information about location in order to invoke a service;
- relocation transparency, which masks the relocation of a service from applications using it;
- replication transparency, which masks the fact that multiple copies of a service may be provided in order to provide reliability and availability.

Access and location transparencies are achieved by leveraging semantic interoperability principles and technologies: knowledge about the domain problem, which is typically held by human analysts and programmers, is formalised in a set of logical statements, or "axioms", written in a standard computer language available for machine processing. Human knowledge is thus transferred to machines and shared by them. Any particular representation of concepts and relationships in a specific data structure is associated, through a process of annotation, with its interpretation in terms of the domain problem. Machines can therefore discover and leverage equivalence relationship between different data formats with common meaning, and automate, therefore, the translation across these formats. Automated computer logical inference replaces human programming of software to operate on different – but equivalent – data formats however they may be exchanged. Semantically-annotated data is furthermore linked to constitute a shared semantic graph, or "web of transport data", whose physical distribution across networked machines is invisible to consuming services.

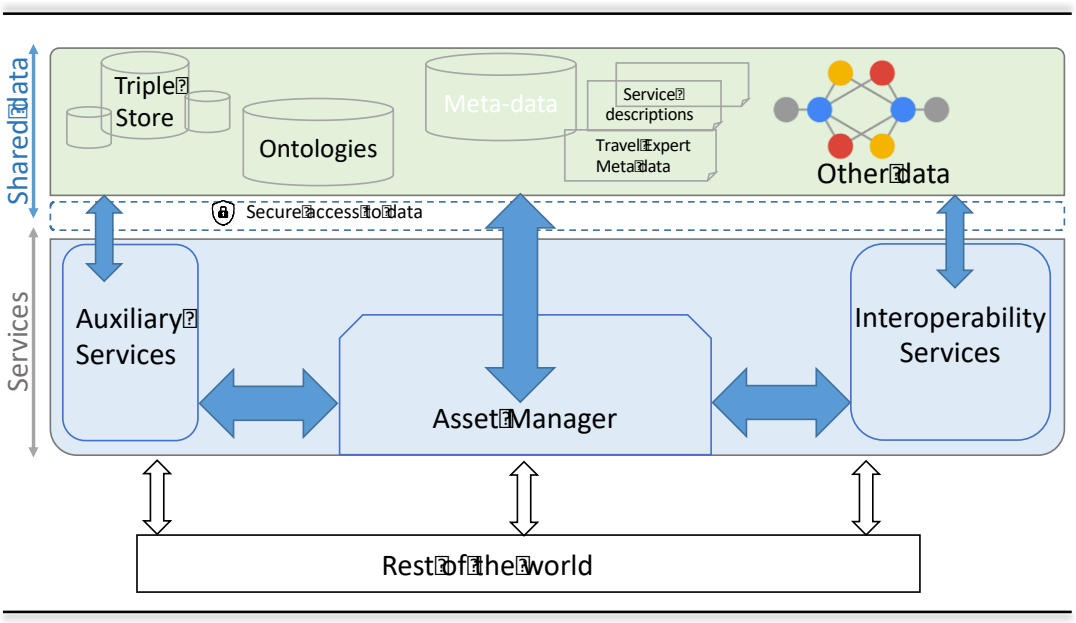


Figure 1 Generic Architecture of the Interoperability Framework

Figure 1 depicts the generic architecture and scope of IF which spans two logical planes called Data Layer and Service Layer. A layer in this architecture is a way of grouping a related set of functionalities. In other words, there is not necessarily a predefined relation between different components of different layers. A given functionality might be performed as cooperation among different layers or independently.

The rest of this section first provides an overview of the two layers, and of some security aspects related to issues of data access. Then, it describes in some further detail the elements of the Service Layer, and in particular the Asset Manager (Section 2.1) and the Interoperability Services (Section 2.2).

Data Layer

Generally speaking, the travel services provided by different transportation sectors usually are not an isolated set of operations, but they require data collection from many sources. Such data may comprise a wide range of categories and types including various standards and specifications, transportation ontologies, code lists, historical mobility data, etc. Currently, access to data is achieved through one-to-one data exchange among different parties. Given the divergence of standardization within and across the transportation domain, such data exchange should be then followed by a hard-coded translation data model and standardization of source data to the desired specification. One of the central goals of the IF is to overcome this barrier through the provision of semantic interoperability. Firstly, by the development of a reference ontology defining a shared meaning of the exchanged information and secondly, by conversion and enrichment of non-interoperable and heterogeneous data to this shared model. In this direction, the Data Layer relies on the back-end databases and management of database operations needed to handle collection, storage, and retrieval of data. It necessarily includes RDF stores keeping RDF graphs such as ontologies, enriched meta-data (according to the reference ontology) and meta-data generated by the Asset Manager describing different assets.

Furthermore, it could (optionally) contain other types of data stores (e.g., to store GTFS data), fostering a sharing ecosystem where different actors expose data and utilize data offered by other parties, while respecting privacy and security concerns.

Accordingly, the technologies, standard and patterns that are most relevant for this layer include Triple Store data bases such as RDF4J [4] and Jena [5], and Graph DBs such as Neo4j [6] and Amazon Neptune [7]. In addition, this layer should practice and follow the Linked Data concept [8]. Linked data is a method of exposing and publishing structured data using web technologies and interlinking the ontologically-relevant pieces of knowledge together. Linked data envisions a structured web of self-describing data (using vocabularies, URI and RFD) where global data graphs spanning through heterogeneous data sources could be discovered, navigated and queried.

Service Layer

The Service Layer in the IF plays a key role to envision a unified and smooth collaboration among various travel service providers as well as consumers of such services, including IP4 applications. In this context, we have defined three categories of services, namely Asset Manager, Interoperability Services and Auxiliary Services. While the services offered by the Asset Manager cover the generic operations required for overall management and accessibility of IF, Interoperability Services include a set of special-purpose operations to bridge the interoperability gap between fragmented transportation operators. Finally, any party could expose its services to be explored and utilized by others; in the scope of the IF, we call Auxiliary Services the services supporting these features.

Security Aspects

The Security aspects in the IF are related to how it accesses data described in the Asset Manager. As described before, the IF focuses on enabling semantic interoperability by providing users the tools and building blocks that can be used to implement an interoperability solution. It is not meant to be a database or a centralized solution, and its main principle is that data should reside as much as possible in the originating systems. To this extent, the main intended usage of the Asset Manager is to manage metadata about assets, therefore storing pointers to data, not data itself.

The information contained in the Asset Manager can be used to enable more advanced features, and it is a key feature of the Asset Manager that it should be able to access referenced data. In case of publicly available data, or when the Asset Manager is used to store data (which is allowed, although it is not the intended use), the IF can leverage it to increase the automation level and to lower the effort of implementing an interoperability solution for a transport operator. In case of private data, protected via any authentication scheme (username and password, OAuth, JSON Web Token or any other means), this is not possible unless the Asset Manager is provided with the user's credentials for the specific resource. This could open security holes since the Asset Manager would also store highly sensible information. It is yet to be investigated whether the automation solutions could be made available for private remote data, and whether such solutions could benefit from an explicit description of the authentication schemes to be included in the assets' metadata.

2.1 ASSET MANAGER

The Asset Manager is a pivotal component of the IF offering the basic functionality to publish, share, discover and manage various artifacts that might be published/utilized by

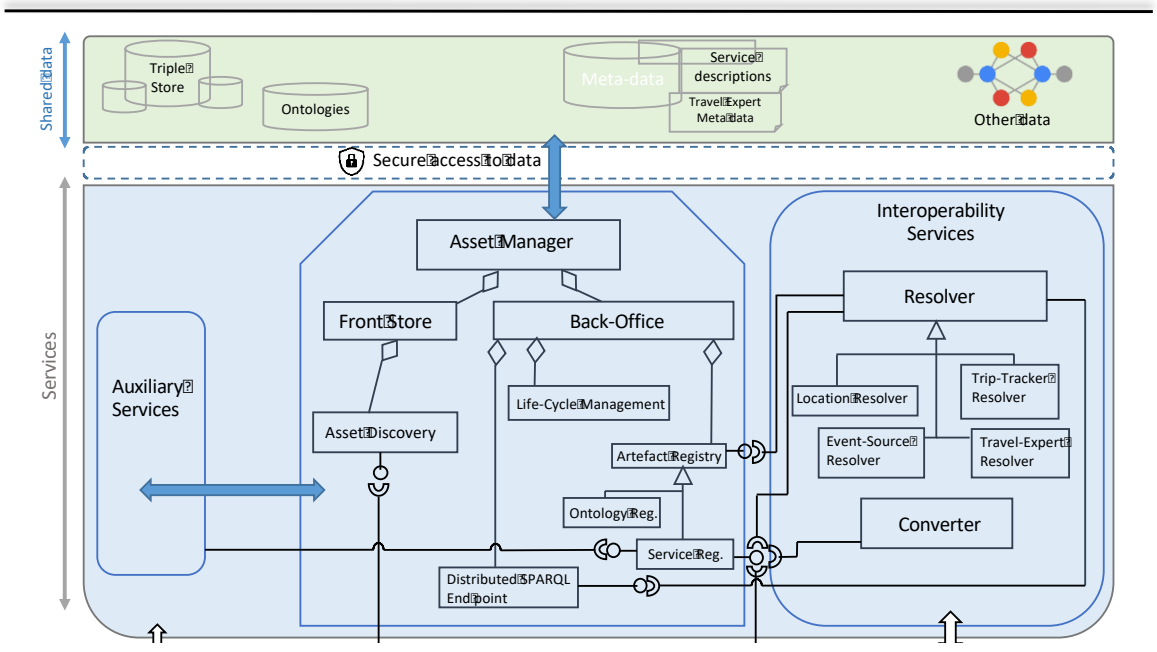


Figure 2 – Internal Components of the Interoperability Framework

external clients and other internal components of the IF such as Converters and Resolvers (see also Section 2.2). As its same suggests, this component is centered around the concept of Asset. Its primary objective is the provision of tools for the registration, storage and then the discovery of such assets. In the scope of the IF an asset includes – but is not limited to – ontologies, data sets and service descriptions for interoperability services such as converters and auxiliary services. The Asset Manager hence constitutes the initial point of interaction with the IF from the external client point of view, and it is the key element that interconnects different components and layers of the IF from an internal perspective.

The Asset Manager includes a number of internal components, which are shown in Figure 2, and which are described in the rest of this section.

2.1.1 Front Store

The Front Store is a composite component maintaining necessary functions, such as the Asset Discovery described below, aiding internal/external service and data consumers to interact with IF.

Asset Discovery

Asset discovery is the capability of automatically identifying assets according to certain requirements. Such requirements can vary from Asset type to Asset type, therefore this component allows defining multiple ways to express requirements related to a specific Asset type. The discovery capabilities are offered through a set of configurable Web APIs, each one tailored to let users query for a specific aspect (or set of aspects).

This feature is enabled by the SPARQL [9] query capabilities of the RDF repository used to store metadata. When the discovery API is configured to search for Assets belonging to different Asset types, it exploits the common set of metadata which are used to describe all the Assets. When referring to a specific Asset type, the discovery API can rely on the specific metadata of each Asset type.

2.1.2 Back Office

This composite component refers to suites of functions and services that are mainly of interest for the internal/external service/data providers, including Life Cycle Management, Artefact Registry and Distributed SPARQL endpoint.

Life Cycle Management

It is an optional component which supports a formal definition of the workflow sequence of stages that must be followed by different types of assets. Life cycle management in the scope of the IF might be applied to track the evolution of a single asset, or to enable a particular workflow process. The former refers to the definition of the different states in which an asset can be (e.g., creation, approval, revision, removal), where the transition between each pair of states should be carefully tracked by the asset manager. The latter includes the situation where the coordination of multiple services is defined as a workflow. For instance,

the deployment of a particular service should be triggered upon completion of another service/process.

BPMN [10] is the best-known standard for the definition of workflow definition/assertion. In addition, the availability of different supporting platforms, such as camunda [11], that support BPMN through a graphical user interface makes this a suitable and practical approach that could be exploited within the IF.

Artifact Registry

The Artifact Registry is the component which stores relevant metadata about all the known artifacts. This component acts also as a transformation layer from the JSON representation of the artifacts which is used in the Web APIs to the internal representation inside the RDF repository.

This component can be configured to host different Artifact Types, such as Ontologies, Services and RDF Datasets. Each artifact type features:

- A common set of metadata expressed via vocabularies such as DCAT-AP [12] and ADMS [13]. This enables a uniform way of discovering assets via standard SPARQL queries.
- A specific set of metadata expressed in JSON-Schema.
- A set of lifecycle management processes. This enables identifying the processes to be triggered when an operation is attempted on an artifact.
- The instructions required to convert metadata from the JSON object provided by the user to their RDF version.

Distributed SPARQL endpoint

The distributed SPARQL endpoint (also known in the literature as federated SPARQL query processor) is a component that is able to evaluate SPARQL queries over a set of SPARQL endpoints that may belong to different organizations (and hence available under different domain names), providing a unified access to a complementary set of (sometimes overlapping) knowledge graphs.

The concept of federated query processing in SPARQL was not present in the initial version of SPARQL (SPARQL1.0) and was introduced in SPARQL1.1, with the inclusion of the SERVICE clause in SPARQL queries, which is used to provide references to the SPARQL endpoints to be considered. There are also many works in literature that consider that SPARQL queries may be written without specifying the exact location (aka SPARQL endpoint) where each graph pattern will be evaluated, and hence the federated query engine is responsible for locating the SPARQL endpoints where the different parts of the SPARQL query will be evaluated. That is, these works include an additional step of data source identification and query planning. This is done in systems like FedX [14] and ANAPSID [15].

