Contract No. H2020 – 826172

SEMANTICS FOR PERFORMANT AND SCALABLE INTEROPERABILITY OF MULTIMODAL TRANSPORT

D4.2 A lightweight solution to automate the generation of ontologies, mappings and annotations (C-REL)

Due date of deliverable: 31/10/2019

Actual submission date: 24/04/2020

Leader/Responsible of this Deliverable: UPM

Reviewed: Y

Document status		
Revision	**Date**	**Description**
1	01/10/2019	Agreed table of contents
2	27/10/2019	Initial texts for all sections except for automation of mappings
3	5/11/2019	Final texts for all sections except for automation of mappings
4	15/11/2019	Inclusion of section on automation of mappings and document ready for QA
6	21/11/2019	Final version after TMC approval and Quality Check
7	17/04/2020	Final QA
8	24/04/2020	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 describes the current status in the management of assets in the context of Shift2Rail, paying special attention to the development process of ontologies, mappings and annotations that are being created with the focus of achieving interoperability across data sources, as part of the implementation of the Shift2Rail Interoperability Framework.

Based on this analysis of the current status of such asset management and development process, several approaches are explored and designed for helping in the automation of some of the tasks involved in this process. Therefore, a proposal for the collaborative engineering of ontologies for Shift2Rail projects is made, based on existing standards for collaborative ontology engineering in the state of the art, used in many other domains. A solution for providing a higher degree of automation in ontology engineering is also provided, based on the fact that many of the data formats for the data sources to be used are available in XML and XML Schema (e.g., NeTEx), making use of existing tools for XML Schema to ontology generation, and the need for further refinement once the initial steps of ontology creation are done. A potential approach for the automatic generation of mappings, using a range of machine learning techniques, is also presented as a potential solution for the automation of some of the tasks related to the development of mappings and annotations to bridge the gap between existing data sources and the developed ontologies.

Finally, the deliverable discusses on the automation of the asset management lifecycle, based on the use of a number of artifacts and systems commonly used in the context of software engineering and in the context of workflow management processes.

All the techniques and proposals that are presented in this deliverable are in the process of being implemented and deployed, with different degrees of achievement of the planned processes, and will continue to be developed and fully tested in the remaining period of work of the project, and applied to the context of Shift2Rail to become the reference implementation of techniques for asset management, ontology development and mapping and annotation creation.

ABBREVIATIONS AND ACRONYMS

Abbreviation	**Description**
API	Application Programming Interface
AVMS	Automatic Vehicle Monitoring
BPMN	Business Process Model and Notation
C-REL	Core Release
CCTV	Closed-circuit television
CRM	Customer Relationship Management
DCAT	Data Catalog Vocabulary
DNS	Domain Name System
DNS-SD	DNS Service Discovery
DPI	Dynamic passenger information
DRM	Driver Relationship Management
EBSF	European Bus System of the Future (EU-funded project)
EIF	European Interoperability Framework
EU	European Union
FMS	Vehicle Fleet Management System
FSM	Finished State Machine
FOAF	Friend of a friend is a machine-readable ontology
H2020	Horizon 2020 framework programme
HTTP	HyperText Transfer Protocol
IF	Interoperability framework
IP	Internet Protocol
IP4	Innovation Program 4
ISO	International Organization for Standardization
ITxPT	Information Technology for Public Transport
ITS	Information
JSON	JavaScript Object Notation

MaaS	Mobility as a Service
NAP	National Access Point
ORM	Object Relational Mapping
RDF	Resource Description Framework
REST	Representational State Transfer
S2R	Shift2Rail Joint Undertaking
SCXML	State Chart XML
SOAP	Simple Object Access Protocol
SPARQL	Protocol and RDF Query Language
WP	Work Package
XML	eXtensible Markup Language
XSD	XML Schema Definition

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This deliverable describes a set of solutions to automatically generate relevant assets required for the conversion/translation of heterogeneous messages exchanged by different systems, and how to manage them ensuring the consistency of an IF-based ecosystem.

The concept of “automation” is indeed applied in its wider meaning in this deliverable. Automating the generation of ontologies, mappings and annotations, as well as the management of assets in the asset manager, means providing tools to cover the entire development lifecycle of such assets. This means helping developers in creating ontologies by analyzing heterogeneous schemas and providing half-processed assets, ready to be perfectioned. Automation means also helping developers in working collaboratively, allowing a consistent shared editing of ontologies. It means also helping an IF-based ecosystem in managing the lifecycle of different types of assets, defining processes which can coordinate human tasks and services with the aim of allowing the evolution of existing assets while maintaining a global consistency and minimizing deployment errors.

An IF-based ecosystem may comprise a wide set of asset types. In this deliverable, we are focused on solutions tailored to ontologies, mappings and annotations, as well as converters as artifacts ready to be deployed.

Several state-of-art tools have been studied and evaluated to be appropriately incorporated into IF. Ontology development is an activity more and more frequent in the software development and has been separately evolved from software development practices. As a result, ontology development can take a long time and effort. In consequence, there are methodologies, methods and tools that help to automate and generate mappings, ontologies and annotations. Additionally, ontology developers need to evaluate the quality of the generated ontologies during their development process, so as to identify potential errors before making them public as a new release. Due to the parallel development of the ontology network by different ontology developers, the team needs to make any quality report public to all its members, so they are aware of any potential issue with any published ontology.

In this context of collaborative ontology development, it is also important to facilitate the automation of ontology generation from existing non-ontological resources available in different formats. The focus in this deliverable is on exploiting XML schemas so as to generate initial ontologies from them, using state-of-the-art tools created for this purpose, and whose results need to be refined so as to produce ontologies with no errors.

Automation in the the creation of mappings is also relevant in an OBDA context like the one proposed for the access to heterogeneous data sources in heterogeneous data formats. Annotations can be created manually or automatically, and in the context of our automation goals we make proposals to improve the creation of such annotations in an automatic manner, using machine learning mechanisms.

Finally, existing state-of-the-art tools for automating the processes associated to the management of the lifecycle of assets are discussed in the deliverable, focused on the workflow engineering and software engineering processes commonly used in other contexts.

2. CURRENT STATUS OF THE ONTOLOGY, MAPPINGS AND ANNOTATIONS DEVELOPMENT PROCESS, AND ASSET MANAGEMENT, IN SHIFT2RAIL

This section provides an overview of the current status of ontologies in the context of Shift2Rail (section 2.1), as well as mappings and annotations that have been created for some of the projects involved in this initiative (section 2.2). We also provide a brief description of the current asset management process and tools (section 2.3).

It is important to consider that, according to GOF4R deliverable D5.1 (Deployment Roadmap), ontologies, schemas and mappings should be considered as open source, and hence an open source management of these artifacts is expected to be applied.

2.1 THE SHIFT2RAIL ONTOLOGY NETWORK

Some ontologies have been already developed in the context of ongoing or already finished Shift2Rail projects. Evidence of the development of such ontologies are provided in the following Shift2Rail project-related deliverables:

- IT2Rail deliverable D1.1 (IT2RAIL Domain Ontology Specification and Repository), delivered on October 2017. This deliverable is the main one covering the IT2Rail ontology, as well as the development process followed to create such ontology.
- IT2Rail deliverable D2.7 (Travel Shopping Ontology Document), delivered on May 2018. This deliverable describes the part of the IT2Rail ontology that covers the travel shopping use cases.
- IT2Rail deliverable D3.7 (Booking & Ticketing Ontology document), delivered on November 2018. This deliverable describes the part of the IT2Rail ontology that covers the booking and ticketing use cases.
- IT2Rail deliverable D4.7 (Trip Tracker ontology document), unavailable from the IT2Rail website. This deliverable should describe the part of the IT2Rail ontology that covers the trip tracker use case.
- IT2Rail deliverable D5.7 (Travel Companion ontology document), delivered in November 2017. This deliverable describes the part of the IT2Rail ontology that covers the travel companion use case.
- IT2Rail deliverable D6.7 (Business Analytics ontology document), delivered in November 2017. This deliverable describes the part of the IT2Rail ontology that covers the business analytics use case.
- ST4RT deliverable D3.2 (Extension to the S2R reference ontology), delivered on November 2018. This ontology was originally supposed to extend the IT2Rail reference ontology, including all semantic concepts from both the Full Service Model (FSM) XSD and the XSD definitions used by Railway Undertakings (TAP TSI 918 XML). However, a decision was made to create an “Integrated ST4RT Ontology” that also made changes to the original IT2Rail reference ontology, refining and modifying some of its concepts. The development of this ontology was specifically focused on allowing issuers that have implemented the FSM standards to request a reservation from a Railway Undertaking (RU) that have only implemented the TAP 918 XML standard. Figure 1 shows a UML class diagram with the most

relevant concepts from this integrated ontology, where the classes colored black are those newly introduced in the Integrated ST4RT Ontology, whereas those colored in red are imported from the IT2Rail ontology.

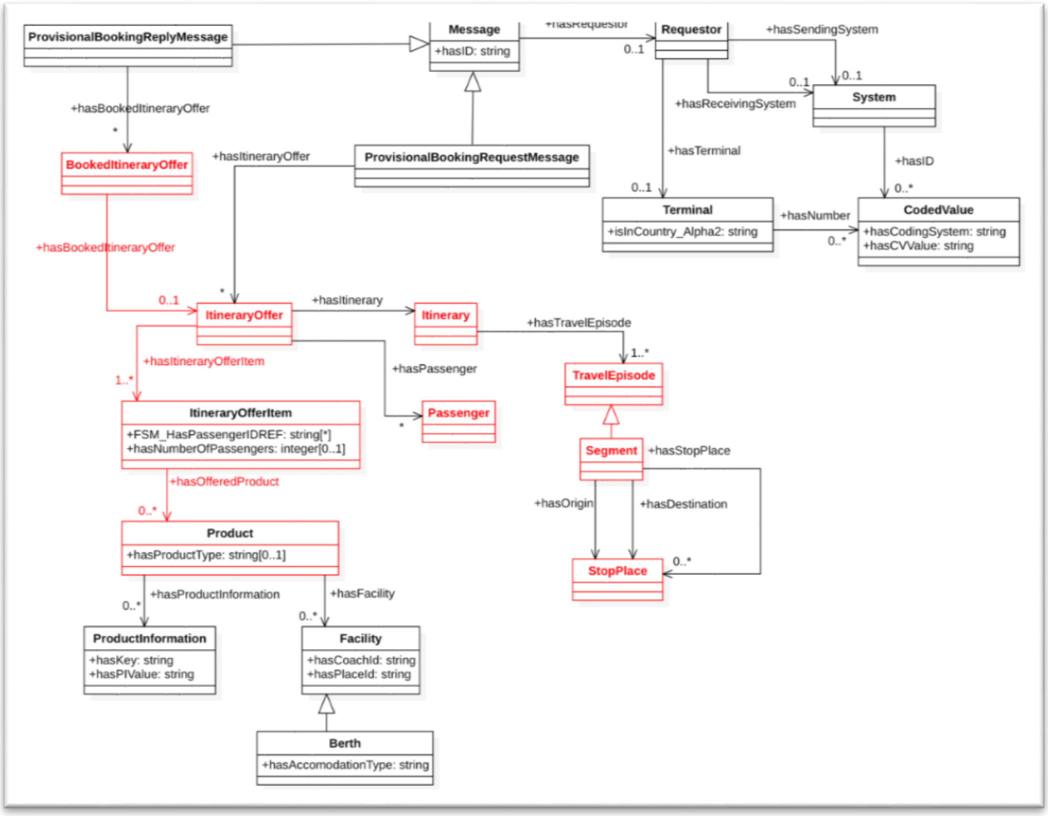


Figure 1. Excerpt from the Integrated ST4RT Ontology (from ST4RT deliverable D3.2)

These ontologies are not published yet according to best practices for ontology development. That is, they are not available as de-referenceable entities in their corresponding URIs (http://www.it2rail.eu/ontology/ and http://st4rt.eu/ontologies/st4rt). Their implementation in OWL is however available associated to the corresponding deliverables (this is the case for the integrated ST4RT ontology), or via user and password registration at https://it2rail.ivi.fraunhofer.de/webprotege/ (in the case of the IT2Rail ontology).

Furthermore, there is no evidence, from reading those deliverables, about the usage of state-of-the-art ontology engineering methodologies or processes in the development of these ontologies, nor of a principled process for the development and maintenance of the Shift2Rail ontology network. IT2Rail describes the process followed to create the ontology without making any mention to existing ontology engineering methodologies (using Excel spreadsheets for collecting terms, pre-defined Word documents to describe some extra information about those terms, and UML class diagrams to represent the conceptual model of the ontologies). With respect to change management, a discussion on the use of Mantis for issue tracking is provided, but no clear references to a public pointer for this Mantis system is given, nor how this process is governed.

Indeed, the aforementioned ST4RT deliverable explicitly mentions the fact that the governance process of the Interoperability Framework (IF) Assets defined by the GoF4R project should be the ones applied for managing the requests of changes in the IT2Rail ontology and for the publication of the Integrated ST4RT ontology as a new IF asset. But this has not been done so far, something that we are addressing in the context of this project.