2.2 INTEROPERABILITY SERVICES

Interoperability Services comprise the set of functions, processes and tools that are specifically designed to fill the interoperability gaps between the heterogeneous operators of the mobility and transportation sectors and facilitate the seamless co-operation among them. These services address the generic and primary interoperability requirements that are shared among various actors of the transport ecosystem and might be utilized by them in different manners and to accomplish different goals and applications.

Figure 2 shows the main elements of the set of Interoperability Services, which are described in the rest of this section.

2.2.1 Resolver

Resolver services are specialized Interoperability Services dedicated to providing access, location, relocation and replication transparencies to interacting applications, masking them from the physical distribution, access protocols and formats of meta-data and data resources available in the Data Layer. Example resolvers developed in the IT2Rail project [1] are the following:

- **Location Identification** returns geographical coordinates of Locations that a Traveller requests by name.
- **Locations Resolver** returns a list of Stop Places within a requested radius from a point specified by its geographical coordinates. It is used during the Shopping process to identify transportation stops in the vicinity of Locations selected by Travellers from the list returned by Location Identification.
- **Network Statistics Provider** generates “meta routes” operated by Transportation Service Providers. These “meta routes” are elements in the construction of meta-network used by the Shopping process.
- **Travel Expert Resolver** identifies Travel Expert and Booking Engine web services that can generate offers and bookings for specified “meta travel episodes” that satisfy a Traveller’s mobility request at the time of Shopping and Booking. It is used by the orchestrators to identify the subset of networked Travel Experts that participate in a coordinated distributed shopping and booking one-stop-shop instance.
- **Navitia Decoder** associates Stop Places and Transportation Services with the encodings used by the Navitia platform[1] for use by Trip Tracking in the identification of disruptions.
- **Travel Expert and Booking Engine Brokers** are a special type of resolvers (according to the definition above), which mediate the interaction between the

[1] https://www.navitia.io/

Shopping and Booking orchestration functions, respectively, and the Travel Expert or Booking Engine services provided by Transport Service providers for the generation of offers and bookings that satisfy Traveller's mobility requests.

2.2.2 Converter

As mentioned earlier, the primary goal of the IF is to overcome the fragmentation of the transportation ecosystem by fostering semantic interoperability, which smooths the interactions and cooperation among transport-related services despite the heterogeneity of the underlying data models. A system is semantically interoperable when exchanged data could be understood unambiguously and interpreted uniformly. In this direction, Converters, which act as adapters between two distinct formats and are able to map the information expressed in one format to the other, are essential elements in the IF.

SPRINT will extend the Converter (briefly overviewed in Section 5) that was developed in the ST4RT project [2], with the aim to improve its performance and automation. The core idea behind the future enhancement of the Converter is the concept of componentization – or modularity – which is broadly accepted as a good practice in software engineering. The current body of the Converter component has been built as a sequence of loosely coupled phases, which fosters the idea of transforming it into a modular architecture. More precisely, the Lifting Phase exploits an annotation-based approach to translate java objects representing data described in a source format into RDF graphs. It is then followed by an Enrichment Phase to fill in any missing data. Similarly, in the reverse process, the Lowering Phase takes the aforementioned RDF graph to build the desired instances of Java classes representing data described in the target format. In between, there are two more phases, namely the Ontology Loading Phase and the Rule Transformation Phase, which provide additional support to expand and modify the RDF graph through inference and/or dedicated transformation rules. Each of these phases could be implemented as a stand-alone module to maximize the performance of the Converter, which leads to an overall performance enhancement.

The benefits of the decomposition of the Converter into smaller independent units, especially when implemented with cutting-edge technologies, promise to be significant (see also Section 6). The next section surveys various architectural options that could be used to improve the Converter.

3. ARCHITECTURAL PATTERNS FOR DISTRIBUTED AND HETEROGENEOUS SYSTEMS

This section overviews the general architecture patterns suitable for construction of the IF given the distributed and heterogeneous nature of mobility and transport domain. In particular, it first overviews service-oriented architectures (Section 3.1), then it studies cloud-based patterns (Section 3.2), and finally it tackles peer-to-peer architectures (Section 3.3).

3.1 SERVICE-ORIENTED ARCHITECTURES

The Service-Oriented approach is among the most popular paradigms in distributed computing. The core idea is to encapsulate in, and expose the functions offered by the system as "*Services*". In this context, a service refers to a contractually specified functionality with specific properties such as discoverability, location transparency, loose coupling, interoperability [16].

Given the fragmented transportation ecosystem composed of administratively and geographically distributed actors and operators, service-based techniques and technologies seem relevant approaches to build the IF. Service-Oriented Patterns and technologies could be categorized mainly in two different levels: how to design and develop a single service and how to design and develop an ecosystem of services co-operating with one another. In the following we first overview the main standards and architectural principles to build services and manage their relevant aspects (Section 3.1.1), then we provide an overview of the leading trends of service-based frameworks (Section 3.1.2).

3.1.1 Service design

In this section we overview the main trends in the development of service-based systems.

WS-*

WS-* standardization (often referred as SOAP) is considered as the legacy model for architecting service-oriented systems. It is composed of a stack of technologies, including SOAP (an XML-based language used to define the messaging architecture and format), XML for the payload scheme, WSDL [17] specifications for the description of the interface and capabilities of a service, and most often HTTP as transportation protocol. On one hand, one of the best features of SOAP is its systematic approach to the discovery and binding of a service – through Universal Description Discovery and Integration (UDDI) and many well-established distributed/centralized directory-based discovery mechanisms – which is one of the primary concerns of the IF. On the other hand, WSDL and the discovery process are mainly focused on the characterization of "syntactical" aspects of services; even if many parallel approaches such as WSMO [18] and WSDL-S [19], have been conducted to incorporate semantics into WSDL, the performance and scalability of those solutions are still subject of debate. However, given that many already existing systems in transaction ecosystems are built on and/or interacting with SOAP, *semantic annotation* of service

descriptions seems among the promising practical approaches within the IF to bridge the gap and enhance pure SOAP-based systems.

REST

REST is the dominant development paradigm in the modern web due to its salient features including simplicity, higher performance and smaller bandwidth usage. Moreover, REST principles are mainly focused on standardization of interfaces and it is quite flexible for dealing with various data-formats in addition to XML, which is an important factor in the provision of interoperability. Yet the most relevant principle of RESTful architectures in the context of the IF is the concept of HATEOAS (Hypermedia as the Engine of Application State).

Hypermedia [20] is a well-known concept in the web, through which the human user can find any information only by following links (URI) and interact with a server/application using forms. Accordingly, The HATEOAS principle of REST enables the clients (machines in this case) to bind and interact with a service without additional knowledge of service interfaces. HATEOAS in REST is mainly achieved by attaching meaningful links/URIs to the data models of the transferred data and machine understandable semantics/metadata to such links.

The most famous resource description approaches to build hypermedia APIs include Hydra [21], HAL [22] and Siren [23]. Hydra introduces a vocabulary to specify which types of actions one can perform on a given resource. HAL is a media type which utilizes the concept of link relation to define semantic relations among several resources in a machine-readable format that allows clients to easily navigate through resources. Finally, Siren, provides both resource descriptions, allowed actions on resources, and information about the relation of a resource to other resources.

These specifications tackle a fundamental problem in the web, which is also present in the web of transportation and related service providers' APIs. To give an example, a client application that is developed to interact with a particular travel service usually cannot operate – on the fly – with another one. The application is statically paired with a service and the necessary information to engage with the service is hard-coded at design time. However, each travel service has its own data model and API specification. Accordingly, an application developer must study the API documentation of a service – at design time – and then develop the client application to consume such service. In this direction, the main benefit of the aforementioned media types is to enable the development of self-descriptive and vendor-independent API representations to allow for the seamless automated integration of a – generic – client for any service.

Service Composition and Workflow (Mash-Up)

Service Composition refers to the set of processes followed to mash several existing services up in order to create an added-value service. It is relevant in the context of the IF to compose multiple Resolver services, or combination of Resolvers and Converters, or a

combination of other auxiliary services with Converters. For instance, a client might be interested to a service S1 which provides the desired output represented through standard A, while the expected/desired standard used by the client is B. In this case, a mash-up composed of service S1 and an A-B converter service could be created to enable the user to interact with S1 following the required standard.

3.1.2 Service-oriented frameworks

Service Oriented Middlewares

Middleware-based approaches are often known as the best practice to bridge legacy systems/standards/functions and modern implementations following a façade and adapter-based pattern. The fact that the IF needs to deal with many already-existing legacy systems is another reason that makes middleware approaches relevant to our study.

In general, the term middleware refers to a software layer that resides between the technological layer and the application to hide to application developers the details and complexity of various underlying technologies. It potentially provides a unified and homogeneous view and interfaces to underlying data and functions. As for other domains, Service-Oriented approaches have become the dominant model for middleware development in comparison with other types of middleware such as Event-Driven and Message-Oriented middlewares in distributed systems.

Microservices

The Microservice architecture is inspired by the service-oriented computing and breaks the so-called monolithic application into independently executable artefacts – i.e., “micro” services. Most common monolithic applications package all the server-side components into a single executable unit based on the underlying programming language and framework. For instance, a self-contained JAR file, a WAR (in the case of java application deployable on an application server such as Tomcat), a directory hierarchy (e.g., Node.js). Following the microservices architecture, each business logic and functional unit of an application is presented as a service (most often with a REST API) tied to a narrow set of responsibilities.

The Microservices pattern has become a popular architectural style across different domains mainly because it reduces the complexity of application development and makes it more agile. Similarly, within the scope of mobility and transportation domain, and particularly the development of the IF, a Microservices-based approach seems a suitable architectural style as discussed in following.

- Firstly, in comparison to the monolithic style, microservices-based approaches are more scalable in both the horizontal (inbound request expansion) and vertical (add/update libraries) directions. Both of these factors are crucial non-functional requirements in the construction of the IF given the potentially broad range of service consumers (wide range of mobile applications such as ticketing, shopping, planning,

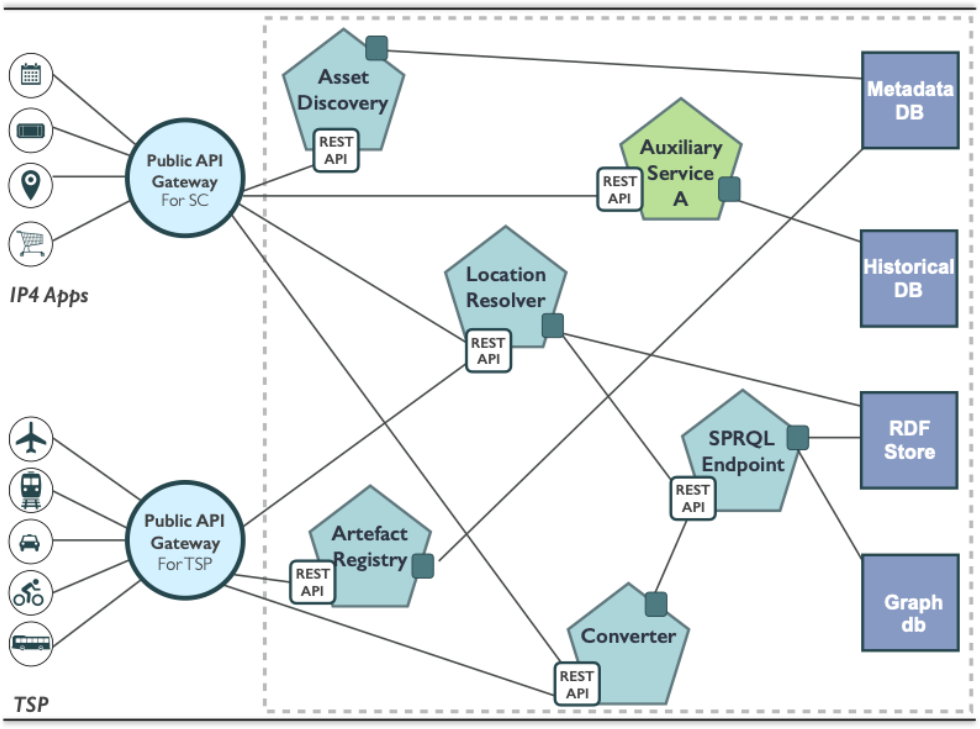


Figure 3-Sketch of Microservices-based Architecture of Interoperability Framework

etc.) as well as stakeholders and service providers of different transportation modes and sectors.

- Furthermore, due to the separation of the concerns and self-containment of each component, a microservice-based application is easier to evolve and to be maintained. It relieves the technology lock restriction of the monolithic application and enables the developers to adopt a diversified set of underlying technologies for programming and development for each microservice. This is a highly important point to be taken into consideration since the IF is expected to be a cooperative framework were heterogenous mobility sectors expose their data and services to other parties either to build a common goal, or to provide separate operations. In both cases, freedom for service providers to choose the underlying technology which best fits with their own infrastructures could greatly encourage such cooperation.
- Another advantage of the functional decomposition of the IF is higher performance and efficiency by matching the best technologies based on the characteristics and requirements of specific functional units. For instance, in the context of the IF, as shown in Figure 3, it could allow for a Historical Data store Service that utilizes time series data technologies for managing historical data and a document-based database technology to store, manage and query other types of artefacts.

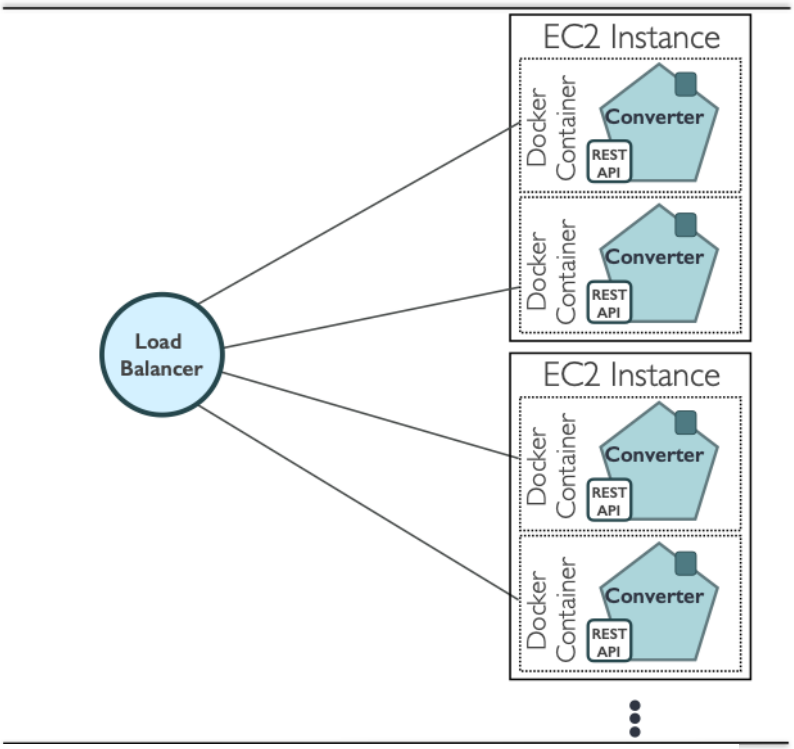


Figure 4 Containerization of Converters

- Moreover, the general idea behind microservices could be an inspiration for the architecture of the internal components of the IF. By decomposing a component into several sub-functions, we could increase their reusability. It also allows upgrading of specific functions to the cutting-edge technologies which in turn enhances the performance of the whole system. For example, in its current implementation, the Converter developed in the ST4RT project is composed of several functional units or steps namely, lifter, enricher, ontology loader, transformation rule manager and lowerer. Each of these functional units could be implemented as a unique service working in – predefined or possibly built at runtime – orchestration with one another. A direct result of such implementation is that there could be multiple instances of each functional unit developed by various service providers where one can choose among them and compose its own version of the converter as depicted in Figure 4.

Enabling technologies for Microservices

As described above, the microservices-architecture is by design centered on loosely-coupled implementations of cohesive sub-components (micro-services) [24]. The promises of microservices would not be achieved without replication of one or multiple instances of each microservice in distributed deployment units. Hence it is not surprising that container technologies and microservices are closely intertwined.

Similar to Virtual Machines (VM) such as VMware [25] and VirtualBox [26], containers isolate a portion of underlying system resources to consolidate multiple applications on a single

server/host. A container is not a VM, but since the applications and usage of these technologies are overlapping, they are often evaluated against one another [27]. Compared to VMs, containers are lighter (with sizes in the order of magnitude of megabytes instead of gigabytes as for VMs) and faster (seconds vs minutes). Firstly, because each VM equates to a complete OS (guest OS) which in turns is heavyweight and very resource-consuming software. Furthermore, the hypervisor itself (i.e., the component of VM frameworks in charge of managing and allocating resources to the multiple guest OSs) consumes some amount of host resources.

Each container, on the other hand, is like an independent process (or workload) which resembles a whole production-like environment along with all the libraries and binaries needed for the application be run all in a single OS. In other words, instead of virtualizing the hardware to run multiple OSs, it virtualizes an OS to run multiple isolated workloads.

Although container technology has been around for a long time [28], it became popular with the advent of Docker [29]. Docker extends the LXC, the Linux container [30] with kernel and application-level APIs. The container engine in Docker, called Docker Daemon, is a service running in the background of the Host OS and manages Docker Containers as an isolated process. Utilizing the namespaces, it provides a totally separated view of the OS and its resources such as process tree, network, and filesystems for each containerized application. The building blocks of the Docker framework are docker files, that is script documents including all the commands necessary to create a docker image. An image is a single pre-built application – or stack of applications –which are ready to run. A Docker container is created through the docker image. It packs an application and all dependencies required to run it into a single software unit which is then portable and executable in any infrastructure supporting Docker daemon.

A consequence of the availability of a portable, self-contained and ready-to-run instance of an application is the possibility to duplicate such applications on a large scale. It potentially solves the scalability problem and opens the door to agile software development. However, the managing and monitoring of hundreds of containers, the allocation of resources to them and the on-demand scaling up/down of their deployment is something beyond the concept of containers itself. For these purposes a container orchestrator is required, which is a system in charge of deployment, scaling and monitoring the containers. One of the most successful examples of container cluster management is Kubernetes [31] which has been open sourced by Google in 2014. Kubernetes takes the Docker image to create a deployment unit for it. One can also specify the bare metal needed for each individual deployment unit. The Kubernetes framework schedules the underlying resources to be used by containers, automates all the processes needed for composing and deploying the containers, and continuously monitors each instance to be well-operated and functioning. Kubernetes together with containers (Docker specifically) not only made microservices architectures popular, but they constitute the next generation of application development and business management in particular by transforming the way cloud-based systems work

[28]. Section 3.2 provides more details regarding the role of these technologies in cloud-based systems.

Modular Systems

Modular software development, as the name indicates, breaks complex software into different modules and defines the software architecture through the dependencies and relations among these modules. According to [32], a module is “a deployable, manageable, natively reusable, composable, stateless unit of software that provides a concise interface to consumers”. A modular software architecture is conceptually similar to the service-oriented approach as they both focus on decompositions of systems into smaller, manageable and self-contained components. However, the inherent difference of a module with a service lies in the composability aspect. More specifically, a service is a logically autonomous unit of software which might have its own application and could be directly consumed by a client independent from other services. A module, on the other hand, is meaningful when it is connected to other modules to compose a system. It is, however, a reusable unit of software, in the sense that a particular module could be combined with various modules to build different systems. Accordingly, dependency among the modules in modular software architectures is an explicit and necessary element.

Modular system architectures are popular within the JAVA community, given that, starting from JAVA 9, a native modular system has been added to this platform. OSGi [33], however, is the best-known modular framework in JAVA, which is also inspired by the service-oriented approach. In the OSGi approach an application is broken down into multiple – extensible and downloadable – bundles such that the service layer could dynamically connect (discover and bind) different bundles to create a coherent business logic. This architectural pattern seems relevant to be considered as a candidate for the development of Converters.

In this context, we could also cite Apache Camel [34], which is an open source Java framework best known as an integration tool that natively supports most of the Enterprise Integration Patterns (EIP) [35]. EIP formalize the integration process by identifying different integration patterns and defining a workflow for the most common business tasks in a generic and code-agnostic manner. For example, Content-Based Router [36] is an EIP that allows routing a message to the desired destination based on its content. Through the Camel framework, one could build these patterns using a set of formal workflows, common vocabulary and basic syntax, instead of architecting and coding from scratch the whole system (which often becomes very complex).

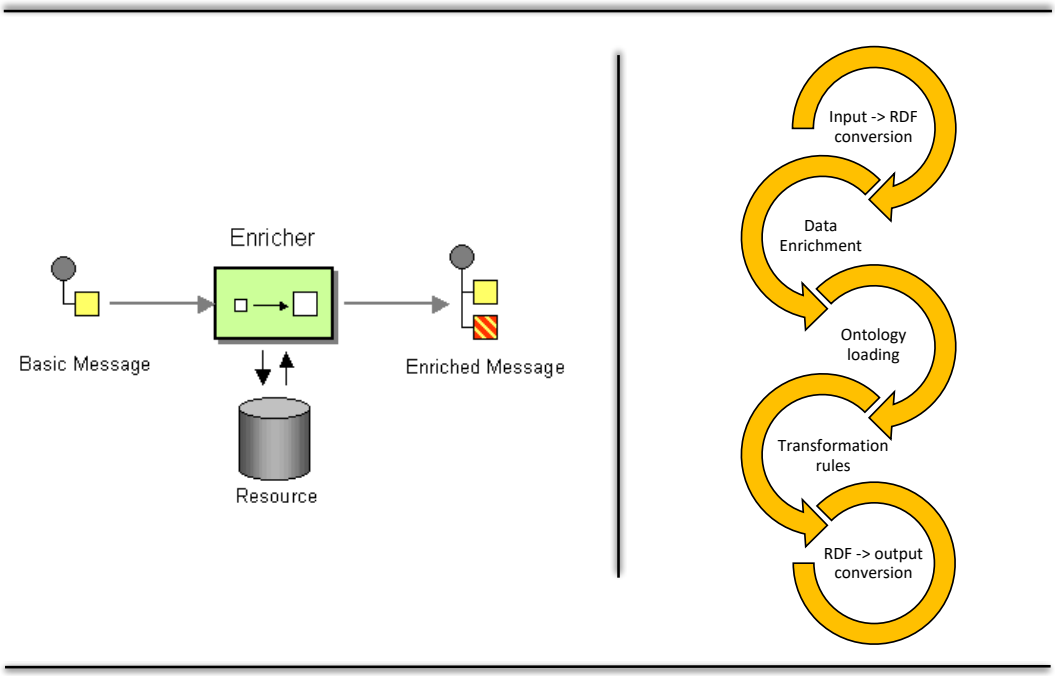


Figure 5 Modularization of the Converter through Content Enricher Pattern

In this vein, the core of a conversion process realized by the IF Converter can be seen as a chain of Content Enrichers [37], which is an example of EIP. Once the input message has been converted into triples, the Resource is the local in-memory RDF repository which is then enriched and transformed. As depicted in Figure 5, each phase of the conversion process could be modeled as one step of the Camel workflow sequence; starting from the input message, each stage is composed of a set of processes and produces the output that ultimately is routed to the successive endpoints.

An interesting aspect of Camel is the portability of the so-called endpoints and routes. They could be deployed as stand-alone components, wrapped as OSGi modules or packaged as containers. Moreover, each block of Camel could be implemented as REST API. These two features make Camel a suitable tool for the development of Converters since it is compatible with advantageous architectural patterns such as microservices, which constitute one of the most interesting approaches for the realization of the IF.

3.2 Cloud-Based Architectures

Cloud computing borrows and utilizes the core idea of the service-oriented approach and has emerged as a promising paradigm to host and deliver services. The definition of cloud-based systems encompasses a wide range of concepts, so that eventually everything (not only functions, but also, for example, infrastructure) could be offered to the interested user as a Service [38]. More precisely, the National Institute of Standards and Technology (NIST) defines cloud computing as a model "*enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage,*

applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [39].

Cloud-based systems are often categorized into three models, namely Infrastructure, Platform and Software as Service (IaaS, PaaS and SaaS), as represented in Figure 6. Although these types of cloud computing are complementary with one another, each type addresses different needs of software development and business management. Accordingly, there is not one model that fits all situations, but each may cover a different

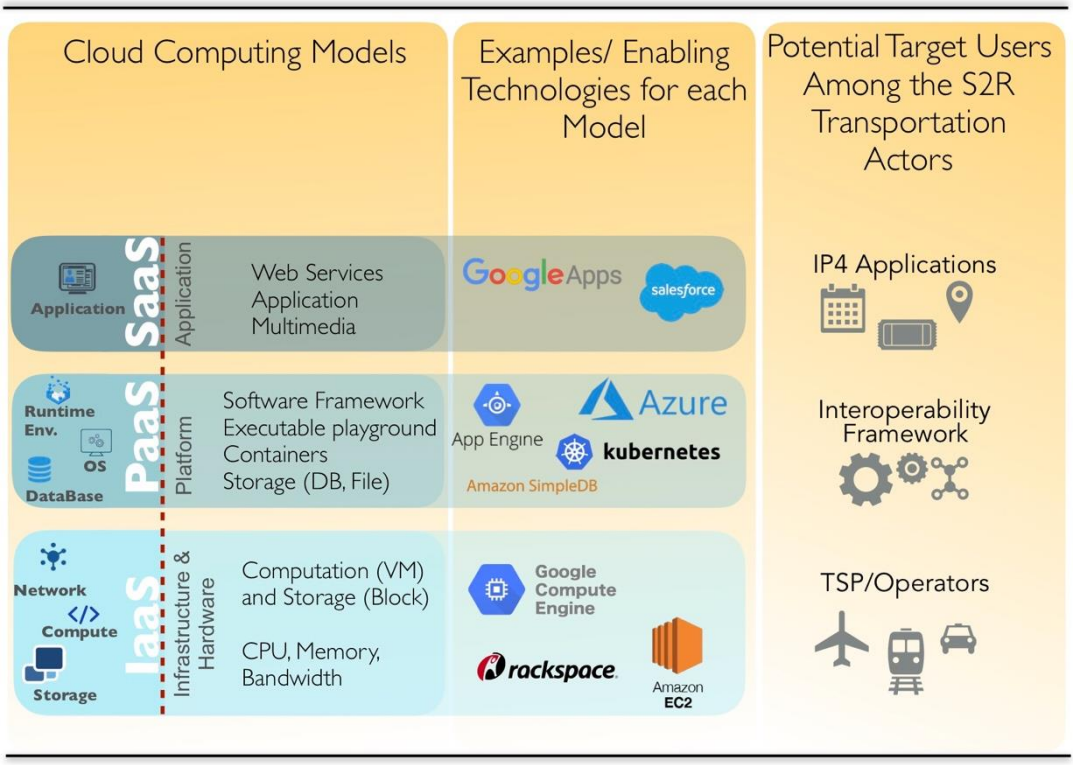


Figure 6 – This figure depicts different cloud models, example and enabling technologies. It also shows which cloud model is more suitable/applicable for different S2R transportation actors.

audience and application domain. In the following, we briefly overview each type and its applicability and benefits in the different areas of transportation and mobility.