2.2 THE SHIFT2RAIL SET OF MAPPINGS AND ANNOTATIONS

The usage of mappings and annotations for the lifting and lowering processes to be done in the context of Shift2Rail projects is identified in a good number of deliverables, including the general overview of the interoperability framework with the inclusion of converters as a key component in the framework.

In the ST4RT deliverable D4.1 (Analysis of state-of-the-art ontology conversion tools) several tools are identified that can be used for the manual or semi-automatic generation of annotations, even including systems that can be used for the annotation of texts, what is considered out of the scope of SPRINT. In the context of ontology-based annotation for XML documents and Java objects, which is more in the scope of what we are trying to deal with in SPRINT, several key components/tools were identified:

- The EMPIRE framework, a “widely-known Java persistence framework for use in Semantic Web projects where data is stored in RDF and accessible through SPARQL queries. By providing an implementation of Java Persistence API (JPA) and using it to abstract the minutiae of RDF, it lowers the learning curve for new developers and helps provide a straightforward path for migrating or enhancing existing traditional web applications to use semantic technologies”.
- The PINTO framework, “a lightweight framework that provides mappings for RDF statements into POJOs and viceversa. Pinto offers a mechanism to convert POJOs to RDF triples and vice versa. The offered mechanism can exploit specific annotations of POJO methods and properties.”

In both cases, several types of annotations are declared with @RdfId, @Rdfproperty, @Rdfsclass and @Iri.

Then, the ST4RT deliverable D3.1 (Annotations for Mapping from Legacy Data Models to Ontologies) provides several examples of the types of annotations used by the converter, both for lifting and lowering, which have been manually generated. The examples provided in this deliverable contain both simple and complex mappings and transformations.

However, there is no central repository where all these annotations are maintained, even though they are considered to be open source. And given the fact that they are inserted inside Java code as annotations, using the aforementioned frameworks, the maintenance of such annotations and mappings is not trivial and may require clear improvements.

2.3 THE SHIFT2RAIL ASSET LIFECYCLE MANAGEMENT

Asset lifecycle management in the context of the Shift2Rail Interoperability Framework means the possibility to share, advertise and govern the access to different assets involved in connecting different services provided by different and competing Transport Operators. A first solution to this problem has been proposed in the IT2Rail project. The CMS-like solution called Asset Manager proposed two different Web interfaces, one devoted to accessing assets and the other one dealing with contributing to the IF ecosystem by publishing information. The governance problem managed via the Asset Manager implied defining processes to model the lifecycle of the description of an asset. Such processes were defined in the IT2Rail Asset Manager via Finished State Machines defining the possible states of the information describing the Asset, and the possible transitions between them. Such approach was evaluated in the context of IT2Rail but was not widely used in the larger context of the Interoperability Framework-related projects due to the lack of a well-established governance process. In Section 4 we will describe how lifecycle management was implemented in IT2Rail, its limitations and the planned improvements in the SPRINT Asset Manager.

3. AUTOMATION HELPING THE ONTOLOGY, MAPPINGS AND ANNOTATIONS DEVELOPMENT PROCESS

3.1 COLLABORATIVE ONTOLOGY ENGINEERING

3.1.1 State of the art

Several ontology engineering methodologies have been defined in the past decades to transform the initial ontology development processes, which were closer to an art, into a well docuemented and planned engineering activity. Some examples of the most relevant and widespread ontology development methodologies that have been proposed over these decades are: METHONTOLOGY [1], On-To-Knowledge [2] and the NeOn methodology [3]. All these methodologies are strongly based on the assumption that ontology engineering should be considered as a development process that is similar to that of software engineering, and hence many of the steps followed for ontology engineering, as well as the development lifecycles, may be similar to those used in software engineering. Further explanations of all these methodologies can be found in [4] and [5]. Taking into account the context of the Shift2Rail ontology development processes and the expected products (a network of ontologies), the most appropriate methodology to be applied in this context will be the NeOn methodology, which considers as a default that the product of an ontology engineering process should be a network of interconnected ontologies. Therefore, this will be the methodology that will be proposed in the context of Shift2Rail ontology development.

Besides, the Shift2Rail Joint Undertaking proposes a clear governance and structure for open source components, among which ontologies should be considered, where collaborative ontology development needs to be considered. Therefore, in this initial state of the art analysis we consider as well those tools have been built to support collaborative ontology development processes, such as WebProtégé, Moki, VoCol and OnToology.

- WebProtégé [6] facilitates online collaborative ontology development by means of editing, annotating, documenting and visualising ontologies. Indeed, WebProtégé has been already used for the development of the IT2Rail reference ontology, as described in the IT2Rail deliverable D1.1. Unfortunately, WebProtégé does not integrate features for the online publication of ontologies, nor does it support clearly all the steps in the Shift2Rail IF governance process for open-source assets.

- Moki [7] allows modelling ontologies by using the MediaWiki environment. This ontology development system is characterized by functionalities such as a model checklist, quality indicators and ontology publication. However, it does not provide any documentation functionalities, and the management of changes, although handled as in any other wiki, does not follow the principles commonly followed in software engineering contexts (such as those supported by software versioning systems like svn or git).

- VoCol [8] is closer to the needs expected for ontology development processes in Shift2Rail. It supports collaborative ontology development and uses Git repositories to maintain ontology-related files. VoCol provides support for project management, quality assurance, documentation and visualization components. And it also provides support for publishing ontologies and their documentation. Therefore, it would be a potential good candidate for the development of ontologies in Shift2Rail.

- OnToology [9] also integrates a set of tools for the continuous development of ontologies, using also Git repositories to maintain all the ontology-related files and associated documentation. The development of OnToology has been based on the idea of benefiting from best practices from both the software and ontology engineering communities. Instead of allowing direct online edition of ontologies, as it happens with the previous tools, it considers that ontology developers will work offline with tools like Protégé and will upload new versions of the ontology files (in OWL) as it happens in common software development activities.

3.1.2 Methodological and technological proposal for collaborative ontology engineering applied to Shift2Rail

As described in GOF4R deliverable D5.1 (Deployment Roadmap), “the IF governance structure and processes for open-source assets should not only allow stakeholders to modify assets prudently, but it should also be agile, so that modifications can be easily introduced, assuring the overall dependability of the IF”.

Given the fact that the ontologies to be developed in the context of Shift2Rail are considered as open source assets, the following steps, already identified in the aforementioned deliverable, should be considered:

1. Either a stakeholder (a member of the IF ecosystem) with a specific need, or an internal working group of the IF technical governance body submits a proposal to modify or extend any of the ontologies in the Shift2Rail ontology network, or create a new one. The proposal must be accompanied by a detailed motivation statement of the reasons why such modification/extension/creation under assessment has a significant benefit to all actors involved (and not just to the proposer).

   This proposal will be done by creating an issue in the corresponding GitHub repository of the ontology to be modified/extended, or in a general repository for the ontology network, in case of proposing a new ontology.

2. After the issue has been submitted, a temporal window is opened, where every stakeholder can express its opinion and/or critique regarding the usefulness/adequacy of the proposed solution. The duration of the window depends on the type of the asset and is decided by the Technical Board. All the discussions are handled publicly in the corresponding GitHub repository, associated to the original issue.

3. Meanwhile, the Quality Assurance Board evaluates the technical quality of the motivation statement and, in case of low quality, asks the submitter to fix the issues highlighted.

4. The Technical Board assesses the backward compatibility and accepts/discards the proposed modification/extension/creation. In case of rejection, the whole process stops here, and the submitter of the decision is given the reasons for the proposal being rejected.

5. In case of acceptance by the Technical Board, a new version of the ontology (or a new ontology) is released. The Strategic Board should proceed to scrutinise the proposed solution, together with the comments and opinions collected about the modification, and it should accept or reject the proposed modification.

6. In case of acceptance by the Strategic Board, such new version is officially released, becoming the new reference. In case of rejection, the whole process stops here, and the submitter is notified of the outcome and is given the reasons for the proposal being rejected.

This process will be coordinated publiclty in a GitHub organisation dedicated to the maintenance of the Shift2Rail ontology network, and it will be supported technologically by an enhanced version of OnToology, which will be refined during the remaining execution period of the SPRINT project, as described in section 3.2.3. In any case, the development process will need to be done according to the steps proposed in the NeOn methodology, including a clear description of the requirements for the new modifications/extensions or for the new ontology (in the form of competency questions) and the provision of a good documentation. Finally, the developed ontologies will be properly published and registered in the Asset Manager.

3.2 AUTOMATION IN ONTOLOGY DEVELOPMENT

Taking into account the fact that many of the ontologies to be developed for the Shift2Rail ontology network (including already existing ones) are strongly based on existing XML Schemas that have been used for the exchange of data across multiple systems, this section discusses on some of the tools that will be made available to the Shift2Rail constituency for semi-automating some of the steps involved in the ontology development process.

More specifically, we describe two main tools/systems that will be made available for the semi-automatic generation and refinement of OWL ontologies from XML Schema documents, and for the automation of many of the support activities associated to ontology development (generation of user-centered documentation for the ontologies, evaluation of the OWL code, and publication of the ontologies according to Linked Data principles). Section 3.2.1 will provide a brief state of the art about tools that can be used for OWL ontology generation from XML Schema files. Section 3.2.2 will describe our proposal for the generation and refinement of such ontologies from XML Schema files. And Section 3.2.3 will describe how we are extending the current version of OnToology in order to provide support to the whole ontology development lifecycle.

3.2.1 State of the art in ontology generation from XML Schema

Several tools have been proposed in the state of the art for the generation of OWL ontologies from XML Schemas. This format is selected as the main input format in our analysis of automation of ontology generation given the fact that many of the standards that will need to be dealt with in the Shift2Rail interoperability framework are described using XML Schema, reflecting the way in which the XML messages between systems are interchanged, and describing hence an *a priori* informal agreement on the way in which data in this area is provided.

The following tools have been analysed:

- OntMalizer[1] is a tool that allows transformations of XML Schemas (XSD) and XML data to RDF/OWL automatically and also allows you to print the output in various formats such as RDF/XML, RDF/XML-ABBREV, N-TRIPLE and Notation3 (N3).

  Performs complete transformations and it is possible to create OWL representation of XML instances that comply with such XML Schemas using XSOM library java for processing XML Schemas.

  In the case of XML Schemas it uses the following transformation rules[2]:

Table 1. Transformation rules that are used in the Ontmalizer

Types	Description
xs:simpleType	rdfs:Datatype with the suffix “Datatype” on datatype name.
xs:simpleType with xs:enumeration	rdfs:Datatype with the suffix “Datatype” on datatype name.In addition, for the enumerations, an owl:Class as a subclass of EnumeratedValue is created. Instances are created for every enumerated value. An instance of Enumeration, referring to all the instances, is created as well as the owl:oneOf union over the instances. These are mostly informative, as they are not used directly during the XML data to RDF/OWL transformation.
xs:complexType over xs:complexContent	owl:Class
xs:complexType over xs:simpleContent	owl:Class
xs:element (global) with complex type	owl:Class and subclass of the class generated from the referenced complex type
xs:element (global) with simple type	owl:Datatype
xs:element (local to a type)	owl:DatatypeProperty or owl:ObjectProperty depending on the element type. OWL Restrictions are built for the occurrence.

---

[1] https://github.com/srdc/ontmalizer

[2] http://www.srdc.com.tr/projects/salus/blog/?p=189

Types	Description
xs:group	owl:Class
xs:attributeGroup	owl:Class
Anonymous Complex Type	As for Complex Type except a URI is constructed as Anon_#. Also, the class is defined as a subclass of Anon.
Anonymous Simple Type	As for Simple Type except a URI is constructed as Anon_#.
Substitution Groups	Subclass statements are generated for the members.
xsi:type on an XML element	Overrides the schema abstract type with the specified type.

- XmlToRdf[3] is developed in java to convert XML into RDF, one of its main features is its speed in the transformations, and this is achieved by incorporating SAX java (Simple API for XML). This allows you to get a set of information from XML documents in linear sequence and continuous consuming less resources since it is not necessary to load the entire document in memory.

  Sometimes an element will contain mixed content in some XML documents, this tools allows detect and convert mixed content into an RDF list structure.

- CSV2RDF[4] is an open-source tool to generate RDF from CSV files, this is achieved from a template that is an RDF file showing how one row from CSV will be mapped to RDF.

  In our test, the tool works correctly but you would have to take into account the cost and scalability of generating the template because there is little information about it.

3.2.2 Ontology generation and refinement (from XML Schema)

As discussed in the previous section, several tools exist for the transformation of XML Schema into OWL. After several tests with the tools that have been briefly described in the previous section, considering those that we have been able to run with sample XML Schemas from the Shift2Rail domain, we have selected Ontmalizer, which is the one that provides a more complete output in OWL from the provided XML Schema. Figure 2 provides a graphical overview of how the tool actually works.