3.2.1 Infrastructure as a Service (IaaS)

The lower layer of the cloud system deals with deployment, running and controlling infrastructural resources mainly through Virtual Machines. In other words, IaaS providers offer hardware resources such as processing units, storage, and networking infrastructure to their clients as a service and in an on-demand fashion. In comparison with physical data centers/server allocation, IaaS presents substantial benefits for hosting any type of applications including dynamic scalability, reduced cost and maintenance effort. Apparently, the usage domain of the IaaS form of cloud computing is out of the scope of IF's objectives and responsibilities. However, it is an interesting deployment possibility to be taken into account by various transportation and mobility service providers external to the IF itself.

3.2.2 Platform as a Service (PaaS)

Another form of cloud computing is called Platform as a Service, which offers a development and deployment space for clients through which one can code, run and test an application. In general, the PaaS layer is built on the IaaS layer and utilizes the underlying computation, network and storage resources to support users in two directions. Firstly, PaaS offers a suitable platform based on the programming language of the client's choice plus libraries, services and tools which support the whole life-cycle of application development. Secondly, it is in charge of managing all the required aspects for deploying an application – usually on a large scale – including cluster scheduling, load balancing, DNS automation, etc.

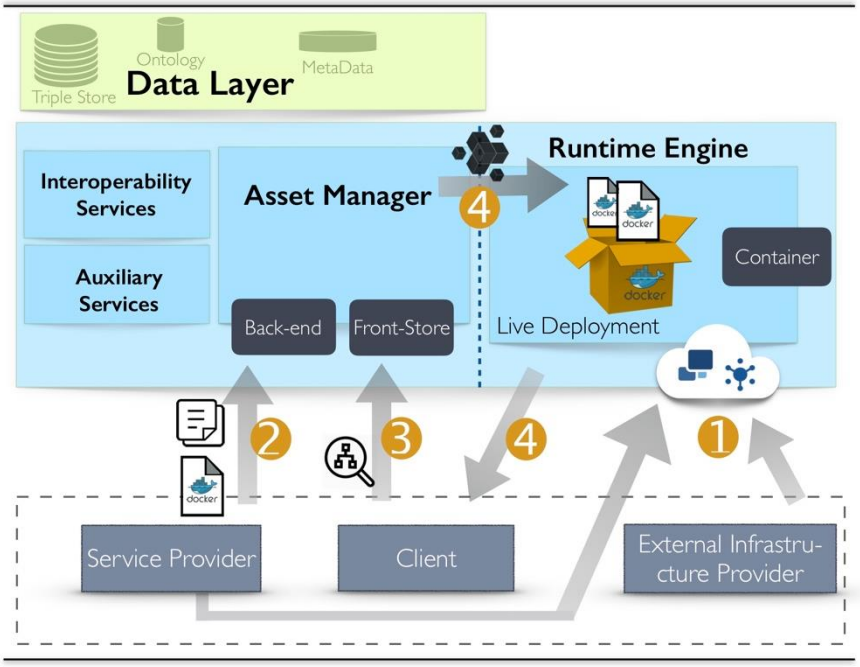


Figure 7 Runtime Environment for the IF

The former offers the so-called in-house development that sets up a full-featured and browser-based coding/compiling framework, that relieves developers to build up and manage the development environment. The latter then supports the automatic packaging, deployment and scaling up/down of the application. In this direction, containers (see Section 3.1.2) are among the key enabling technologies for the materialization of the PaaS model. In the early days of cloud computing, the two above-mentioned features of PaaS were closely intertwined. Furthermore, the client had limited control over the underlying infrastructure since it was owned by the PaaS provider. For instance, in Heroku [40] (among the best-known PaaS provider) an application could be deployed only on the Heroku's AWS infrastructure using Dyno (a lightweight Linux container).

Though the aforementioned scenario is still a popular way of implementing the PaaS model, the new trend in software architectures such as devOps and microservices is to shift the PaaS implementation towards more loosely coupled approaches. Specifically, the emergence of technologies such as Docker [29]and Kubernetes [31] has decoupled the container management and orchestration tasks from the PaaS layer. In other words, the

base resources are no longer restricted to those offered by the PaaS provider; instead, the client can choose the desired infrastructure (i.e., the cloud provider) and has full control over it.

This new trend of PaaS implementation, which is mainly focused on the provision of containers and their orchestration and management, seems a suitable approach for service deployment in the IF. As depicted in Figure 7, one way of reaching the goal of the IF of fostering collaboration and interoperability among transportation actors could be to provide an “executable environment” that lets clients seamlessly deploy the desired service. Following this approach, the IF could be linked to some cluster of infrastructures and take care of the required configuration and dependencies, to run one or a composition of multiple services and make them ready to be consumed by the client.

3.2.3 Software as a Service (SaaS)

The higher layer of cloud-based architectures is Software as a Service. Compared to IaaS and PaaS, it is the more tangible model for end-users since it deals with the actual application and fosters a new form of application utilization. SaaS refers to a model of software which is managed remotely via distributed and virtual resources and is delivered to the user in an on-demand fashion. Accordingly, end-users do not purchase and install software on their own machines, but utilize it – usually – on a subscription basis and have access to it via the internet. From the user perspective, this form of obtaining the right to use the software is advantageous for many reasons. It reduces the initial investment to buy a license, saves money as soon as one stops using the software and increases the safety of data since everything is stored on the cloud rather than on personal PCs, which are more vulnerable to damage and various security hazards. Similar to other domains, the transportation community has moved in the direction of this model. For example, the concept of Mobility as a Service [41], which is an incarnation of the SaaS model, is becoming a more and more popular and widespread approach. Finally, in this context, since the IF is not directly used by ordinary end-users, it might not be the main audience for the SaaS model. On the other hand, the SaaS pattern seems a suitable approach for delivering the results of IP4 projects.

3.3 Peer-2-Peer Architectures

Peer-to-peer (P2P) software architectures were introduced as an alternative to centralized software infrastructures, and in particular to the client-server model. In this architectural style, there is no distinction between server nodes and client nodes, and each node logically acts as both [42]. Accordingly, the requirement for a central component to maintain and manage the knowledge about the whole system is reduced, especially in an unstructured P2P system. Each node has its local and partial view of the system, and could autonomously initiate a connection to other nodes.

P2P became very popular both in academia and industry during the 2000s, leading to a number of successes in a wide range of domains, such as for example file sharing [43]. The P2P model soon found its way in web services field. In particular, structured P2P systems, such as Chord [44] , CAN [45], Pastry [46], and Napster [47] proved themselves as a suitable mechanism for distributed web service discovery [48].

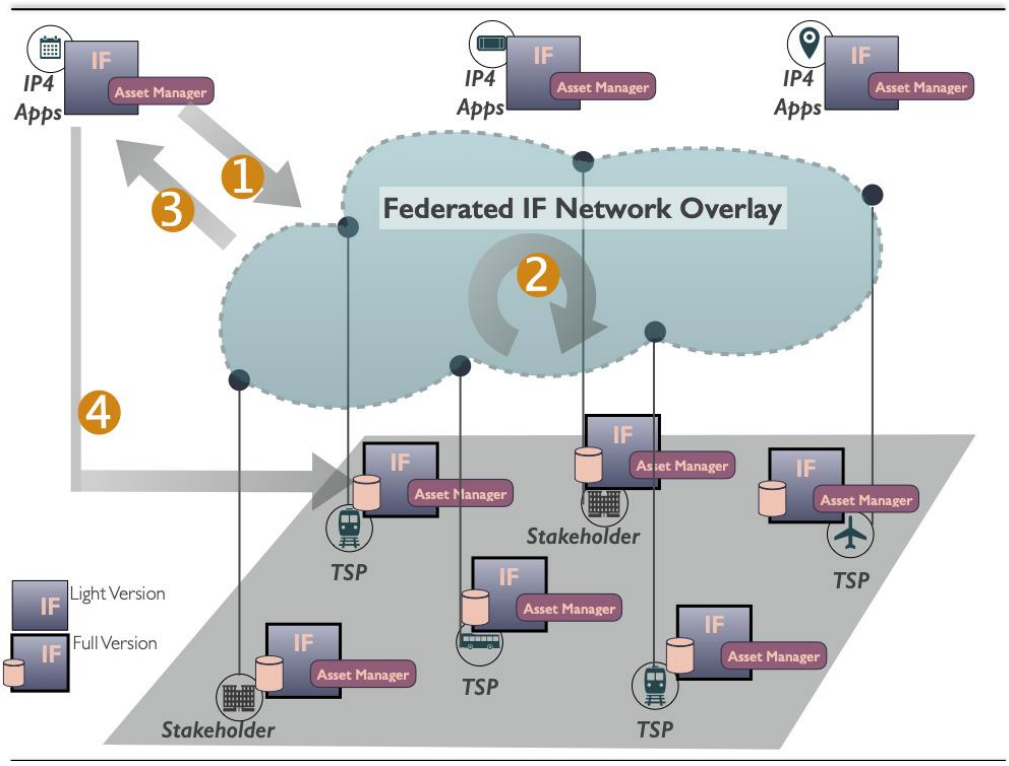


Figure 8 Federated IF

Although the popularity and number of implementations of P2P system has decreased in the last decade – especially in favor of cloud computing approaches –, successful industrial implantations of P2P systems such as Skype shows it still could be advantageous approaches in certain situations. In addition, some researches aim at combining P2P and cloud approaches [49], [50], [51], for example to exploit P2P technologies for the distributed management of cloud resources to overcome the shortcomings of centralized cloud management approaches.

Similarly, the centralized management of the IF may jeopardize its overall scalability and robustness. Making a one (logical) node the single coordinator of the whole system would become a performance bottleneck as the system grows, and it would suffer from the single point of failure problem. To avoid this, a P2P implementation of the IF that relied on a federated architecture (such as the one depicted in Figure 8) would be beneficial under a number of aspects. Indeed, in a federated approach, no single entity would own the IF and, as it is common in P2P systems, the system would be functioning and growing as a grassroots effort of the whole community. Accordingly, it allocates different tasks to multiple nodes, which exchange with one another only the portion of data which is of interest for the recipient. Hence, in comparison to the centralized architecture (which can be only logically

centralized as in cloud-based systems, or both logically and physically centralized such as in client-server approaches) where all data must be transferred to a single unit responsible for all the processing/storage tasks, a P2P approach potentially increases the performance and reduces the storage, processing and communication costs.

Furthermore, the cloud is a costly solution from the economic point of view. P2P however, fosters the idea of “volunteer Computing” [52], where each transportation actor would share its infrastructure to provide the storage and processing required by IF, which, in turn, makes the interoperability among actors in the community possible. Finally, this deployment scenario presumes the locality of data, which is kept where – geographically and administratively – it is originated and/or belongs to, and only distributes the meta-data and service descriptions. All the requests to access data/services would be then redirected to the responsible peer and the client could directly interact with the source peer.

Figure 8 sketches the architecture of a federated IF. A transportation actor (e.g., IP4 application, TSP) could then participate and utilize the IF network, by locally installing an instance of the IF. The power and storage capacity of its machine then becomes part of the overlay network. Following common practices, there could be different versions of IF instances, both light and full. The client would mainly participate in the distributed service discovery process, while the service provider would bear the responsibility of data storage. In this scenario the IF acts as a distributed registry, and its responsibility reduces to discovering the desired service/data for the user. A discovery result is composed of a service description and the endpoint to be called. Deployment and operation of discovered services are then on the premises of the service provider.

4. SURVEY OF BEST PRACTICES TO ADDRESS INTEROPERABILITY IN DISTRIBUTED AND HETEROGENEOUS SYSTEMS

This section analyzes how interoperability issues have been tackled in other domains. In particular, it looks at the Internet of Things (Section 4.1 and Section 4.2) and cloud (Section 4.3) domains.

4.1 THE BIOTOPE PROJECT

"The Internet of Things (IoT) brings opportunities to create new services and products, reducing costs for societies, and changing how services are sold and consumed. A critical obstacle to further IoT innovation is the "vertical silos" that shape today's IoT landscape. These silos impede the creation of cross-industry, cross-platform and cross-organisational services due to their lack of interoperability and openness. The bIoTope project lays the foundation for creating open innovation ecosystems by providing a platform that enables companies to easily create new IoT systems and to rapidly harness available information using advanced Systems-of-Systems (SoS) capabilities for Connected Smart Objects – with minimal investment." [53]

Similar to the transportation sector, IoT is a heterogeneous domain composed of millions of devices interacting with each other through heterogenous standards, communication protocols, and vendor-specific APIs. IoT ecosystems can be spread over vast geographical boundaries and comprise a wide range of stakeholders. The bIoTope project aimed to tackle the interoperability problem in IoT systems. The work carried out in the bIoTope project toward the construction of a federated IoT system could be an inspiration for the SPRINT project.

The strategy followed by the bIoTope project is a top-down one, where interoperable communications and interactions are achieved by encouraging the community to use certain (open) standards. From an architectural point of view [54], the project has employed a federated architecture where the service and data providers register their services/data in a distributed repository (the so-called O-MI node). Subsequently, a consumer could send a discovery request to an O-MI node to find out the desired data and service as well as to discover other O-MI nodes. The communication with repository nodes is achieved via HTTP and web Sockets, and the unified interpretation of such messages is guaranteed by the strict assumption that actors of the system are using specific standards. In other words, the semantic interoperability is hardcoded in the system. Finally, access to the desired service is managed through peer-to-peer interaction [55].

Another aspect of the bIoTope project that could be of interest for SPRINT is their solution for the management of access control requirements. More precisely, the secure access to the data in their system is managed through a token-based system where the bIoTope gateway generates and verifies such token on behalf of the service provider. It hence requires that the data owner relinquishes full access to the marketplace to manage authorization and privacy aspects. The project has foreseen some mechanisms for the data

owners to specify how and under which condition different access permissions must be assigned to the registered users [56].

4.2 OpenIoT - The Open Source Internet of Things

"The OpenIoT middleware infrastructure will support flexible configuration and deployment of algorithms for collection, and filtering information streams stemming from the internet-connected objects, while at the same time generating and processing important business/applications events. OpenIoT is perceived as a natural extension to cloud computing implementations, which will allow access to additional and increasingly important IoT based resources and capabilities. In particular, OpenIoT will research and provide the means for formulating and managing environments comprising IoT resources, which can deliver on-demand utility IoT services such as sensing as a service" [57].

From the architectural point of view, OpenIoT is a middleware-based approach enabling the semantic unification of diverse IoT applications in the cloud. At the lowest level, the sensor middleware collects sensed data from physical and virtual sensors and distributed gateways. Data eventually are cast to the cloud layer (Linked Stream Middleware, LSM) where users could initiate discovery request over data. Furthermore, it provides different components and interfaces for semi-automatic service request definition and presentation [58].

To achieve full interoperability, OpenIoT has defined an ontology as an extension of Semantic Sensor Network (SSN) [59] that is one of the best-known ontologies in the IoT domain developed by the World Wide Web Consortium (W3C). The ontology is a single OWL file which is used for automatic documentation and annotation. LSM takes the raw data as input and converts them to semantically-annotated data in RDF format as output. In addition, the cloud layer stores metadata regarding the sensors and their functions. LSM then exposes SPARQL endpoints to enable the exploration of these semantically annotated data. One of the interesting approaches in OpenIoT is its implementation of continuous queries. For this mode of querying, the SPARQL endpoints of the LSM provides necessary interfaces for users to define their queries. Such a query would be re-triggered as soon as new data arrives in the system and the result would be sent back to the subscriber [60].

However, one limitation of OpenIoT is its authentication and authorization management, which is handled in a centralized manner. Users are requested to register to the system providing a name and password. The Central Authentication Service (CAS) then provides the authenticated user with a token which is valid for a certain time. Such a token is generated based on the permission conditions defined by the service provider beforehand. Permissions are textual values that define actions or behaviors and are defined per service.

4.3 Interoperability in Cloud Frameworks

Another computing domain where interoperability has become a major challenge is cloud computing. Similar to our case, cloud computing is composed of distributed resources (physical, networking and services) working together to realize the promise of cloud computing, that is a global market of collaborative services [61]. The interoperability problem in the cloud stems from the intense competition among the giant cloud providers such as

Google, Amazon and Salesforce, which makes them reluctant to converge towards unified standards. Cloud interoperability, in general, is "*the ability to write code that works with more than one Cloud provider simultaneously, regardless of the differences between the providers*" and semantic interoperability is concerned with how different cloud systems express and understand the same information [61].

The work presented in [61] proposed an open Reference Architecture for Semantically Interoperable Clouds (RASIC). RASIC acts as a mediator between cloud providers with the aim to resolve the semantic conflicts. However, their main contribution is the development of a model for a generic API through which cloud consumers can specify their requirements. The authors discuss how current cloud providers are offering analogous services with similar actions and properties, but through heterogenous APIs, names and structures. They suggest that ontologies and a unified modeling approach could be employed to overcome such interoperability issues. In particular, the authors have developed an ontology and vocabulary for the semantic annotation of both services and resources in the cloud; then, through their semantic engine, a cloud provider could formally (using RDF and OWL) add the mapping between its data model and the reference architecture's models. Using these mappings and suitable reasoning approaches RASIC could find semantically matched concepts and resolve any semantic conflicts at runtime.

In this context, another project of interest is a semantic interoperability framework for SaaS systems in cloud computing environments [62]. The authors discuss that while syntactic interoperability has been achieved in the cloud, it severely lacks semantic interoperability. To overcome this issue, they proposed a broker-based approach which stands between cloud consumers and providers and takes care of the operations needed to ensure interoperability. Following the reference architecture, a federation of the clouds is created so that consumers could not perceive the distribution and fragmentation of the underlying cloud, but worked with the broker as if it were a single cloud computing system. Each cloud provider is responsible to register its services, as well as the service agreements between the provider and the consumers. Subsequently, consumers (of different cloud providers) interact with to the broker as a single gateway to discover, deploy and manage any services.

In addition, the broker plays a central role to provide semantic interoperability. The cloud providers submit the syntactical description of their services through WSDL specifications to the broker. The semantic interoperability layer of the broker then creates the semantic description of the service explaining in a unified manner what it does and what are the requirements, limitations and service quality. Firstly, the WSDL2OWL-S component generates such semantic description out of the WSDL and then, using a special-purpose semantic editor, one can add any additional information to it. Another component of the broker is the ontology repository composed of an ontology editor to create and manage different ontologies to represent the mapping knowledge space, as well as different types of mappings that a consumer might need.

5. RESULTS OF THE ASSESSMENT OF THE FUNCTIONS IMPLEMENTED IN COMPANION PROJECTS

This section provides an assessment of the functions related to the IF implemented in past S2R-related projects. We focused on the outcomes of the projects for which we were able to get access to the developed components, and in particular those of the IT2Rail [1] and ST4RT [2] projects. In particular, Section 5.1 analyzes the services developed in the IT2Rail project, and Section 5.2 assesses the converter technology developed in the ST4RT project.

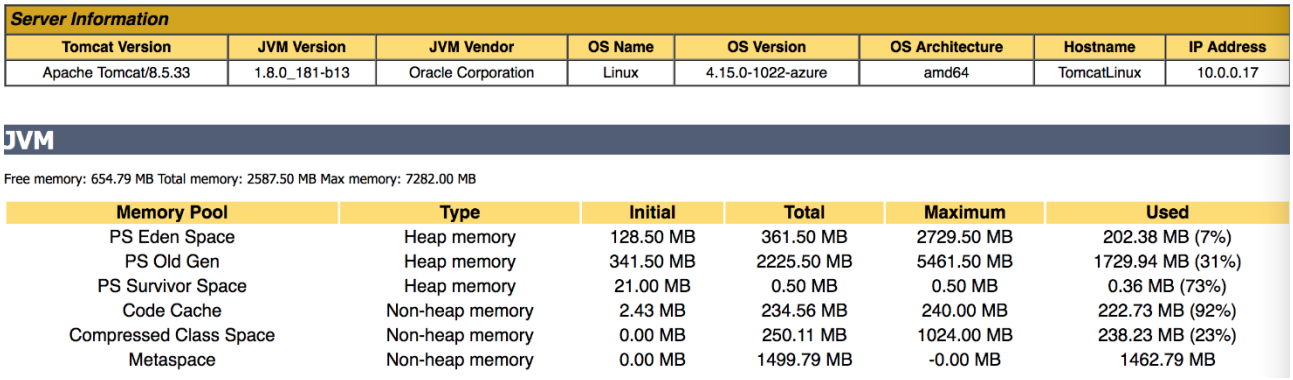
5.1 IT2RAIL PROJECT

The IF technical demonstrator implemented within the IT2RAIL project executes in an environment that is equipped with:

- an installation of the open source Ontotext GraphDB semantic graph database, which implements the data layer
- an installation of the open source Apache Tomcat web application server, which hosts interoperability services
- an installation of the open source Apache WSO2 Carbon middleware, which implements the Asset Manager

all using the Java Development Kit 1.8 and runtime.

Figure 9, Figure 10 and Figure 11 show the details of the three installations:



Server Information

Tomcat Version	JVM Version	JVM Vendor	OS Name	OS Version	OS Architecture	Hostname	IP Address
Apache Tomcat/8.5.33	1.8.0_181-b13	Oracle Corporation	Linux	4.15.0-1022-azure	amd64	TomcatLinux	10.0.0.17

JVM

Free memory: 654.79 MB Total memory: 2587.50 MB Max memory: 7282.00 MB

Memory Pool	Type	Initial	Total	Maximum	Used
PS Eden Space	Heap memory	128.50 MB	361.50 MB	2729.50 MB	202.38 MB (7%)
PS Old Gen	Heap memory	341.50 MB	2225.50 MB	5461.50 MB	1729.94 MB (31%)
PS Survivor Space	Heap memory	21.00 MB	0.50 MB	0.50 MB	0.36 MB (73%)
Code Cache	Non-heap memory	2.43 MB	234.56 MB	240.00 MB	222.73 MB (92%)
Compressed Class Space	Non-heap memory	0.00 MB	250.11 MB	1024.00 MB	238.23 MB (23%)
Metaspace	Non-heap memory	0.00 MB	1499.79 MB	-0.00 MB	1462.79 MB

Figure 9 - Apache Tomcat web application server

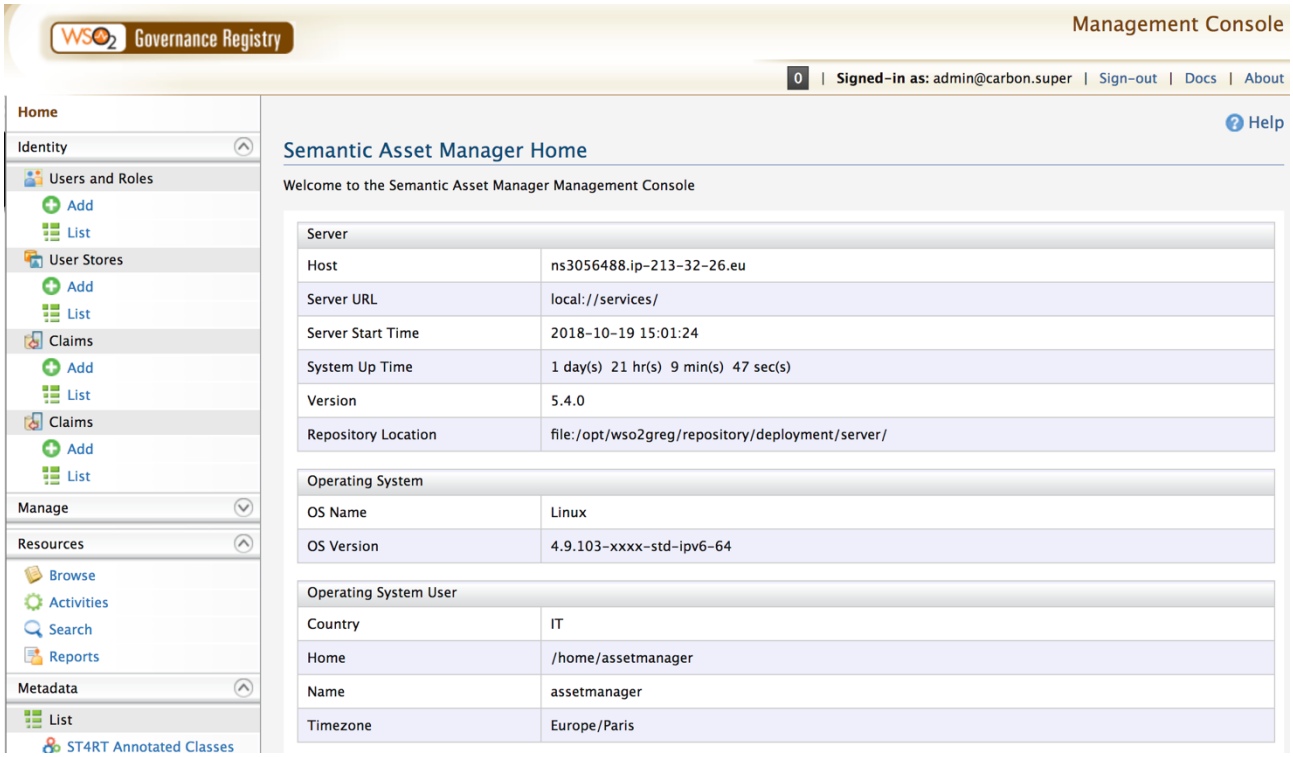
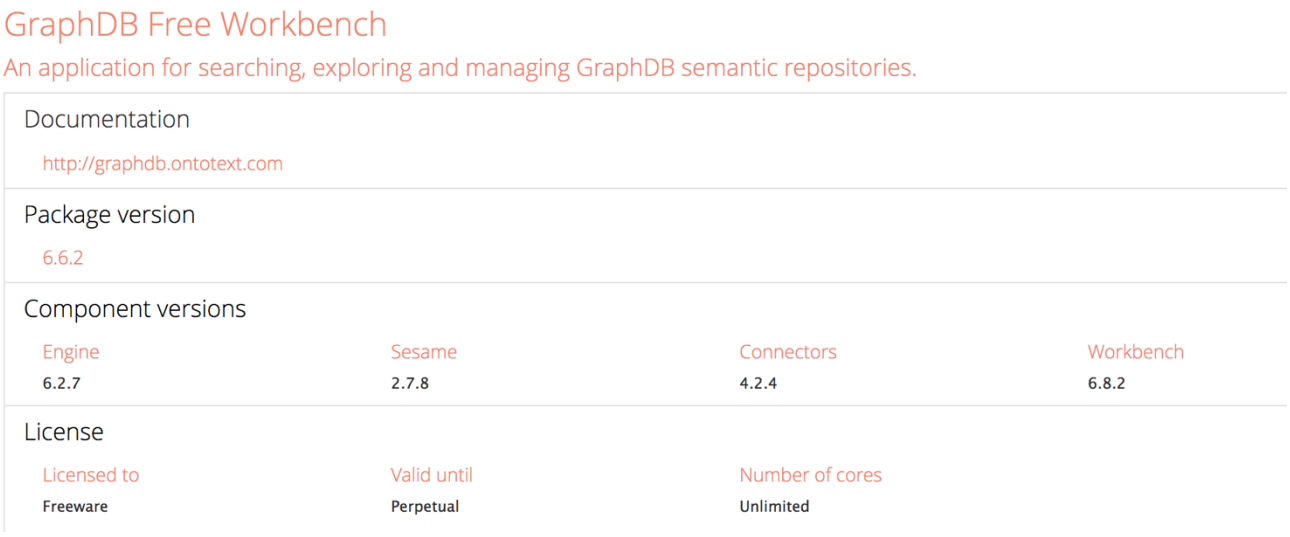