---

[3] https://github.com/AcandoNorway/XmlToRdf

[4] https://github.com/clarkparsia/csv2rdf

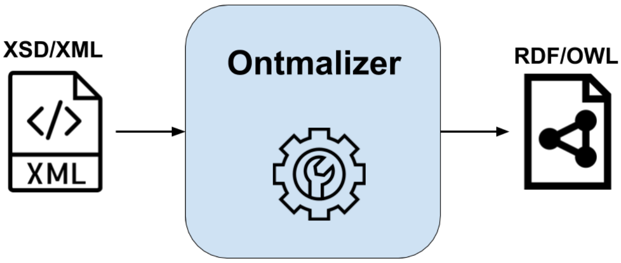


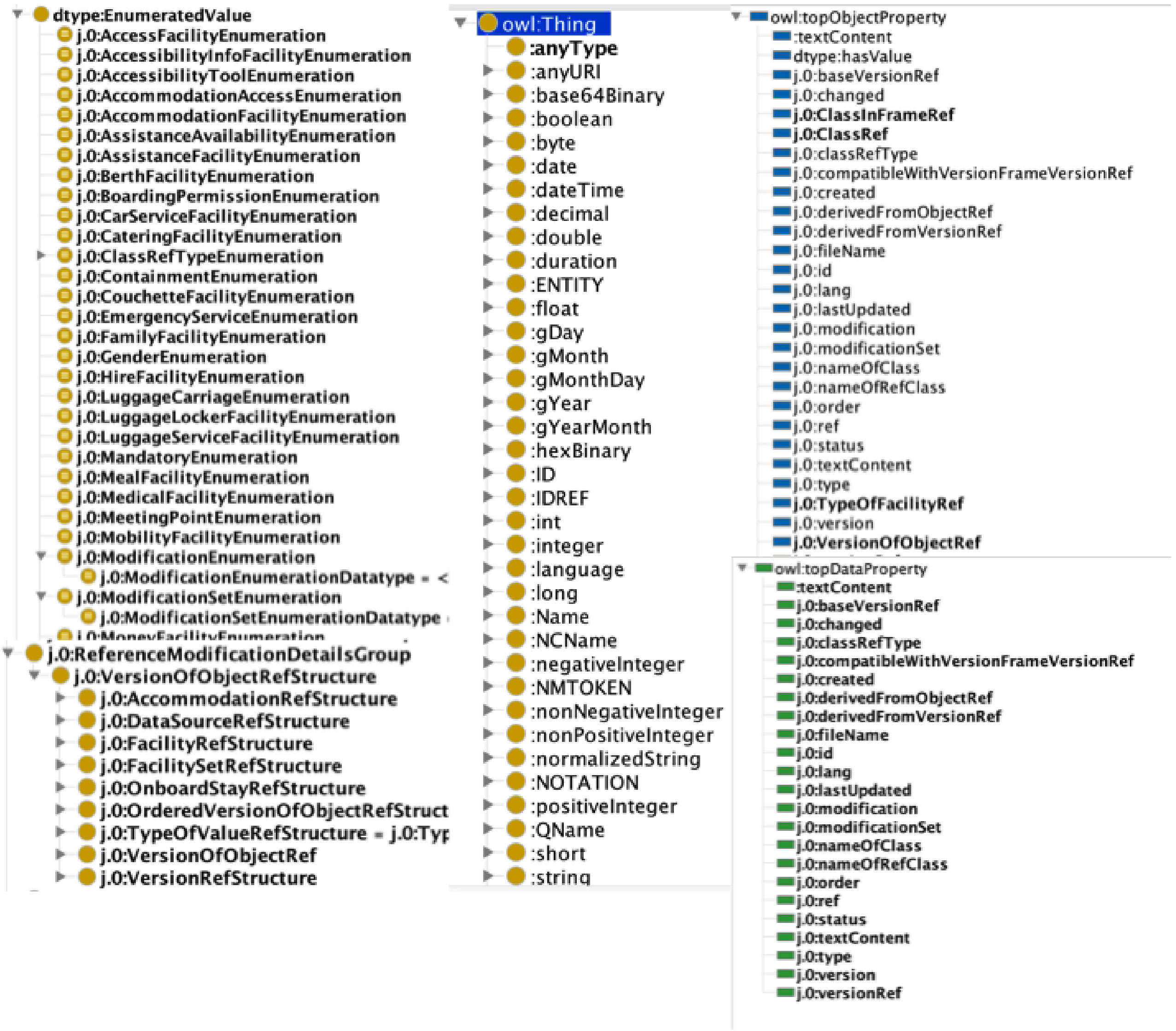
Figure 2. Input and output data for the Ontmalizer transformation process

As an example, let's take as an input an XML Schema from NeTEx, more specifically the XML Schema for the description of facilities, which is available in GitHub[5]. This file contains the definitions of several XML simple types (AccessibilityInfoFacilityEnumeration, MobilityFacilityEnumeration, PassengerInformationFacilityEnumeration, SafetyFacility Enumeration, AccessFacilityEnumeration, AccessibilityToolEnumeration, Accommodation AccessEnumeration, AccommodationFacilityEnumeration, AssistanceFacilityEnumeration, etc.) and several complex types (typeOfFacilityRefs_RelStructure, TypeOfFacilityRefStructure, OnboardStayRef, AccommodationRef, etc.).

Once the Ontmalizer system is run on this XML Schema file, the set of classes and properties presented in Figure 3 are generated.

---

[5] https://github.com/NeTEx-CEN/NeTEx/blob/master/xsd/netex_framework/netex_reusableComponents/netex_facility_support.xsd



- dtype:EnumeratedValue
  - j.0:AccessFacilityEnumeration
  - j.0:AccessibilityInfoFacilityEnumeration
  - j.0:AccessibilityToolEnumeration
  - j.0:AccommodationAccessEnumeration
  - j.0:AccommodationFacilityEnumeration
  - j.0:AssistanceAvailabilityEnumeration
  - j.0:AssistanceFacilityEnumeration
  - j.0:BerthFacilityEnumeration
  - j.0:BoardingPermissionEnumeration
  - j.0:CarServiceFacilityEnumeration
  - j.0:CateringFacilityEnumeration
  - j.0:ClassRefTypeEnumeration
  - j.0:ContainmentEnumeration
  - j.0:CouchetteFacilityEnumeration
  - j.0:EmergencyServiceEnumeration
  - j.0:FamilyFacilityEnumeration
  - j.0:GenderEnumeration
  - j.0:HireFacilityEnumeration
  - j.0:LuggageCarriageEnumeration
  - j.0:LuggageLockerFacilityEnumeration
  - j.0:LuggageServiceFacilityEnumeration
  - j.0:MandatoryEnumeration
  - j.0:MealFacilityEnumeration
  - j.0:MedicalFacilityEnumeration
  - j.0:MeetingPointEnumeration
  - j.0:MobilityFacilityEnumeration
  - j.0:ModificationEnumeration
    - j.0:ModificationEnumerationDatatype ≡ <
  - j.0:ModificationSetEnumeration
    - j.0:ModificationSetEnumerationDatatype
  - j.0:MoneyFacilityEnumeration
- j.0:ReferenceModificationDetailsGroup
  - j.0:VersionOfObjectRefStructure
    - j.0:AccommodationRefStructure
    - j.0:DataSourceRefStructure
    - j.0:FacilityRefStructure
    - j.0:FacilitySetRefStructure
    - j.0:OnboardStayRefStructure
    - j.0:OrderedVersionOfObjectRefStruct
    - j.0:TypeOfValueRefStructure ≡ j.0:Typ
    - j.0:VersionOfObjectRef
    - j.0:VersionRefStructure

- owl:Thing
  - :anyType
  - :anyURI
  - :base64Binary
  - :boolean
  - :byte
  - :date
  - :dateTime
  - :decimal
  - :double
  - :duration
  - :ENTITY
  - :float
  - :gDay
  - :gMonth
  - :gMonthDay
  - :gYear
  - :gYearMonth
  - :hexBinary
  - :ID
  - :IDREF
  - :int
  - :integer
  - :language
  - :long
  - :Name
  - :NCName
  - :negativeInteger
  - :NMTOKEN
  - :nonNegativeInteger
  - :nonPositiveInteger
  - :normalizedString
  - :NOTATION
  - :positiveInteger
  - :QName
  - :short
  - :string

- owl:topObjectProperty
  - :textContent
  - dtype:hasValue
  - j.0:baseVersionRef
  - j.0:changed
  - j.0:ClassInFrameRef
  - j.0:ClassRef
  - j.0:classRefType
  - j.0:compatibleWithVersionFrameVersionRef
  - j.0:created
  - j.0:derivedFromObjectRef
  - j.0:derivedFromVersionRef
  - j.0:fileName
  - j.0:id
  - j.0:lang
  - j.0:lastUpdated
  - j.0:modification
  - j.0:modificationSet
  - j.0:nameOfClass
  - j.0:nameOfRefClass
  - j.0:order
  - j.0:ref
  - j.0:status
  - j.0:textContent
  - j.0:type
  - j.0:TypeOfFacilityRef
  - j.0:version
  - j.0:VersionOfObjectRef

- owl:topDataProperty
  - :textContent
  - j.0:baseVersionRef
  - j.0:changed
  - j.0:classRefType
  - j.0:compatibleWithVersionFrameVersionRef
  - j.0:created
  - j.0:derivedFromObjectRef
  - j.0:derivedFromVersionRef
  - j.0:fileName
  - j.0:id
  - j.0:lang
  - j.0:lastUpdated
  - j.0:modification
  - j.0:modificationSet
  - j.0:nameOfClass
  - j.0:nameOfRefClass
  - j.0:order
  - j.0:ref
  - j.0:status
  - j.0:textContent
  - j.0:type
  - j.0:version
  - j.0:versionRef

Figure 3. Some classes and properties automatically generated from an XML Schema file from NeTEx

It is important to note that the automatically generated OWL file cannot be considered an ontology yet, taking into account that it requires, on the one hand, adequate documentation (including requirements written as competency questions and a final documentation that can be used by humans willing to make use of the ontology), and on the other hand, several additional refinement steps so as to follow usual ontology development practices in OWL that are not properly encoded in the output generated by the tool (e.g., unclear prefixes such as j.0, production of unnecessary classes that reflect datatypes, object properties that are not domain properties but generic ones, enumerations that should be written as SKOS thesauri instead of class taxonomies, and several classes named RefStructure that should be written differently and are really the classes that need to be created).

That is, the following refinement steps need to be done in order to obtain a cleaner OWL file that does not contain typical ontology modelling errors:

- Remove wrongly generated classes that refer to datatypes.
- Clean up ontology prefixes and generate and adequate ontology URI.
- Remove wrongly generated object properties.
- Transform classes and instances generated from enumerations into SKOS codelists and their corresponding SKOS concepts.
- Rename the classes that come from reference structures.

3.2.3 Integrated technology support for the ontology development process

Once that we have presented our methodological proposal for the collaborative development of ontologies in the context of Shift2Rail, as well as some of the tools that will be available to partially automate some of the tasks in this process, we will describe the proposed technology framework to support such ontology development processes.

We will start from the current technology architecture used by OnToology, which is shown in Figure 4. One of the key components in the whole architecture is the change monitor, which acts as a monitoring service for keeping track of changes in a GitHub repository that OnToology is registered to (that is, a repository where an ontology is being maintained during the ontology development process). The Orchestration service controls the main actions of the system, and decides when to invoke the different tools integrated with OnToology. Finally, the Integrator module is responsible for the execution of processes, implemented within OnToology or third party applications.

The Integrator module (hence OnToology) handles the following functionalities:

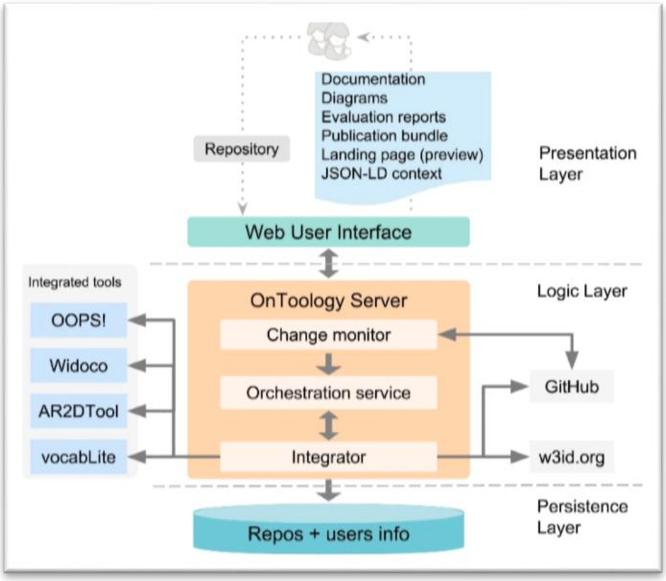


Figure 4. Current architecture of OnToology

- **Documentation**, including several resources generated by different tools. First, it generates the HTML documentation of the ontology, supported by the integration of WIDOCO [10], a standalone application for generating HTML documentation for an individual ontology. WIDOCO extracts metadata properties from ontologies, generates several serializations for content negotiation and creates a provenance summary that records the changes from earlier versions. Second, ontology diagrams are created by AR2DTool[6], a library for generating diagrams based on RDF-encoded information. AR2DTool is meant for drawing diagrams of ontologies, generating not only compiled figures (i.e. PNG files) but also their source code (i.e. GraphML6 files) for ontology developers to further edit them if they wish. Third, OnToology generates a repository pre-visualization page, which is supported by the integration of VocabLite[7], an application designed to create of overview pages for a set of ontologies (and their metadata) in a repository.
- **Evaluation**, by means of OOPS! [11], a web application for ontology diagnosis that automatically detects 33 types of pitfalls in OWL ontologies, including semantic and structural checks as well as best practices verification. For each identified pitfall in the ontology, OOPS! provides its title, description, list of ontology elements affected and its importance level (critical, important or minor). It is worth noting that OOPS! does not perform reasoning on the ontology, as it is a time-consuming task and costly in terms of resources. However, OOPS! detects pitfalls that may lead to reasoning issues. In case of need, users may execute reasoners over their ontologies within their ontology edition environment such as Protégé before registering them in OnToology.
- **Publication**, providing different features following ontology publication best practices. OnToology allows users to reserve a name for their ontology with a permanent URI following the scheme https://w3id.org/def/<user-chosen-name>. It uses the w3id services to point to the location of the published ontology and its resources with content negotiation. Alternatively, OnToology allows downloading a publication-ready bundle containing all the resources needed to publish an ontology on a custom server.

While OnToology is already available and being used in the context of the development of a wide range of ontologies in different domains, enhancements need to be done in order to adapt it to other technology frameworks beyond GitHub, and to facilitate its maintenance. To some extent, OnToology can be seen as a specialised tool that implements a continuous integration pipeline for ontology development. Therefore, such enhancements for Shift2Rail will be done in the context of general-purpose continuous integration technology, as follows:

A continuous integration pipeline for ontology development allows ontology developers to integrate ontologies into a shared repository several times a day. Each check-in is verified by an automated build process, allowing developers to detect any problems early. The pipeline process is triggered when the ontology files are committed to a repository (e.g., a git repository like those stored in GitHub, but also others). Next comes the notification to the build system, which compiles the files and runs unitary tests. Once unitary tests are evaluated successfully, the next step is the integration test. And once the integration test is done, the ontology can be published.

A continuous integration pipeline can be configured by means of a file called Jenkins file, which is part of the infrastructure provided in the Shift2Rail Asset Manager. Jenkins offers continuous integration and a continuous delivery environment for almost any combination of languages. A

---

[6] https://github.com/idafensp/ar2dtool

[7] https://github.com/dgarijo/vocabLite

Jenkins file contains the set of steps to be followed to reach the integration of ontologies generated by the ontology developers. It provides a faster and more robust way to integrate the entire chain of build, test and deployment tools.

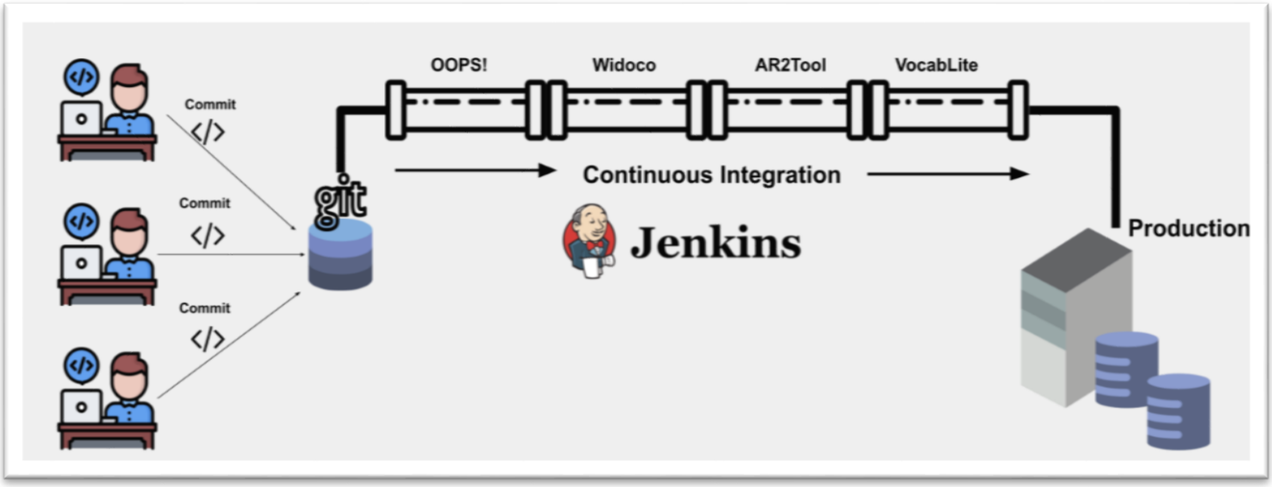
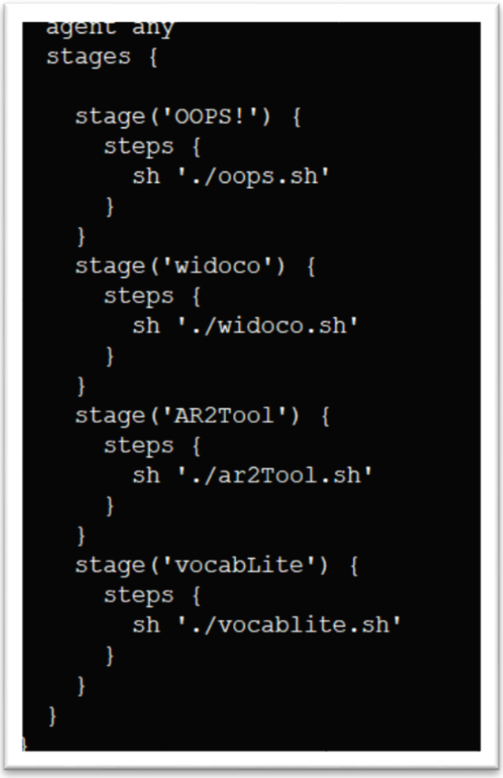


Figure 5. A continuous integration pipeline for ontology development using Jenkins

In Figure 5, we can see that ontology developers commit changes to the source code in a shared repository. Each developer can commit his/her own changes to the shared repository in a parallel fashion, although we will recommend using the collaborative ontology development process defined in section 3.1.2. The Jenkins server checks the shared repository at periodic intervals and every check-in is pulled and then built. We can create Jenkinsfile within the shared repository and in that file we can define the steps of building, testing and deploying ontologies. Once testing is successful, Jenkins can deploy the built application on the test server.

For the time being, we have created the structure with the pipelines shown below, all of these need to be defined in the Jenkinsfile configuration file.

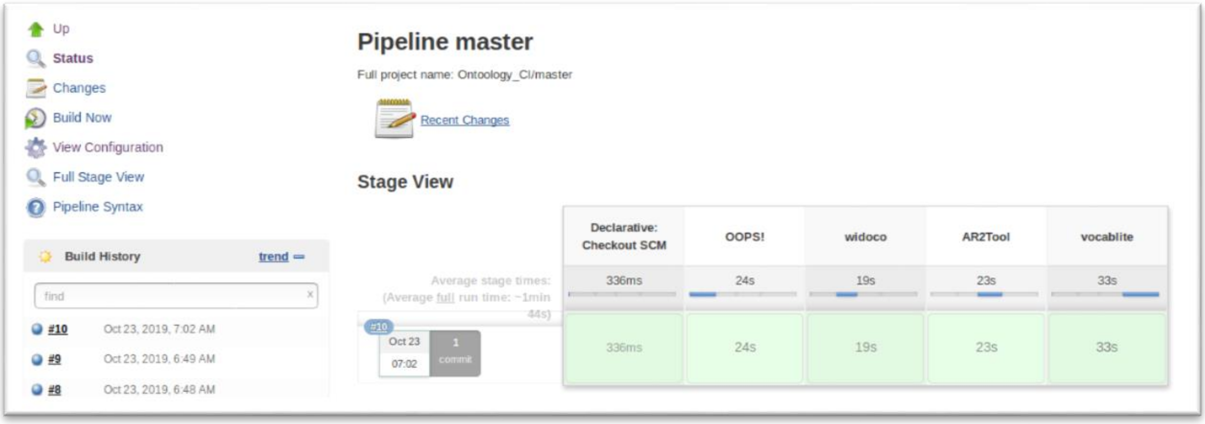


```
agent any
stages {

  stage('OOPS!') {
    steps {
      sh './oops.sh'
    }
  }
  stage('widoco') {
    steps {
      sh './widoco.sh'
    }
  }
  stage('AR2Tool') {
    steps {
      sh './ar2Tool.sh'
    }
  }
  stage('vocabLite') {
    steps {
      sh './vocablite.sh'
    }
  }
}
```

Figure 6. Default Jenkinsfile configuration file for ontology development projects

At each step, the execution parameters of each tool that is contacted in the continuous integration process have been defined. In Figure 7 we can see the pipeline execution process.



#10 Oct 23, 2019, 7:02 AM
#9 Oct 23, 2019, 6:49 AM
#8 Oct 23, 2019, 6:48 AM

Pipeline master

Full project name: Ontoology_CI/master

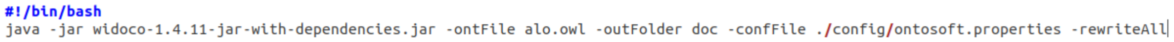
Stage View

Average stage times:
(Average full run time: ~1min 44s)

	Declarative: Checkout SCM	OOPS!	widoco	AR2Tool	vocablite
	336ms	24s	19s	23s	33s
#10 Oct 23 07:02 1 commit	336ms	24s	19s	23s	33s

Figure 7. A sample pipeline execution process

And in Figure 8 we show a defined step in our Jenkinksfile, in this case for Widoco. We can observe that our ontology in this case is called alo.owl and that in this case is the one that will be modified by the different ontology developers.



```
#!/bin/bash
java -jar widoco-1.4.11-jar-with-dependencies.jar -ontFile alo.owl -outFolder doc -confFile ./config/ontosoft.properties -rewriteAll
```

Figure 8. Shell script running the Widoco documentation system for an ontology

One of the additions that will be made in the context of the enhancement and adaptation of OnToology to such a continuous integration system will be a final step that includes the publication of the ontology as an asset in the Asset Manager, instead of only publishing in an online server.

3.3 AUTOMATION IN MAPPING AND ANNOTATION CREATION

3.3.1 Introduction

This section presents our approach to make the annotation phase of the conversion process between different standard formats more automated and efficient. A ST4RT converter is a software artefact that acts as an adapter between two distinct formats. The core idea behind the conversion process is to go beyond pure "syntactic" interoperability, where interested parties are forced to adopt a unified set of formats for data exchange, and instead leverage "semantic" interoperability, which enables different systems to communicate with each other through their native standards, by mapping their concepts to a common ontology, which provides an unambiguous and homogeneous view of data.

Given a suitable mapping between the source/target data and a reference formal ontology, a ST4RT converter first transforms data expressed in the source format into an intermediate representation based on the reference ontology. Then, following a similar procedure in the reverse direction, the converter translates the intermediate representation into the target data model. A ST4RT Converter relies on an annotation-based process, through which the source and target standards are semantically annotated to state the mappings between their data models and a reference ontology. Figure 9 depicts a general concept of conversion process introduced in the ST4RT project that provides an automatic translation between two heterogenous formats. This eliminates the need of forcing transport bodies to follow one common standard format.

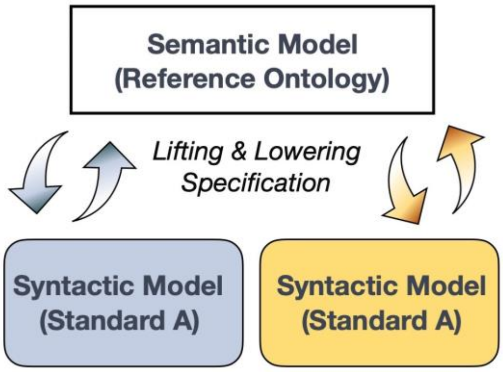


Figure 9. ST4RT Conversion Process

The annotation process in the ST4RT Converter, however, is carried out manually, which hampers the efficiency and overall performance of the process. Human users who are expert in both the reference ontology and the desired standards are required to establish the mapping between them. In the proposed approach, we aim at making the annotation-creation process more automated by taking advantage of machine-learning algorithms and methodologies

3.3.2 Background

This part describes the procedure for mapping a source standard to a target standard in the transportation domain. In this context, an automated conversion occurs when it is possible to translate a concept in a standard to its corresponding concept in the other standard without human intervention.

To implement and explore our approach, we have considered two different transportation standards/sets of concepts. The same concept/operation/instance in the two representations is captured using different words/terminologies or, possibly, different structures. Whereas humans can fairly easily understand the correspondence between different – but analogous – terms, it is much harder for machines to do so. In addition, when the number of terms in a standard is significant, finding the right correspondences is difficult and time-consuming also for humans. So, our goal is to cover this gap by providing an automated mechanism to suggest the mapping/translation of terms representing same concept.

In the rest of this section we provide a brief introduction to the two sets of transportation concepts, the IT2Rail glossary and the Transmodel data standard, that have been used for our experiments.