Figure 10 - Apache WSO2 Carbon Installation (Asset Manager)



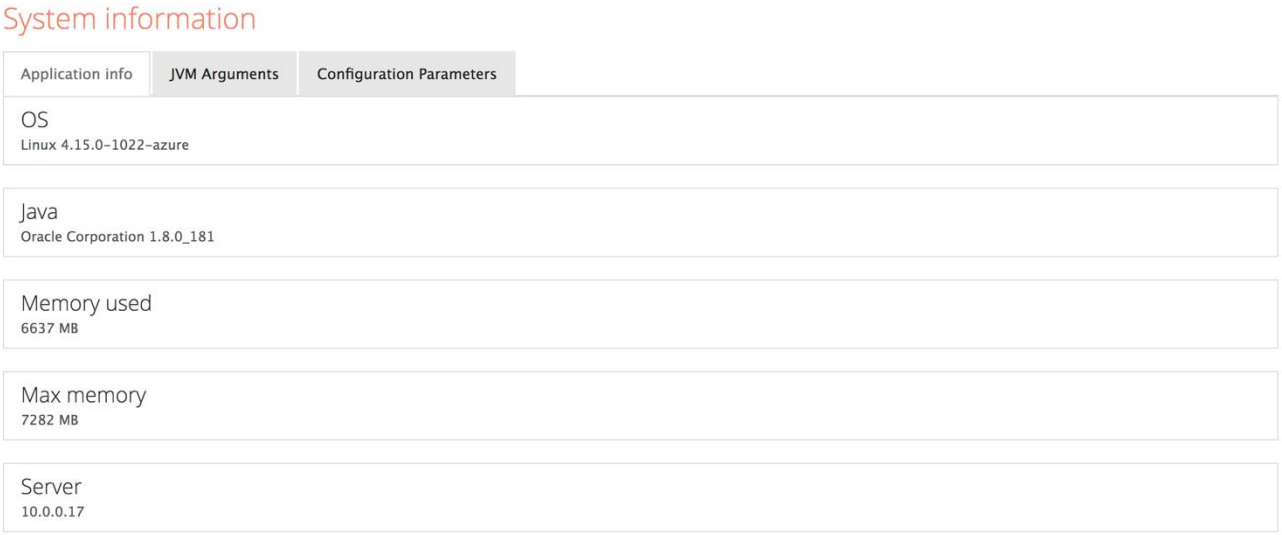


Figure 11 - Ontotext GraphDB Semantic Graph database

The interoperability services of the IF are implemented as Web Archive (WAR) files deployed on the Apache Tomcat web application server. Figure 12 shows the deployed services on the Apache Tomcat management console.

/it2rail-bookingengine-broker-3.0-FREL	*None specified*	Booking Engine Broker F-REL
/it2rail-location-identifier-3.0-FREL	*None specified*	Location Identification F-REL
/it2rail-locations-resolver-3.0-FREL	*None specified*	Location Resolver F-REL
/it2rail-navitia-decoder-3.0-FREL	*None specified*	Event Source Resolver - Navitia Decoder F-REL
/it2rail-netex-provider-3.0-FREL	*None specified*	NeTEX provider F-REL
/it2rail-travelexpert-broker-3.0-FREL	*None specified*	Travel Expert Broker F-REL
/it2rail-travelexpert-resolver-3.0-FREL	*None specified*	Travel Expert Resolver F-REL

Figure 12 – IT2Rail interoperability framework services deployment on Tomcat container

For both unit and integration testing of the IT2Rail pilot demonstration the SoapUI 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.

Figure 13 shows a snapshot of the SoapUI tool configured for testing of the main Interoperability Framework services.

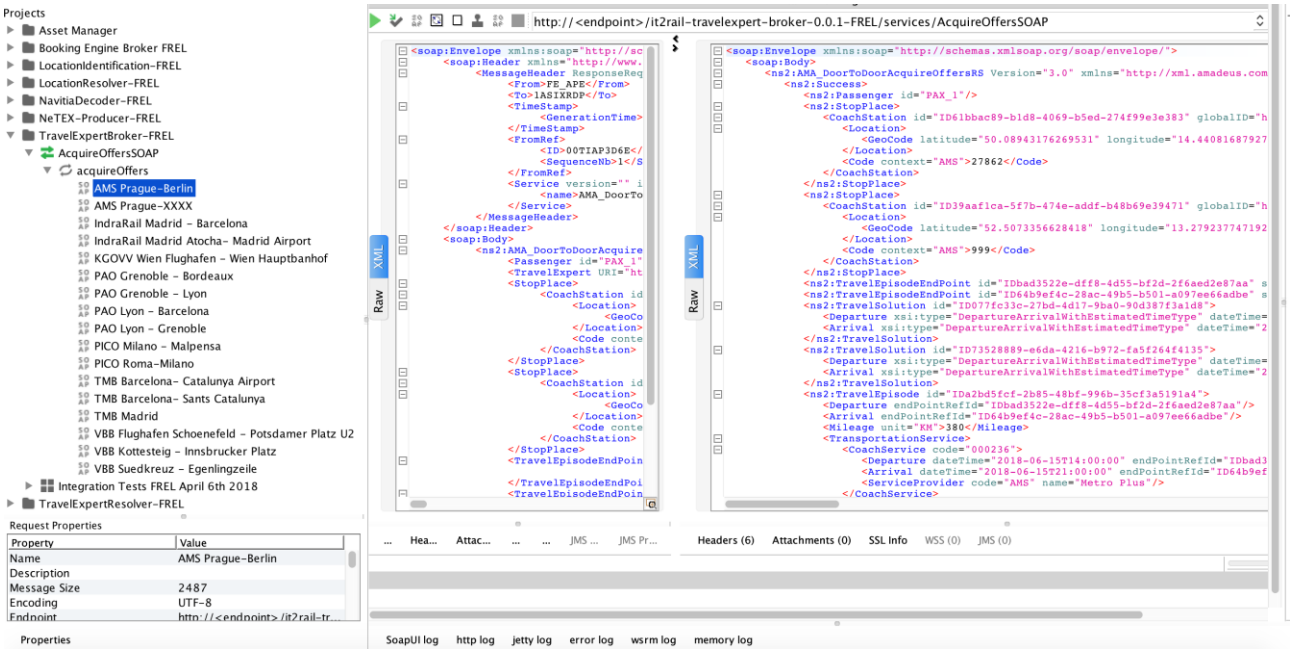


Figure 13 - SoapUI configuration for testing campaign

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

1. SNCF (mainline French Rail) PAO services
    a. <endpoint>/it2r/sales/searchSolutions
2. AMS (long distance Coach operators, Czech Republic) eshopcv services
    a. <endpoint>/v1/Connection
    b. <endpoint>/v1/ConnectionInfo

3. Trenitalia (mainline Italian Rail) PICO Services
    a. <endpoint>/Sale/SaleProcess/SolutionEngine/TravelSolution/search
    b. <endpoint>/Sale/SaleProcess/SaleCoordinator/searchBase
4. 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
5. KGOVV (Austrian Public Transport), HaCon services
    a. <endpoint>/openapi/vao/restproxy/trip
6. TMB (Madrid, Barcelona Public Transport) Indra Rail services
    a. <endpoint>/HMI2_APP/service/otp/getRoute
7. 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:

1. 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
2. AMS (long distance coach services), eshopcv services
    a. <endpoint>/v1/SeatBlock/
    b. <endpoint>/v1/Ticket/
3. Trenitalia (main line Rail operator), PICO Services
    a. <endpoint>/Sale/SolutionEngine/CatalogReservation
    b. <endpoint>/Sale/SaleProcess/OrderProcess

4. 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
5. VBB (Public Transport Berlin/Brandenburg) HaCon services
    a. <endpoint>/shopping/ShoppingMessages/VBB/purchaseRequest
    b. <endpoint>/ shopping/ShoppingMessages/VBB/retrieveRequest

Test suites defined in the SoapUI tool have been created to exercise the interoperability framework with the external Travel Expert and Booking Engine.

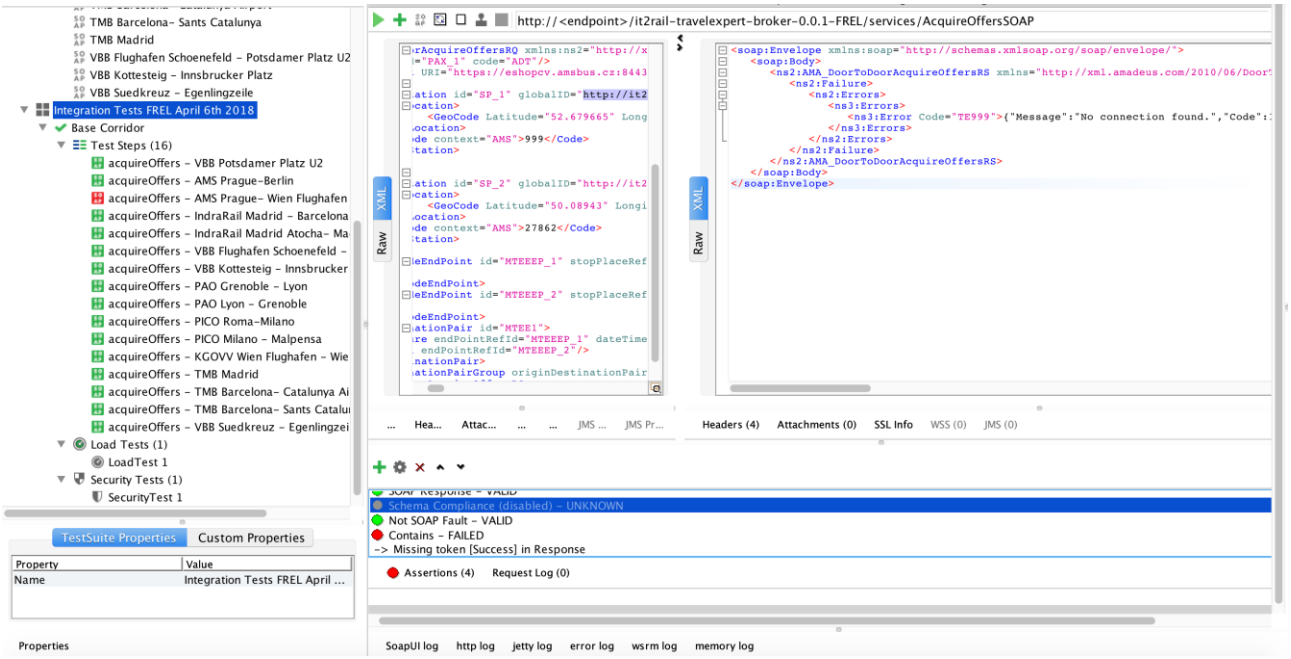


Figure 14 - Travel Expert Broker automated test

In the example shown in Figure 14, 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.

5.1.1 IT2Rail implementation and deployment main features

One of the most important requirements in the design of the IF is its ability to be extensible and to be deployed in multiple instances on a variety of different run time environments. Extensibility permits the development and deployment of new services – e.g., resolvers – by means of appropriate configuration parameters, deployment of multiple instances provides a measure of horizontal scalability and high availability, and deployment on multiple runtime environments allows a degree of transparency to an organization with respect to other tooling, such as logging, security, configuration, or operations management (which it may already be using on a local or cloud-based system platform).