IT2Rail: IT2Rail (Information Technologies for Shift2Rail) is a project whose main objective consists in enabling the development of door-to-door solutions providing a seamless travel experience by giving access to a complete multimodal travel offer (combining air, rail, coach and other services), which connects the first and last mile of long-distance journeys. The IT2Rail glossary comprises a list of terms – along with their definitions – capturing the key concepts that are used throughout the IT2Rail project. Table 2 shows some of the concepts defined in the IT2Rail glossary, whose aim is to provide non-technical, understandable definitions to be shared amongst the project stakeholders, both technical and non-technical ones. The list of key terms and definitions provided in the IT2Rail glossary is aligned line with the technical definitions provided in the IT2Rail ontology produced and compiled in the context of the Interoperability Framework.

Table 2 Example of IT2Rail concepts

Term in IT2Rail	Definition
Access Area	An area such as a concourse and/or a booking hall and/or other areas (e.g. commercial or lounge) which is secured or not and which is managed by a StopPlace manager.
Airport	Airport is an infrastructure prepared to operate and accommodate the arrival and departure for air traffic.
Booking	Operational process involved in the sales process to commit to a sales transaction binding the customer and supplier on the offer

Transmodel: Transmodel is the short name for European Standard Public Transport Reference Data Model. Transmodel provides an abstract model of common public transport concepts and data structures that can be used to build many different kinds of public transport information systems, including timetables, fares, operational management, real time data journey planning etc. Table 3 shows a few terms defined in Transmodel.

Transmodel establishes a consistent terminology for describing public transport concepts, providing definitive equivalents for use in the national languages of each participant nation. Where public transport-related words in vernacular use may span a number of different concepts and lead to differences of interpretation, Transmodel establishes a more precise technical terminology for unambiguous use by public transport information system developers. For example, the terms 'trip', 'journey', 'service', are overlapping concepts that in Transmodel are used only in a more specific manner.

Table 3. Example of Transmodel concepts

Term in Trasnmodel	Definition
ACCESS	The physical (spatial) possibility for a passenger to access or leave the public transport system.
ACCESSIBILITY ASSESSMENT	The accessibility characteristics of an entity used by passengers such as a STOP PLACE
CONTROL CENTRE	An ORGANISATION PART for an operational team who are responsible for issuing commands to control the services
DIRECTION	A classification for the general orientation of ROUTEs

As mentioned above we aim to use a machine-learning approach, and in particular the Word2Vec technique [12], to catch the similarity among the concepts in different standards. Below we provide a brief overview of this approach.

Word2Vect

As the name suggests, Word2vec is the technique/model to produce word embeddings. Word2vec represents words in a vector space: words are represented as vectors whose position in the vector space is such that words wish similar meaning are close, whereas dissimilar words are located far away. This similarity is also considered as a semantic relationship. Figure 10 shows the function “Cosine Distance” which is used to calculate the similarity between different terms. Mathematically, for two words to be similar the cosine of the angle between the corresponding vectors should be close to 1, i.e. the angle should be close to 0. Figure 10 shows the formula used to calculate the cosine similarity between two terms. The cosine similarity is the normalized dot product of 2 vectors and this ratio defines the angle between them.

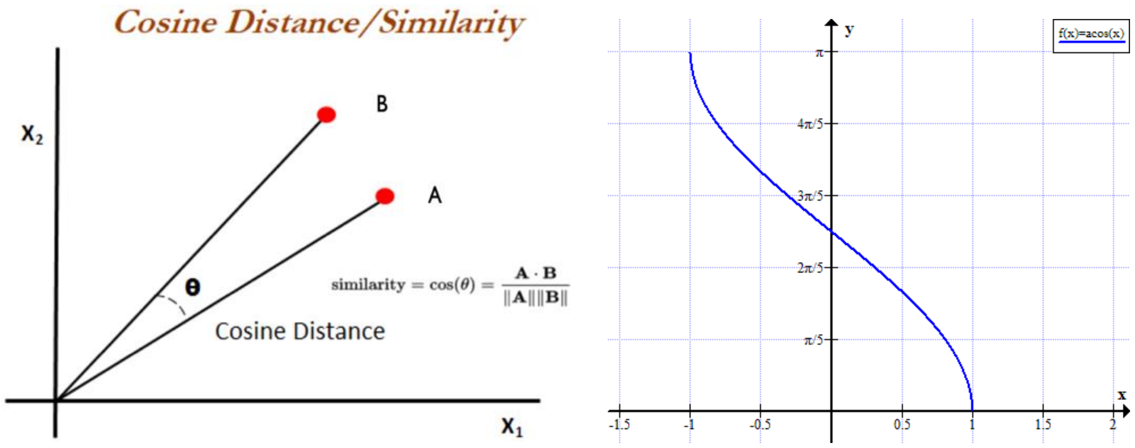


Figure 10. (a) Word2Vec Cosine distance, and (b) Cosine angle

As shown in Figure 10, the cosine distance varies from 1 to -1, and the corresponding angle varies from 0 to 180 degrees. Two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude.

The basic idea of the word2vec model is to represent objects in a space where proximity means that two items are similar. Figure 11 shows a qualitative depiction of such a space.

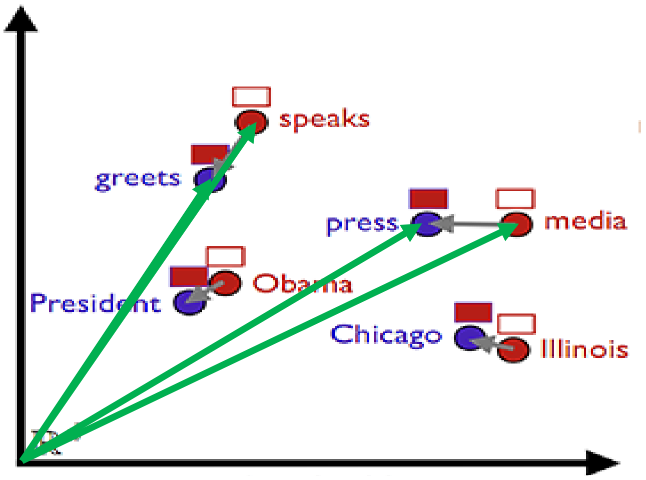


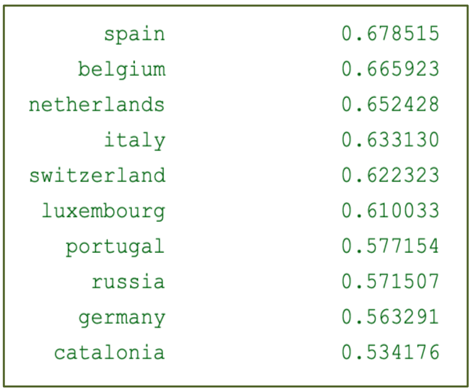
Figure 11. Words in word2vec vector space

In this case the similarity between words is determined by the cosine of the angle between them. Hence, the greater the cosine measure of the angle, the more similar the words will be, as depicted in Figure 11; for example, the value of the cosine of the angle between the vectors from the origin to “greets” and “speaks” is close to 1, so they are located closer to each other in the vector space. Similarly, the cosine of the angle between the vectors connecting “press” and “media" to the origin is closer to 1 than to 0. Under cosine similarity, complete similarity corresponds to a 0-degree angle, which means that cosine similarity between words is one.

The word2vec tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns the vector representation of words. The resulting word vector file can be used to support many natural language processing and machine learning applications.

There are multiple pretrained models like Google's original model [13], Glove and Facebook's fastText for many languages. To implement our proposed technique, we use Google's original pretrained model, which has a size of 1.5 Giga Bytes. It includes word vectors for a vocabulary of 3 million words and phrases that they trained on roughly 100 billion words from a Google News dataset. The vector length is 300 features.

A simple way to investigate the learned representations is to find the closest words for a user-specified word. The distance tool serves that purpose. For example, if one enters 'France', the distance tool will display the most similar words and their distances to 'France', which should look like as shown in Figure 12.



```
      spain             0.678515
    belgium             0.665923
netherlands             0.652428
      italy             0.633130
switzerland             0.622323
 luxembourg             0.610033
   portugal             0.577154
     russia             0.571507
    germany             0.563291
  catalonia             0.534176
```

Figure 12. Word Cosine Distance for word "France"

The main features of the pre-trained Google news model are the following:

- **Stop words:** Some stop words like "a", "and", "of" are excluded, but others like "the", "also", "should" are included.
- **Misspelled words**: There are some misspelled words. For instance, the model includes both "mispelled" and "misspelled"–the latter is the correct one.
- **Paired words:** The model included words that are the combination of other words. For instance, it includes "Soviet_Union" and "New_York".
- **Numbers:** Numbers are not directly supports; for example "100" is not included; however, the model includes entries such as "###MHz_DDR2_SDRAM" where the '#' are intended to match any digit.

3.3.3 Suggesting mappings through learning mechanisms

This section describes the algorithm realizing the proposed technique. It comprises 5 steps which are described in the workflow section.

Algorithm

```
[Initialize]

Given source_d: Source Data,  target_d: Target Data,  M: Model
 1.[load M] → vocab_list, model
 2.[Read source_d, target_d]
       a. Read texts: key_term_extraction(source_d) → source_d
                      key_term_extraction(target_d)  → target_d

       b. foreach word $W_s$ in source_d: compareTo($W_s$, vocab_list) → f_s
          foreach word $W_t$ in target_d:   compareTo($W_t$, vocab_list)  →f_t


 3.[find n similar words]
      a.  foreach word $s_i$ in f_s:
          find n similar words for $s_i$ from model → [$s_i$, $sm_{i1}$, $sm_{i2}$, … $sm_{in}$]
                $Ws_i$ = [$s_{i0}$, $sm_{i1}$, $sm_{i2}$, … $sm_{in}$],  $i \in [0,j]$, where j is no: of rows/words in f_s

      b.   foreach word $t_i$ in f_t:
           find n similar words from model → [$t_0$, $sm_1$, $tm_2$, … $tm_n$]
               $Wt_i$ = [$t_{i0}$, $tm_{i1}$, $tm_{i2}$, … $tm_{in}$],  $i \in [0,k]$, where k is no: of rows/words in f_t


 4.[Match all words in $Ws_i$ to $Wt_i$]
        a.  foreach word $s_i$ in $Ws_i$:
              foreach word $t_i$ in $Wt_i$:
                 findVectorMatch($s_i$, $t_i$) → $vector\_value_i$
                 findOrigin($s_i$,$t_i$)→ ($Os_i$,$Ot_i$)

        $Wmatch_{in}$ = [$Os_i$, $s_i$, $Ot_i$ ,$t_i$  $vector\_value_i$]     where, $Os_i$, $Ot_i$  are origins of $s_i$ ,$t_i$, respectively
        and $Wmatch_{in}$  is a 2D matrix containing matched pair, origin and vector_value of matched
pair.
        b.  foreach matchpair ($s_i$, $t_i$) in $Wmatch_{in}$ :
            filterVectorThresh ($vector\_value_i$>0.5)→  $Wmatch_{in}$_filtered


  5. [count match instances]
     foreach origins of matchPairCount($Os_i$,$Ot_i$) in $Wmatch_{in}$_filtered:  → [$Os_i$,$Ot_i$ ,matchCount ]
         if ($Os_m$ == $Os_p$ and $Ot_m$  == $Ot_p$ )    where {m, p in [0,n] and m≠p}
         matchCount = matchCount+1
[End]
```

Figure 13. Word Matching Algorithm

Workflow

The flow diagram of Figure 14 provides a high-level description of the algorithm. The method starts with the pre-processing of the text, during which the text is cleaned, special characters and stop words are removed, and the file is loaded in text format. A detailed description of each step is provided in the rest of this section.

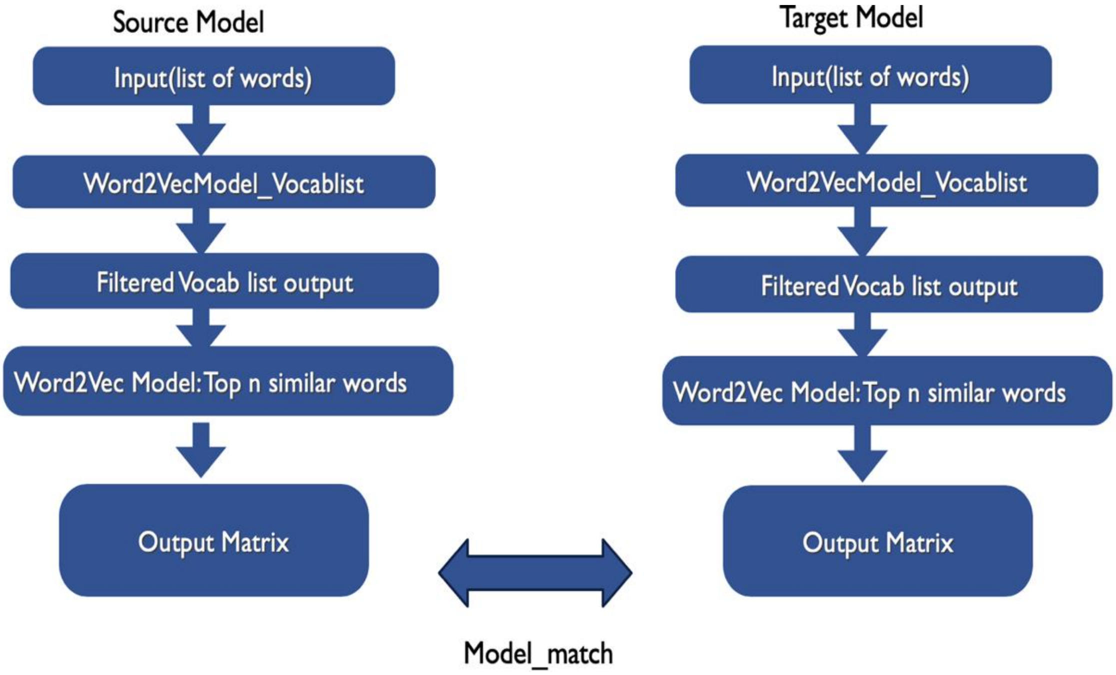
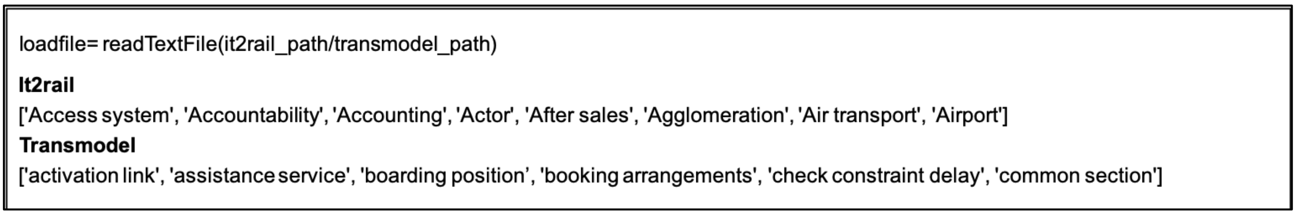


Figure 14. Word_Matching workflow

Step.1. [load M] → vocab_list, model: The Google news model is loaded using the *gensim* package, which converts the model to keyed vectors and generates the vocab list. So, the outputs of this method are `model` and `vocab_list`:

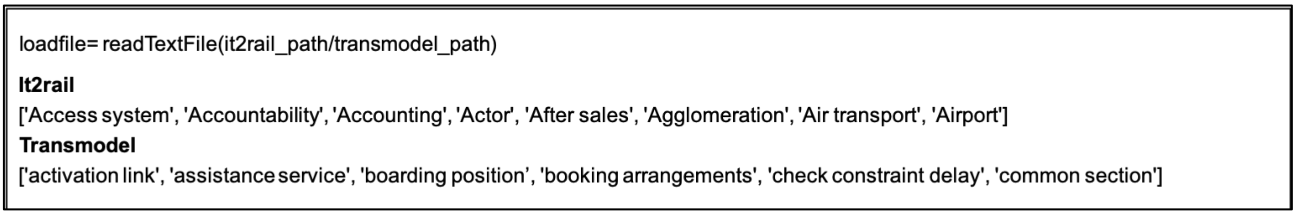
- `model`: It is the keyed vector representation of the pretrained model, through which one can look for similarity between words, nearest words etc.
- `vocab_list:` It is the vocabulary list of the model, i.e., all the unique words in the model. It is used in Step 2, to filter our input sources (IT2Rail, Transmodel).

Step.2. [Read Glossaries]: In this stage, first the complete glossaries of the transport data formats under consideration (IT2Rail, Transmodel) are loaded, then all the key terms (i.e., the main concepts defined by such standards) are extracted (see) and stored separately to be processed in the subsequent steps.



loadfile= readTextFile(it2rail_path/transmodel_path)

It2rail
['Access system', 'Accountability', 'Accounting', 'Actor', 'After sales', 'Agglomeration', 'Air transport', 'Airport']
Transmodel
['activation link', 'assistance service', 'boarding position', 'booking arrangements', 'check constraint delay', 'common section']



```
loadfile= readTextFile(it2rail_path/transmodel_path)
It2rail
['Access system', 'Accountability', 'Accounting', 'Actor', 'After sales', 'Agglomeration', 'Air transport', 'Airport']
Transmodel
['activation link', 'assistance service', 'boarding position', 'booking arrangements', 'check constraint delay', 'common section']
```

Figure 15. Loading data from Source and Target files

The extracted key concepts of each glossary are then compared to `Vocab_list` and the key concepts that do not appear in `Vocab_list` are excluded. In other words, we filter them out to keep only the words which are part of `vocab_list.`

As represented in the Algorithm (See Figure 13), the outputs of this step are `f_s` and `f_t`, where:

- `f_s:` contains all the key concepts in IT2Rail that are part of `Vocab_list`.
- `f_t:` contains all the key concepts in Transmodel that are part of `Vocab_list`.

Step 3: [find n similar words]: Then the model is applied to each word of the `f_s` and `f_t` sets to get *n* number of similar words suggested by the model for each term appearing in `f_s` and `f_t`.

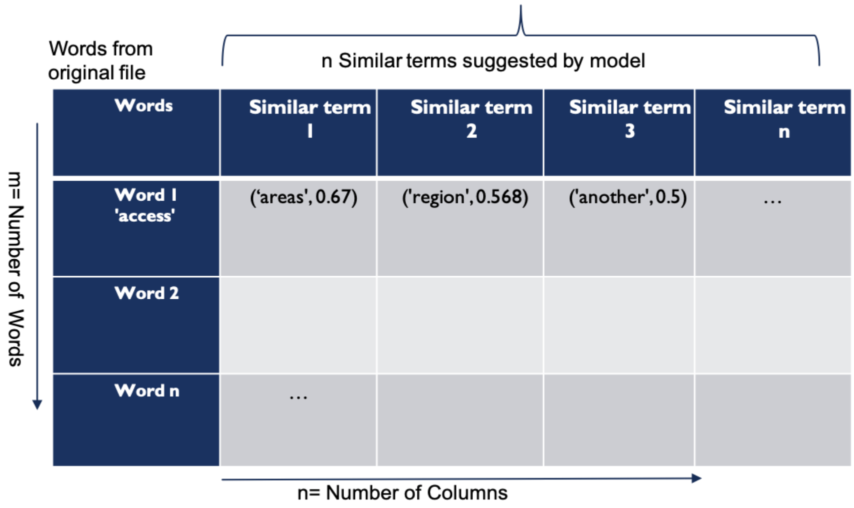
So, the output is a *j*n* matrix of elements where:

- *j*: number of words in input
- *n*: number of similar words suggested by the model for each term.

For example, if the input file has 100 words and the user wants to get 5 similar words from the model, then the size of output matrix will be the following:

- 100*6= 600, where 100 is number of rows, and 6 is number of columns.
- 6 (number of columns) = 5 (number of similar words) + 1 (the original word itself)

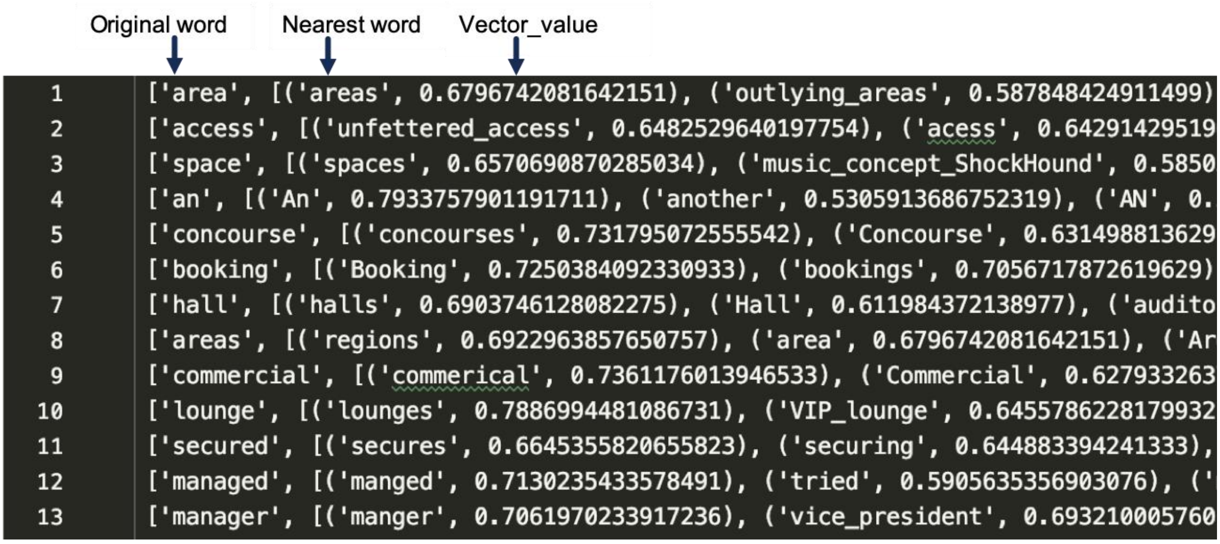
Figure 16 shows the format of the matrix getting *n* similar words from the model. The first column contains all words from the original file and the other columns are the words suggested by the model. So the actual output of this step are two matrices, namely Ws_i and Wt_i for `f_s` and `f_t`, respectively.



Words	Similar term 1	Similar term 2	Similar term 3	Similar term n
Word 1 'access'	('areas', 0.67)	('region', 0.568)	('another', 0.5)	…
Word 2				
Word n	…			

Figure 16. Matrix representation Ws_i, Wt_i

Figure 17 shows a screenshot of Ws_i for the IT2Rail model (i.e., f_s). As explained in Figure 16, in each row the left-most word is a word in the original file, and the others are those suggested by model. Each suggested word is associated with a numeric value, which is the vector value of each word that shows how much the two terms (original and suggested word) are similar. Similarly, same approach is applied for the Transmodel standard (f_t), which produces Wt_i.



```
['area', [('areas', 0.6796742081642151), ('outlying_areas', 0.587848424911499)
['access', [('unfettered_access', 0.6482529640197754), ('acess', 0.64291429519
['space', [('spaces', 0.6570690870285034), ('music_concept_ShockHound', 0.5850
['an', [('An', 0.7933757901191711), ('another', 0.5305913686752319), ('AN', 0.
['concourse', [('concourses', 0.731795072555542), ('Concourse', 0.631498813629
['booking', [('Booking', 0.7250384092330933), ('bookings', 0.7056717872619629)
['hall', [('halls', 0.6903746128082275), ('Hall', 0.611984372138977), ('audito
['areas', [('regions', 0.6922963857650757), ('area', 0.6796742081642151), ('Ar
['commercial', [('commerical', 0.7361176013946533), ('Commercial', 0.627933263
['lounge', [('lounges', 0.7886994481086731), ('VIP_lounge', 0.6455786228179932
['secured', [('secures', 0.6645355820655823), ('securing', 0.644883394241333),
['managed', [('manged', 0.7130235433578491), ('tried', 0.5905635356903076), ('
['manager', [('manger', 0.7061970233917236), ('vice_president', 0.693210005760
```

Figure 17.Top n similar words suggested by model for IT2Rail words

Step.4: [Match all words in Ws$_i$ to Wti]: At this stage we have a 2D matrix for each format. Then, we take each word from the source format (say “IT2Rail”) and match it with each term of the target format (i.e., “Transmodel”). Note that this matching is performed between 2 matrices. The matching function used in this step computes the cosine similarity is among two terms.

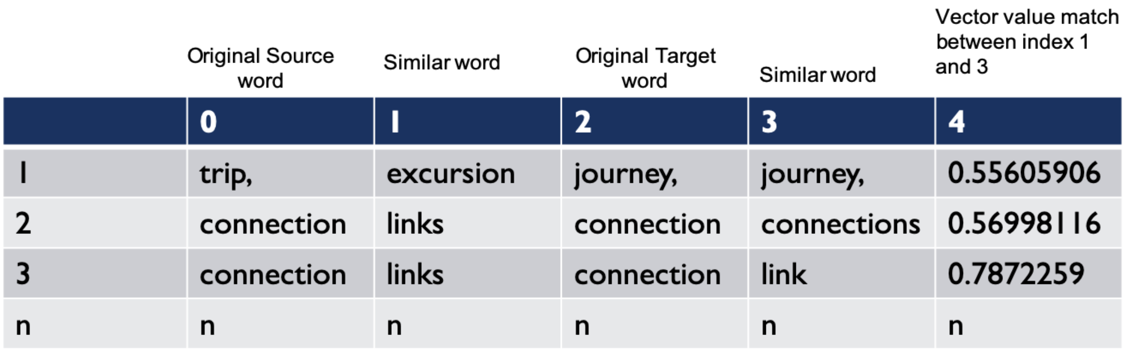
Since two 2D matrices are matched against each other, the dimensions output matrix is as following. IF the input matrices $(Ws_i,\ Wt_i)$ have the following dimensions:

- Ws_i : *j*: number of rows in the standard
  n: no. of similar words retrieved
- Wt_i : *k*: no of rows in the target standard,
  n: no. of similar words retrieved

Then, the output matrix dimension is:

$$Wmatch_{in}\ : j * k\ * (n+1)^2$$

Figure depicts a graphical explanation of $Wmatch_{in}$ and its elements, whereas Figure shows an actual example of the (raw) $Wmatch_{in}$ output, where the second column (i.e., the Source Word) is matched to column four (i.e., Target Word) and the fifth column shows the degree of similarity between these two terms. Finally,