In order to support extensibility, a common Semantic Graph Manager component was developed grouping all methods required to serialize/deserialize java classes to/from RDF graphs, and to persist and query such graphs in the IF's data layer, namely its triple store. This component uses dependency injection, based on configuration files, to obtain specific behavior, namely, to select the specific data sources in the data layer, the provider of template SPARQL queries to be used on the data sources, and the concrete implementations of an abstract interface to converters, which are also injected from the Asset Manager repository. A resolver is essentially obtained by the specification of this configuration, the writing of appropriate SPARQL query templates and, where required, the development and publication on the Asset Manager of specific converters.

The ability to deploy multiple instances of services on different runtime environments is obtained through a specific packaging of the software as Web Application Resource (WAR) and Java Archive (JAR) files, using Apache Maven dependency management.

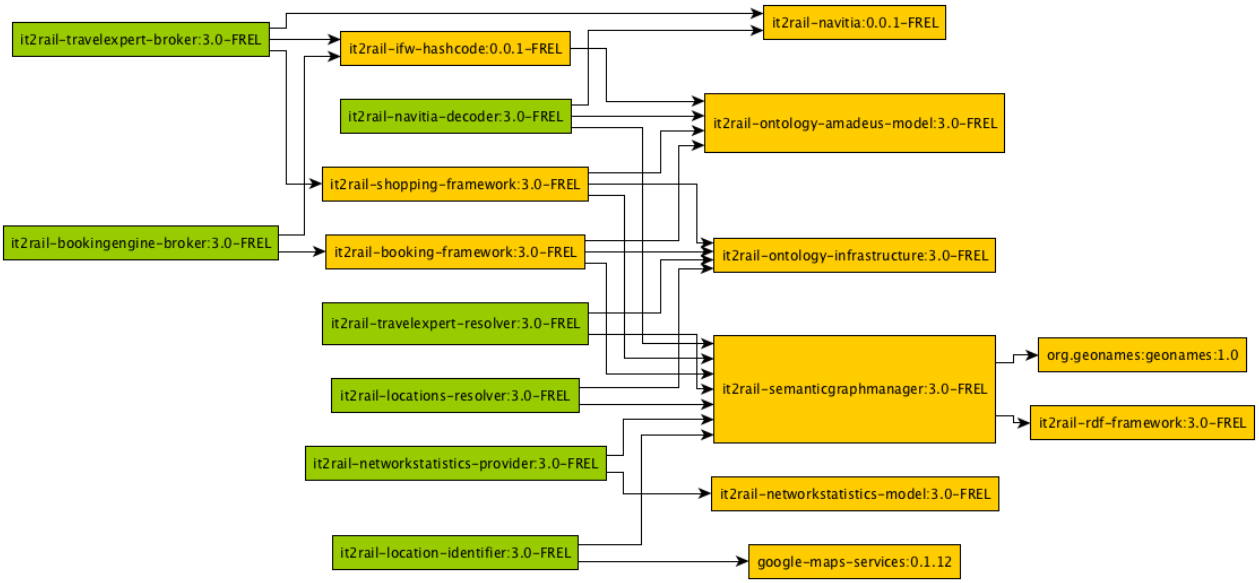


Figure 15 - IT2Rail services packaging

Figure 15 shows, in green the WAR files implementing IF Resolver services, and in orange 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.

The IT2RAIL design, implementation and packaging is described in more detail in the project’s “D1.8 Proof-of-concept Packaged resolvers Full Features” deliverable[2].

5.1.2 IT2RAIL Findings and Limitations

The IT2RAIL pilot demonstration achieved its stated functional objectives. It also validated its fundamental design decisions, particularly the systematic usage of dependency injection controlled by configuration, the ability to use concrete implementations of abstract converters obtained from the Asset Manager at runtime, and the packaging into self-contained WARs built under Apache Maven dependency management using underlying common JARs.

It also stressed the need for better tools to support the ontology annotation process, which is currently manual and labor-intensive. While the handling of security protocols was delegated by design to the specific runtime environment, a serious security vulnerability was identified, but not solved in the course of the project, in the runtime injection of converters from the Asset Manager, i.e. for preventing the injection of malicious or defective implementations in the execution of the services. This requires the development of security mechanisms at the Asset Manager and/or the Semantic Graph Management component, and should additionally be the object of appropriate control procedure in the governance process.

5.2 ST4RT PROJECT

The ST4RT project [2] has delivered semantic conversion technology packaged as a “Converter” software artifact enabling bi-directional mapping of FSM and TAP-TSI messages in a specific use case, i.e. booking of a Berth on Trenitalia night train traveling from Roma Termini to Palermo Centrale stations. This converter has been developed explicitly as an extension of the IF initially delivered by project IT2Rail, in which the extension consisted in the handling of specific FSM and TAP-TSI messages.

The ST4RT Converter transforms a Java object to an RDF graph. The ST4RT project transformation is based on manually annotated Java classes (created through the JAXB framework) which are generated from XSD schemas. Annotations define which elements are processed and how.

The ST4RT Converter does not use physical data storage. All data are processed in memory, except the master data in the OWL file which contains mapping between different

[2] http://www.it2rail.eu/download.aspx?id=a1b4380d-127a-4e03-9bb5-3798d3ef3a5d

codings of the same concept. The Apache Jena framework is already used for reading masterdata from OWL files. Indeed, the use of a datastore/database could have a positive impact on the performance of the converter. The central storage would allow a remote data management and it could lead to better performance in case of complicated conversions. Another possible positive impact of using a central storage could be in the keeping of necessary data during stateful communication, to avoid losing important data which are not required by the second format.

SPARQL is used for building and reshaping the RDF Graph in the ST4RT project.

The ST4RT converter is an independent component. There is a REST interface implemented so the ST4RT converter can be run on an Apache Tomcat server separate from the software which needs to use conversion. The ST4RT converter can be also integrated into existing software without using the REST interface. The ST4RT converter currently does not communicate with other systems –everything it needs is contained inside local packages – though in the future different solutions might be explored.

The implementation of this new converter has been tested in the following scenarios:

a. TAP-TSI ReservationRequest to FSM PreBookingRequest
b. TAP-TSI ReservationReply to FSM PreBookingResponse
c. FSM PreBookingRequest to TAP-TSI ReservationRequest
d. FSM PreBookingResponse to TAP-TSI ReservationReply
e. TAP-TSI ReservationRequest / Reply transaction to an FSM Simulator
f. FSM PreBookingRequest/Response transaction to a TAP-TSI Processor
g. FSM Offering process to IT2Rail Travel Expert Broker for shopping

Since the ST4RT project was explicitly targeted at adding a new converter to the IT2Rail initial implementation of the Interoperability Framework, its deployment and testing uses the same environment – Apache Tomcat web application server, Apache WSO2 Carbon Asset Manager, and OntoText GraphDB triple store – described above.

The ST4RT and IT2Rail artifacts are packaged as web archive (WAR) files that implement the IF web services. Figure 16 shows the deployed services from both projects in the Apache Tomcat management console.

Applications					
Path	**Version**	**Display Name**	**Running**	**Sessions**	**Commands**
/fsm-booking-engine	*None specified*	FSM Booking Engine	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 60 minutes
/graphdb-workbench-free	*None specified*	It2Rail repository sparql endpoint	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/it2rail-bookingengine-broker-3.0-FREL	*None specified*	Booking Engine Broker F-REL	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 60 minutes
/it2rail-fsm-ofering-broker-1.0	*None specified*	tap-booking-service	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 60 minutes
/it2rail-location-identifier-3.0-FREL	*None specified*	Location Identification F-REL	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/it2rail-locations-resolver-3.0-FREL	*None specified*	Location Resolver F-REL	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 60 minutes
/it2rail-navitia-decoder-3.0-FREL	*None specified*	Event Source Resolver - Navitia Decoder F-REL	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 60 minutes
/it2rail-netex-provider-3.0-FREL	*None specified*	NeTEX provider F-REL	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 60 minutes
/it2rail-travelexpert-broker-3.0-FREL	*None specified*	Travel Expert Broker F-REL	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 60 minutes
/it2rail-travelexpert-resolver-3.0-FREL	*None specified*	Travel Expert Resolver F-REL	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 60 minutes
/st4rt-convertor-service	*None specified*	st4rt-conversion-service	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 60 minutes
/tap-tsi-booking-engine	*None specified*	TAP-TSI Booking Engine	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 60 minutes
/tap-tsi-processor	*None specified*	TAP-TSI Processor	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 60 minutes

Figure 16 - Deployed integrated IT2Rail ST4RT converter services

Items highlighted in red in the leftmost column are specific artifacts created for the technical demonstrator of the ST4RT converter in the IT2Rail interoperability framework:

1. /fsm-booking-engine uses the ST4RT converter to generate a full FSM PreBookRequest / FSM PreBookResponse session using a TAP-TSI processor
2. /it2rail-fsm-offering-broker-1.0 uses the IT2Rail rdf framework upgraded with a merging of annotation features developed by the ST4RT project to support the FSM Offering process with the SNCF, Trenitalia, AMS, VBB and IndraRail Travel experts used in the IT2Rail proect
3. /st4rt-convertor-service demonstrates the ST4RT converter in the federated graph environment
4. /tap-tsi-booking-engine generates a full TAP-TSI ReservationRequest / ReservationReply session using the remote FSM Simulator provided by OLTIS Group
5. /tap-tsi-processor creates a TAP-TSI Reservation Reply in response to a TAP-TSI Reservation request and is used in conjunction with the /fsm-booking-engine service described in point 1 above

Since test scenarios are exposed as web services, the test campaign is performed with the use of the open-source SoapUI web service testing application.

Figure 17 shows the SoapUI set up for IF testing.

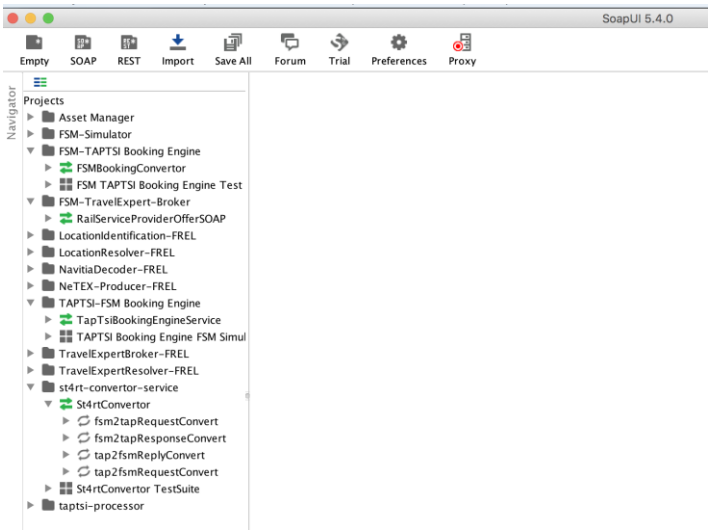


Figure 17 - SoapUI projects for Interoperability Framework testing

The four expanded projects on the leftmost column show, marked by a green icon, the four specific web service bindings corresponding to the four test scenarios implemented in the demonstrator.

In addition, two other “tools” are used in the test campaign: an FSM Simulator developed by OLTIS Group and deployed on their servers, and a TAP-TSI Processor developed by Trenitalia to provide actual TAP-TSI Reservation replies TAP-TSI Reservation request for the specific ST4RT use case.

Detailed outcomes from the campaign test are described in ST4RT project deliverable “D5.5 Report on the results of the IT2Rail semantic broker demonstration scenario”[3].

5.2.1 ST4RT project implementation and design features

The ST4RT demonstrator was explicitly developed as an extension of the IT2RAIL project as dedicated specific converter to support bi-directional FSM/TAP-TSI exchanges in a booking scenario. As such, its main design requirement was the ability to be injected at runtime in the Semantic Graph Manager. However, an additional requirement was added to allow it to be run within the HEROS middleware provided by project partner HitRail. Since the HEROS middleware has no access to IF’s Asset Manager or the Triple Store in the Data layer, the Semantic Graph Manager’s configuration capabilities were exploited to have local files play the role of Ontology Repository and Data layer. When the ST4RT converter runs

[3] Cfr. http://www.st4rt.eu/download.aspx?id=27b9e8c3-1cc3-48c0-9d76-6b4a6e02ac51

in this configuration, it is statically bound to the Semantic Graph Manager to access the ontology and the local RDF graphs. The ST4RT converter can therefore be run both in a stand-alone mode, using the Semantic Graph Manager as an underlying utility, or within the Semantic Graph Manager when running in the IF's context, the modes being controlled by the appropriate configuration.

5.2.2 ST4RT Findings and Limitations

The ST4RT pilot demonstrator also achieved its stated objectives, and in fact a second demonstration scenario was added to test the Converter in both the HEROS and IF deployment environments. From an architectural standpoint its main finding is that it is indeed possible to develop specialized converters that run in the IF without changes to the framework itself. These converters can therefore be packaged as independent artifacts to be injected as required.

However, due to the nature of the TAP-TSI specification, the annotation model had to be extended in order to perform complex semantic inferences through SPARQL queries. While this is a powerful extension the capabilities of semantic conversion, it adds considerably to the manual effort of the annotation process and in its debugging. It is necessary therefore to develop appropriate productivity tools to support it.

6. DISCUSSION

This document aimed at overviewing and analyzing the current state of the art and best practices covering various architectural aspects of interoperability. Section 2 introduced the current generic architecture of the IF and its main elements (which will be further elaborated in the next steps of the SPRINT project). It also briefly described the motivations behind the design, and it overviewed the functions provided by the components and the relevant technologies and tools for their implementation. Table 1 summarizes the different components of the IF versus their respective possible technologies, tools and patterns.

Table 1 Summary of tools, techniques and technologies suitable for different components of IF

Tools Patterns and Technologies	Interoperability Framework Functions					
	Data Layer	Asset Discovery and Registry	Life Cycle Management	Distributed SPARQL endpoint	Converter	Resolver
BPMN			✓			
Camel					✓	
Camunda			✓			
Containers: Docker					✓	
Graph DB: Neo4j, Neptune	✓			✓		
HATEOS: HAL, Hydra, Siren		✓			✓	✓
Linked data	✓	✓				
Mash-Up		✓			✓	✓
OSGi					✓	
Orchestration: Kubernetes, Azure, Google Container					✓	✓
Semantic Annotation: WSDL-S, SAWSDL, WSML, SA-REST, hREST		✓				

Triple Store data bases RDF4J Jena [2], Jena	✓			✓	✓	

Section 3 studied and analyzed architectural patterns, especially for the construction of distributed systems. It presented the benefits of each pattern for the development of the IF itself and of the individual components – if applicable. In addition, it identified the cutting-edge technologies currently employed for the development of such patterns are (which are listed in Table 1)

Table 2 Comparison of different Deployment Architectures

Evaluation Dimensions	**Deployment Architecture**				
	Middleware	Microservices	Modular	P2P	cloud
Maintenance over time.					
Cost					
Horizontal Scalability.					
Vertical Scalability.					
Resilience to Failure.					
Technology Independency Across Stakeholders.					
Loose Coupling					
Agile Development					
Security					

Evidently, each architectural style has its own advantages and shortcomings. To select one (or more) approach and balance the tradeoff between its pros and cons, we identified the aspects which are most relevant and pivotal in our case and assess the different patterns against them. Table 2 summarizes the discussions and comparisons of the different architectural style against the selected evaluation dimensions presented in each corresponding section. The green and red colors, respectively, indicate the most and least suitable patterns – if any – with respect to the others.

According to our analysis, modular service-oriented approaches – in general – and microservices – specifically – seem the more promising approaches. The focus of microservices-based architectures is on the decomposition of software into smaller, but

autonomous, components dedicated to a narrow set of functions. The great degree of independence of the various components fosters an agile software development and leads to many advantages including higher scalability and efficiency. The former is achieved due to the reusability of such components. The latter, instead, is a result of selective scalability, which is the ability to scale in/out a specific microservice based on demand (instead of duplicating the whole software) and greatly facilitates both horizontal and vertical scalability. In addition, the overall performance of the system could be greatly enhanced thanks to the possibility of technology optimization for each component. As discussed in Section 3.1.2, the interaction among various components is achieved in a technology-agnostic manner and through standard API (usually RESTful). Accordingly, the best-suited tools, technology, and platform to develop each microservice could be adopted solely based on the characteristics and requirements of that microservice, regardless of the technology that is used for rest of the system.

However, the most important result we reached in our study was that there is not a one-size-fits-all solution. A multidisciplinary approach is needed in order to address various requirements and use-cases. For instance, consider the scenario in which the IF offers a runtime execution environment for – automatic – deployment of a service on-the-fly, versus the simplest use case where the IF mainly plays the role of service registry. In the latter case, it is the responsibility of the TSP to register its service following the proposed vocabulary, annotation and service description language supported by the Asset Manager. In the service discovery phase, the Asset Manager returns to the client the endpoint of the service, which is deployed, run, and maintained at the TSP premises. Presumably, it would be very difficult to address the first scenario without taking advantage of cloud computing as well. In particular, in Section 3.2.2 we discussed in detail how the combination of could infrastructure and microservice-based technologies could address such a requirement.

Section 4 overviewed solutions for tackling the interoperability problem in other domains. More precisely, we focused on IoT and cloud-based frameworks since – similar to our case – both deal with geographically and administratively distributed ecosystem characterized by heterogeneous actors and standards. Furthermore, for the sake of completeness, we analyzed projects that cover various software architectures. This allowed us to analyze and identify the pros and cons of different architectural styles in practice. In particular, modular and service-oriented approaches seemed to be the key to solving the interoperability problem. In addition, the survey greatly helped us to recognize different requirements, critical aspects and common practices for the development of an interoperability framework. For instance, all solutions highlight the necessity and significance of semantic-based approach, and encourage utilization of a shared ontology in order to achieve semantic interoperability in addition to syntactical interoperability.

Finally, Section 5, assessed the results of previous S2R projects that are related to the goals of the SPRINT project. The assessment showed the need to strengthen the security features of the Asset Manager, to avoid for example potential attacks in which malicious code is

injected in the services offered. It also highlighted the need to improve and ease the annotation process and mechanisms developed in the ST4RT project.

REFERENCES

[1] IT2Rail Consortium, “www.it2rail.eu,” [Online].

[2] ST4RT Consortium, “www.st4rt.eu,” [Online].

[3] ISO/IEC, *10746-3:2009 Information technology -- Open distributed processing -- Reference model: Architecture,* 2019.

[4] B. Hanssens, “RDF4J,” 2019. [Online]. Available: http://rdf4j.org/.

[5] “Apache Jena,” [Online]. Available: http://jena.apache.org/index.html.

[6] “neo4j,” [Online]. Available: https://neo4j.com/.

[7] “Amazon Neptune,” [Online]. Available: https://aws.amazon.com/neptune/.

[8] C. Bizer, T. Heath and T. Berners-Lee, “Linked data: The story so far.,” *n Semantic services, interoperability and web applications: emerging concepts,* pp. 205-227, 2011.

[9] E. Prud, A. Seaborne and others, “Sparql query language for rdf,” 2006.

[10] “BPMN,” [Online]. Available: http://www.bpmn.org/.

[11] “Camunda,” [Online]. Available: https://camunda.com/.

[12] I. S. A. and others, *DCAT application profile for data portals in Europe,* 2015.

[13] M. Dekkers, “Asset description metadata schema (adms),” *W3C Working Group,* 2013.

[14] A. Schwarte, P. Haase, K. Hose, R. Schenkel and M. Schmidt, “FedX: Optimization techniques for federated query processing on Linked Data,” in *ISWC2011, Part I,* 2011.

[15] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo and E. Ruckhaus, “ANAPSID: Anadaptive query processing engine for SPARQL endpoints,” in *ISWC2011, Part I,* 2011.

[16] G. Bieber and J. Carpenter, “Introduction to service-oriented programming (rev 2.1),” *OpenWings Whitepaper, April,* 2001.

[17] R. Chinnici, J. J. Moreau, A. Ryman and S. Weerawarana, “WSDL 0.2 Specification,” W3C, 2007. [Online]. Available: https://www.w3.org/TR/2007/REC-wsdl20-20070626/.

[18] J. Domingue, D. Roman and M. Stollberg, “Web service modeling ontology (WSMO)-An ontology for semantic web services,” 2005).

[19] R. Akkiraju, J. Farrell, J. A. Miller, M. Nagarajan, A. P. Sheth and K. Verma, *Web service semantics-wsdl-s,* 2005.

[20] wikipedia, “Hypermedia,” [Online]. Available: https://en.wikipedia.org/wiki/Hypermedia.

[21] “Hydra,” [Online]. Available: http://www.markus-lanthaler.com/hydra/.

[22] “HAL,” [Online]. Available: http://stateless.co/hal_specification.html.

[23] “SIREN,” [Online]. Available: https://github.com/kevinswiber/siren.

[24] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Manuel, M. Fabrizio, M. Ruslan and S. Larisa, “Microservices: yesterday, today, and tomorrow,” *n Present and ulterior software engineering,* pp. 195-216, 2017.

[25] “VMware,” [Online]. Available: https://www.vmware.com.

[26] “VirtualBox,” [Online]. Available: https://www.virtualbox.org/wiki/VirtualBox.

[27] S. Newman, Building microservices: designing fine-grained systems., O'Reilly Media, Inc., 2015.

[28] D. Bernstein, “Containers and cloud: From lxc to docker to kubernetes.,” *IEEE Cloud Computing 1, no. 3,* pp. 81-84, 2014.

[29] “Docker,” [Online]. Available: https://www.docker.com/). .

[30] “Linux containers,” [Online]. Available: https://linuxcontainers.org/.

[31] “Kubernetes,” [Online]. Available: https://kubernetes.io.

[32] K. Knoernschild, ava application architecture: modularity patterns with examples using OSGi, Prentice Hall Press, 2012.

[33] “OSGi,” [Online]. Available: https://www.osgi.org/.

[34] “Camel,” [Online]. Available: http://camel.apache.org/.

[35] G. Hohpe and B. Woolf, Enterprise integration patterns: Designing, building, and deploying messaging solutions., Addison-Wesley Professional, 2004.

[36] G. Hohpe and B. Woolf, “Content-Based Router,” [Online]. Available: https://www.enterpriseintegrationpatterns.com/patterns/messaging/ContentBasedRouter.html.

[37] G. Hohpe and B. Woolf, “Content Enricher,” [Online]. Available: https://www.enterpriseintegrationpatterns.com/patterns/messaging/DataEnricher.html.

[38] P. Banerjee, R. Friedrich, C. Bash, P. Goldsack, B. Huberman, J. Manley, C. Patel, P. Ranganathan and A. Veitch, “Everything as a service: Powering the new information economy,” *Computer,* pp. 36--43, 2011.

[39] P. Mell and T. Grance, “https://www.nist.gov/programs-projects/nist-cloud-computing-program-nccp,” [Online]. Available: https://www.nist.gov/sites/default/files/documents/itl/cloud/cloud-def-v15.pdf.

[40] “Heroku,” [Online]. Available: https://www.heroku.com/.

[41] S. Hietanen, “Mobility as a Service,” *the new transport model* , pp. 2-4, 2014.

[42] R. Steinmetz and K. Wehrle, “What Is This “Peer-to-Peer” About?.,” in *In Peer-to-peer systems and applications*, Berlin, 2005.

[43] S. Saroiu, P. K. Gummadi and S. D. Gribble, “Measurement study of peer-to-peer file sharing systems.,” *Multimedia computing and networking,* pp. ol. 4673, pp. 156-171, 2002.

[44] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek and H. Balakrishnan, “Chord: a scalable peer-to-peer lookup protocol for internet applications,” *IEEE/ACM Transactions on Networking (TON),* pp. 17--32, 2003.

[45] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, “A scalable content-addressable network,” *ACM,* 2001.

[46] A. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems,” in *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, Springer, 2001, pp. 329--350.

[47] “Napster,” [Online]. Available: http://www.napster.com/.

[48] E. Meshkova, J. Riihiarvi, M. Petrova and P. Mahonen, “A survey on resource discovery mechanisms peer-to-peer and service discovery frameworks,” *Computer networks 52, no. 11,* pp. 2097-2128, 2008.

[49] B. Furht and A. Escalante, Handbook of cloud computing, New York: Springer, 2010.

[50] S. Jo and J. Han, “Convergence P2P cloud computing,” *Peer-to-Peer Networking and Applications,* pp. 1153--1155, 2018.

[51] V. Sinha, A. Gupta and G. S. Kohli, “Comparative Study of P2P and Cloud Computing Paradigm Usage in Research Purposes,” in *In International Conference on Advances in Communication, Network, and Computing*, Berlin, Heidelberg, 2011.

[52] D. P. Anderson and G. Fedak, “The computational and storage potential of volunteer computing,” in *In Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), vol. 1*, 2006.

[53] “bIoTope,” [Online]. Available: https://biotope-project.eu/.

[54] N. Kolbe, S. Kubler, J. Robert, Y. Le Traon and A. Zaslavsky, “Towards semantic interoperability in an open IoT ecosystem for connected vehicle services,” in *2017 Global Internet of Things Summit (GIoTS)*, IEEE, 2017.

[55] bIoTope, “D3.5 Prototype of Platform Integration using API Mediators,” [Online]. Available: https://biotope-project.eu/results.

[56] J. Robert, S. Kubler, N. Kolbe, A. Cerioni, E. Gastaud and K. Framling, “Open IoT ecosystem for enhanced interoperability in smart cities—Example of Metropole de Lyon,” *Sensors*, vol. 17, p. Multidisciplinary Digital Publishing Institute, 2017.

[57] “openIoT,” [Online]. Available: http://www.openiot.eu/.

[58] “OpenIoT Project,” [Online]. Available: https://github.com/OpenIotOrg/openiot.

[59] M. Compton, P. Barnaghi, L. Bermudez, R. Garcia-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog and others, “The SSN ontology of the W3C semantic sensor network incubator group,” *Web semantics: science, services and agents on the World Wide Web*, pp. 25--32, 2012.

[60] J. Soldatos, N. Kefalakis, M. Hauswirth, M. Serrano, J.-P. Calbimonte, M. Riahi, K. Aberer, P. P. Jayaraman, A. Zaslavsky, I. P. Zarko and others, “Openiot: Open source internet-of-things in the cloud,” in *Interoperability and open-source solutions for the internet of things*, Springer, 2015, pp. 13--25.

[61] N. Loutas, E. Kamateri, F. Bosi and K. Tarabanis, “Cloud computing interoperability: the state of play,” in *IEEE Third International Conference on Cloud Computing Technology and Science*, 2011 .

[62] R. Rezaei, T. K. Chiew, S. P. Lee and Z. S. Aliee, “A semantic interoperability framework for software as a service systems in cloud computing environments,” *Expert Systems with Applications*, Vols. 5751--5770, no. Elsevier, 2014.

[63] M. author, “my paper,” *my journal*.