	Original Source word	Similar word	Original Target word	Similar word	Vector value match between index 1 and 3
	0	1	2	3	4
1	trip,	excursion	journey,	journey,	0.55605906
2	connection	links	connection	connections	0.56998116
3	connection	links	connection	link	0.7872259
n	n	n	n	n	n

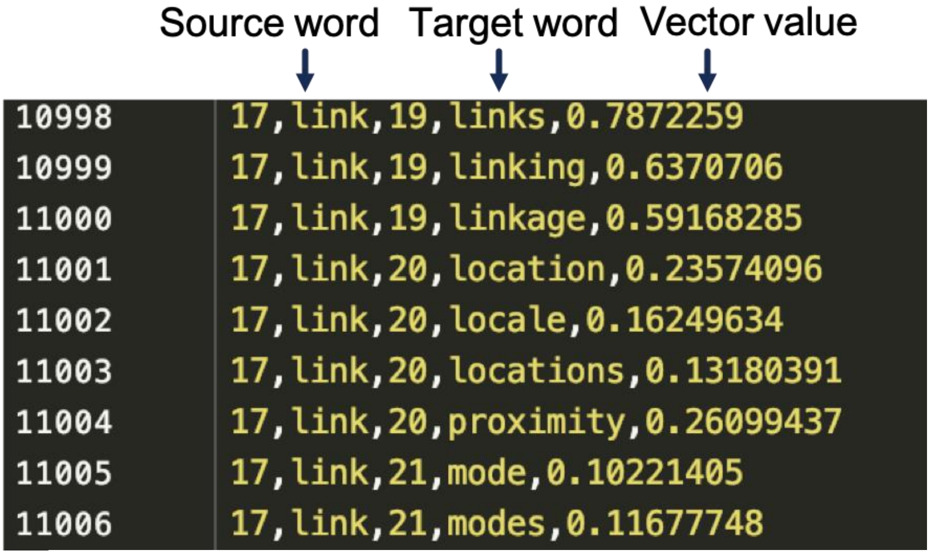
Figure 18. Format of Wmatchin, where index 1 shows match from model for word at index 0, word at index 3 shows match from model for word at index 2. finally, index 4, shows the vector match between words at indexes 1 and 3

Step.5: [count match instances]: This stage calculates the number of occurrences of each pair of suggestions. In particular, when following the approach presented above we might trace back to a particular matching of “source word → target word” from different paths (i.e., passing through a different similar word), hence multiple times. For example, as shown in **Error! Not a valid bookmark self-reference.**, which is a partial screenshot of the whole results, the match “Trip → journey” has been repeated three times with various vector numbers (lines 153 to 155). However, such combination has been achieved through different similar words. For instance (see line 154 in Error! Not a valid bookmark self-reference.) “excursion” has been calculated as a cosine similar word to the word “trip” (in the source format), and “trek” as the similar word to “journey” (from target format). Hence, we have considered the Trip → Journey pair as a match. Similarly, we obtained the same conclusion from other paths as represented in lines 153 and 155 in **Error! Not a valid bookmark self-reference.**.

is the final representation of $Wmatch_{in}$*_filtered*

Looking at Figure one can observe that the smaller vector value, such as line no. 11001 representing a pair “link, location” with a vector value 0.235, indicates that these terms are not really similar. Hence, we have assumed that a reasonable threshold for considering two terms as similar is a vector value more than 0.5. To this end, we filter the output of step 4 and we exclude the pairs whose vector value is below the threshold.

Step.5: [count match instances]: This stage calculates the number of occurrences of each pair of suggestions. In particular, when following the approach presented above we might trace back to a particular matching of “source word → target word” from different paths (i.e., passing through a different similar word), hence multiple times. For example, as shown in **Error! Not a valid bookmark self-reference.**, which is a partial screenshot of the whole results, the match “Trip → journey” has been repeated three times with various vector numbers (lines 153 to 155). However, such combination has been achieved through different similar words. For instance (see line 154 in **Error! Not a valid bookmark self-reference.**) “excursion” has been calculated as a cosine similar word to the word “trip” (in the source format), and “trek” as the similar word to “journey” (from target format). Hence, we have considered the Trip → Journey pair as a match. Similarly, we obtained the same conclusion from other paths as represented in lines 153 and 155 in Error! Not **a valid bookmark self-reference.**.

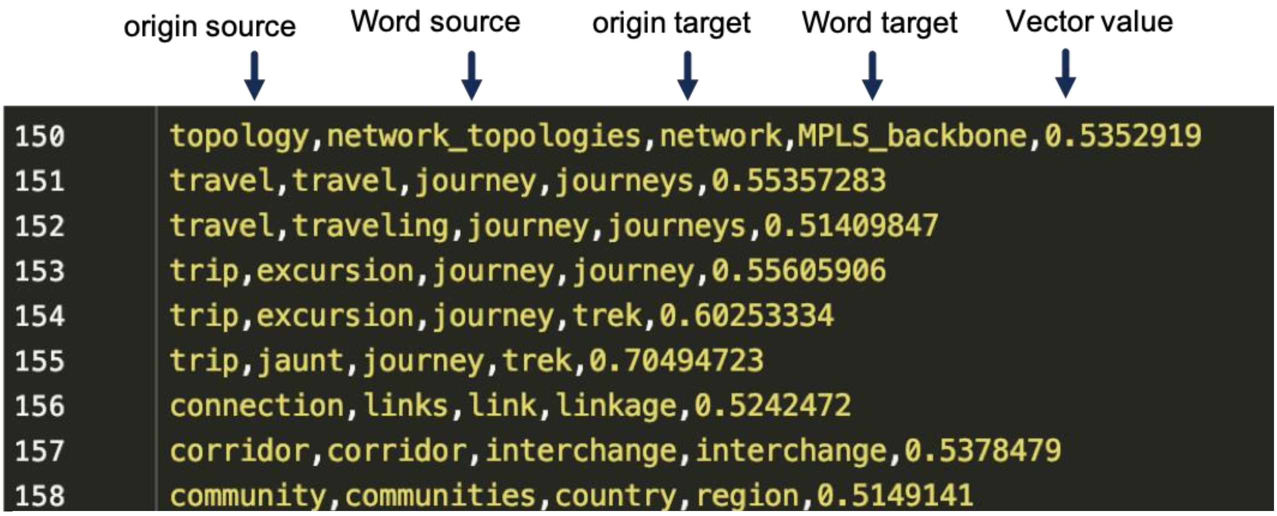


```
17,link,19,links,0.7872259
17,link,19,linking,0.6370706
17,link,19,linkage,0.59168285
17,link,20,location,0.23574096
17,link,20,locale,0.16249634
17,link,20,locations,0.13180391
17,link,20,proximity,0.26099437
17,link,21,mode,0.10221405
17,link,21,modes,0.11677748
```

Figure 19. Match between Source and Target Format

shows final representation of $Wmatch_{in}$_*filtered*, which is the extension of its representation showed in Figure . This figure actually contains the origin words of both source and target format which are being matched with one another. This improved representation of

$Wmatch_{in}$*_filtered* is created: (i) to contain only terms whose vector value is greater than the threshold as explained above; and (ii) to be able to trace back to the actual words in both source and target formats for which the match is finally being suggested.



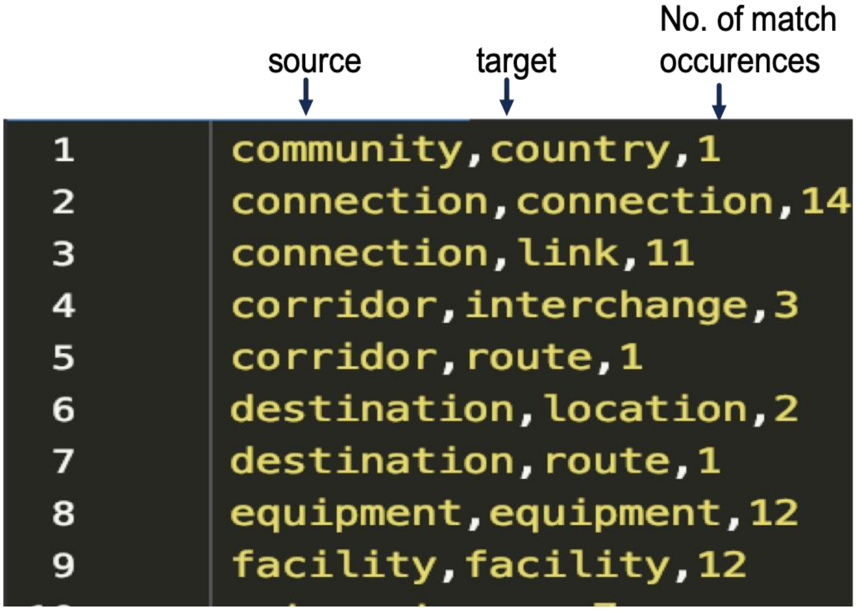
```
150   topology,network_topologies,network,MPLS_backbone,0.5352919
151   travel,travel,journey,journeys,0.55357283
152   travel,traveling,journey,journeys,0.51409847
153   trip,excursion,journey,journey,0.55605906
154   trip,excursion,journey,trek,0.60253334
155   trip,jaunt,journey,trek,0.70494723
156   connection,links,link,linkage,0.5242472
157   corridor,corridor,interchange,interchange,0.5378479
158   community,communities,country,region,0.5149141
```

Figure 20. Match between Source and Target Format includes origin

Step.5: [count match instances]: This stage calculates the number of occurrences of each pair of suggestions. In particular, when following the approach presented above we might trace back to a particular matching of "source word → target word" from different paths (i.e., passing through a different similar word), hence multiple times. For example, as shown in **Error! Not a valid bookmark self-reference.**, which is a partial screenshot of the whole results, the match "Trip → journey" has been repeated three times with various vector numbers (lines 153 to 155)[8]. However, such combination has been achieved through different similar words. For instance (see line 154 in **Error! Not a valid bookmark self-reference.**) "excursion" has been calculated as a cosine similar word to the word "trip" (in the source format), and "trek" as the similar word to "journey" (from target format). Hence, we have considered the Trip → Journey pair as a match. Similarly, we obtained the same conclusion from other paths as represented in lines 153 and 155 in **Error! Not a valid bookmark self-reference.**.

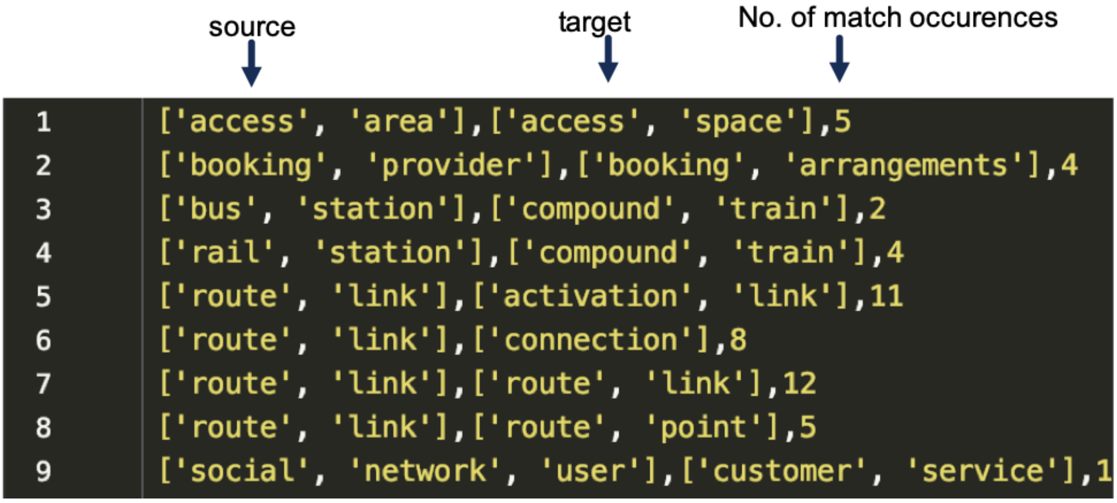
---

[8] The number of repetitions for pair Trip → Journey in our actual experiment is twelve.



```
community,country,1
connection,connection,14
connection,link,11
corridor,interchange,3
corridor,route,1
destination,location,2
destination,route,1
equipment,equipment,12
facility,facility,12
```

Figure 21. Count number of match instances



```
['access', 'area'],['access', 'space'],5
['booking', 'provider'],['booking', 'arrangements'],4
['bus', 'station'],['compound', 'train'],2
['rail', 'station'],['compound', 'train'],4
['route', 'link'],['activation', 'link'],11
['route', 'link'],['connection'],8
['route', 'link'],['route', 'link'],12
['route', 'link'],['route', 'point'],5
['social', 'network', 'user'],['customer', 'service'],1
```

Figure 22. Count number for match instances (compound Words)

The listed results discussed above are the outcome of our approach for mapping terms made of single words. However, we have applied the same procedure for compound words (e.g., “Stop Place”) separately and an example of the results is shown in Figure 22.

Discussion

The proposed technique applied to the chosen formats/sets of concepts actually suggested a mapping from source to target that is from one format to another format. Referring to **Error! Reference source not found.** and Figure 22, the number of counts shown represents the number of instances found through the model as explained earlier. Therefore, the system may suggest more than one possible translation for a term in the source to its equivalent word in the target format. Then user can select the best matching among suggested options. For instance, in our experiment we found two suggestions for the word "Locations" as represented in Table 4.

Table 4 Example of a suggested pair by system

	Suggested pair		No. of Repetition
	Source	**Target**	
Match 1	Location	Site	2
Match 2	Location	Places	1

This existence of multiple suggestions for a word could be interpreted and used in different ways as summarized in the following:

1. This procedure could be further cleaned and filtered for specific pair/words when different mappings are possible; for example, the mapping with the higher count should be given preference.
2. Both suggestions can be provided to the user, including the counts, and the user can select which is the better match.

To conclude this section, we proposed a technique to provide a mapping of instances between heterogenous formats in the transport domain. After performing the experiments mentioned above we concluded that the system works with an accuracy of around 25%. However, one can consider that Google's pretrained model, which we have been using throughout our experiments, is trained using Google's news data, which is not transport-specific, hence the model misses many terms used in the transport domain. Because of this limitation of Google's news model, even though our proposed technique is sound enough, we could not obtain the best results given the variety of terms in the vocabulary of the model.

Therefore, for single words we can claim a mapping accuracy of around 30%, while that for compound words is around 20%. To sum up, mapping suggestions could be improved by training a model with data specific to the transport domain, where we should also consider compound/multi term words ("booking, offer") as part of the vocabulary.

4. AUTOMATION HELPING IN THE ASSET MANAGEMENT LIFECYCLE

The Asset Manager is the service devoted to supporting the IF ecosystem day-by-day operations related to data sharing. Its main purpose is to assist and coordinate different actors related to different Transport Operators in the process of publishing assets, ensuring that each asset has been published by the right actor in accordance with the rules shared by all the ecosystem participants.

The Lifecycle manager is devoted to the governance of the publication of assets. Governing the publication process means coordinating both humans and services. As an example, humans need to be aware that a new asset has been published, and that it needs a formal approval before further actions can be carried out. According to human decisions, services implementing specific features can be called, such as converting an asset of type Dataset in its RDF representation according to a set of ontologies and vocabularies.

In the following we will describe the technical limitations of the IT2Rail Asset Manager and propose a different and much more flexible solution to provide support for assets governance. We will then analyze a specific topic related to governance: how to use the information contained in the Asset Manager to automatically create software artifacts. We will describe two different cases related to the IF, namely the automatic synthesis of Converter artifacts and the automatic usage of available Converters to provide different representations of datasets managed by the Asset Manager.

4.1 AN IMPROVED ASSET LIFECYCLE MANAGEMENT FOR SHIFT2RAIL

The Asset Manager implemented in IT2Rail modeled the governance process as a finished state machine (FSM). Each asset was linked to an instance of a specific FSM, which contained the set of possible asset states and the possible transitions between the different states. The format used to represent such information was State Chart XML (SCXML), a W3C specification for the representation of an FSM in XML [14]. Since a governance process is not just about representation, but also a set of actions to be exerted on both humans and machines, the WSO2 Governance manager was the open source product used as a basis to implement asset management, because it extended the SCXML specifications by allowing developers to attach Java classes implementing a specific interface to transitions. The underlying engine was then able to call a specific method inside those classes, allowing a developer to specify complex actions in response to a state change. However, there were two main limitations to such approach: implementing a governance process required mixing XML representation and Java code, and furthermore there was no support for human tasks. Overcoming the former limitation impacted on reusability, since examining the FSM would not have been enough to comprehend the whole process, while the latter limitation would have required implementing a whole layer of interaction with users.

According to the architectural principles expressed in D3.3, we choose to design a component devoted to the management of lifecycle of all the assets in an IF ecosystem. This means that the main requirement of such component is that it must be configurable and must be able to deal with different lifecycle processes, each one governing a specific asset type. Being configurable means that while looking for possible technical solutions to be used or extended, we explicitly excluded programming libraries implementing workflows. Such libraries would have required choosing a specific process to be implemented. Since adopting the Interoperability Framework should not force a Transport Operator or a Transport Authority to embrace a specific governance process, adapting the process to the adopters' needs would have then required re-writing large parts of the code, and would have slowed the time-to-market of an IF-based solution.

Workflow Management Systems are specifically tailored to support the coordination of tasks that make up the work an organization does. A 'workflow' is a sequence of tasks that are part of some larger process, sometimes referred to a 'business process'. The purpose of a workflow is to achieve some result, and the purpose of workflow management is to achieve better results according to some set of goals. One of the specifications dealing with the representation and serialization of business processes is Business Process Model and Notation[9] (BPMN). The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes.

As written before, BPMN specifies both the model and the notation. To be able to execute BPMN processes, an extension mechanism has been defined related to Human and Service Tasks, which are the main elements involved in process execution. The extension mechanism allows implementers to add custom XML elements to such task's definitions. Existing implementations of BPMN therefore provide an execution layer, which exploits such extensions allowing users to define and execute processes coordinating humans and services. Integrating a BPMN-based solution inside the Asset Manager then implies defining a set of governance processes, which must be enriched with the custom XML elements specific to the chosen implementation.

In the following, we will briefly describe some of the existing BPMN implementations, which all can be integrated with the Asset Manager using Web APIs:

- Camunda: https://camunda.com/
- Flowable: https://www.flowable.org/
- jBPM: https://www.jbpm.org/
- Imixs Workflow: https://www.imixs.org/

As a noteworthy alternative to the aforementioned tools, a Workflow Management System not based on BPMN 2.0 specifications is the following:

- Orchestra: https://orchestra.readthedocs.io

4.1.1 Camunda

Camunda is a framework written in Java that implements the BPMN standard for Workflow and Process automation. The community edition is provided with open source license(s).

Its main component is the Process Engine, which is made of POJO classes whose state is persisted in a relational database. ORM is made by means of MyBatis[10].

---

[9] http://www.bpmn.org/

[10] https://mybatis.org/mybatis-3/

The Modeler is a desktop application that allows the authoring of BPMN 2.0 processes.

A REST API layer allows a custom application to interact with the process engine, as well as Camunda Web applications like: Tasklist, Cockpit and Admin.

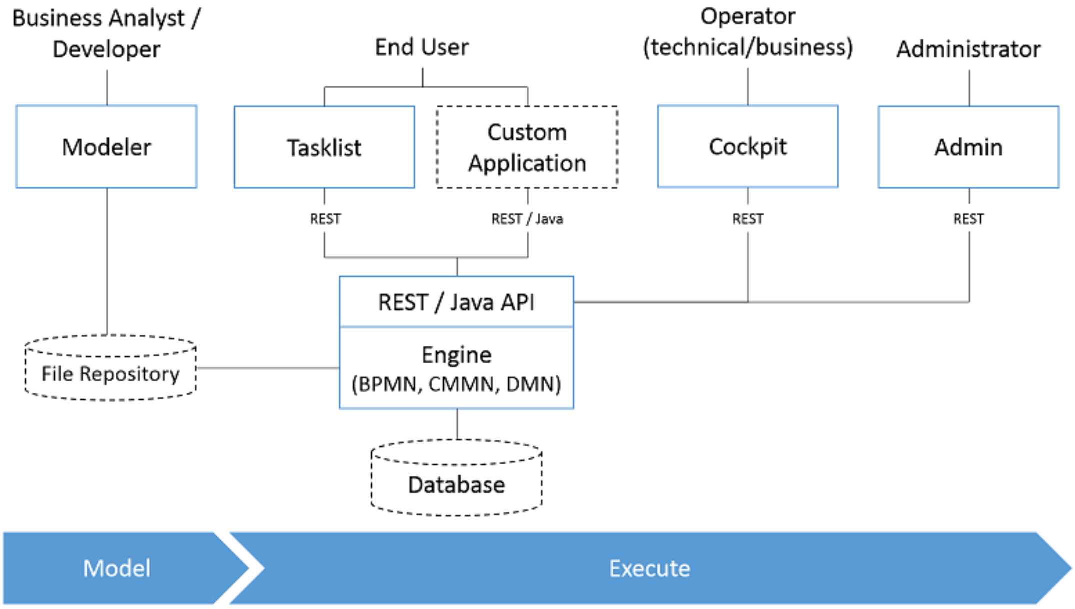


Figure 23. Overview of the main components of the Camunda framework

4.1.2 Flowable

Flowable is a business process engine written in Java that allows to deploy and execute BPMN 2.0 process definitions. It can be used as a JAR library or by means of its REST API. Flowable UI applications are also available like: Modeler, Admin, IDM and Task. It is open source licensed.

The core of Flowable is the engine API. An instance of the ProcessEngine can be created from a configuration file, giving access to the services for the Workflow/BPM methods. All the objects are thread safe and the services are stateless. The main services are:

- the RepositoryService, which is used to deploy process definitions;
- the RuntimeService, which deals with the instances of a process definition;
- the TaskService, which manages the tasks assigned to the users.

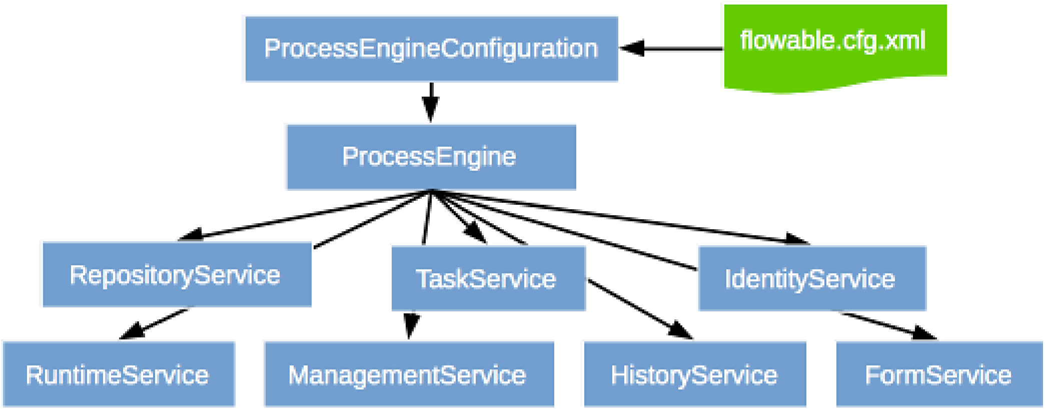


Figure 24. Overview of the Services available from the Flowable ProcessEngine

4.1.3 jBPM

jBPM is a toolkit to help automate business process. It is written in Java and is based on the BPMN 2.0 specifications. It is fully open source.

The project components can be grouped in three main layers:

- Execution, which consists of:
    - the Core Engine, that can be embedded in a custom application or used as a service;
    - the Human Task Service, an optional that takes care of the human task life cycle;
    - the Runtime Persistence Service, an optional service that persists the state of the processes;
- Modeling & Deployment, which is made of Web based tools to model and deploy processes, related forms and manage rules;
- Runtime Management, which is a Web based console that allows business user to manage the runtime processes.

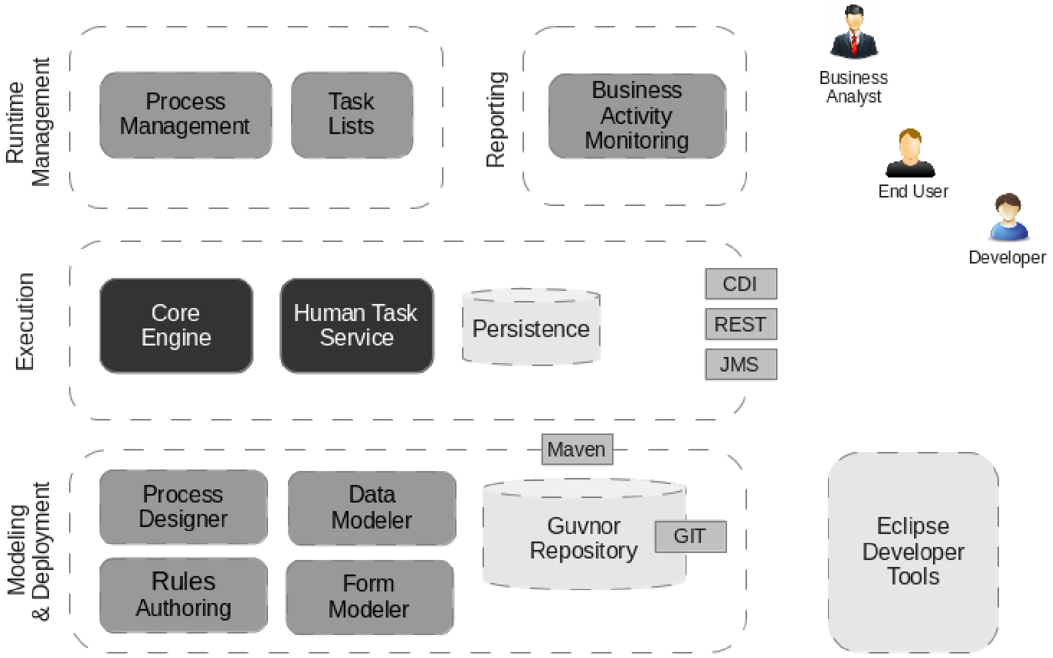


Figure 25. Overview of the components of the jBPM project

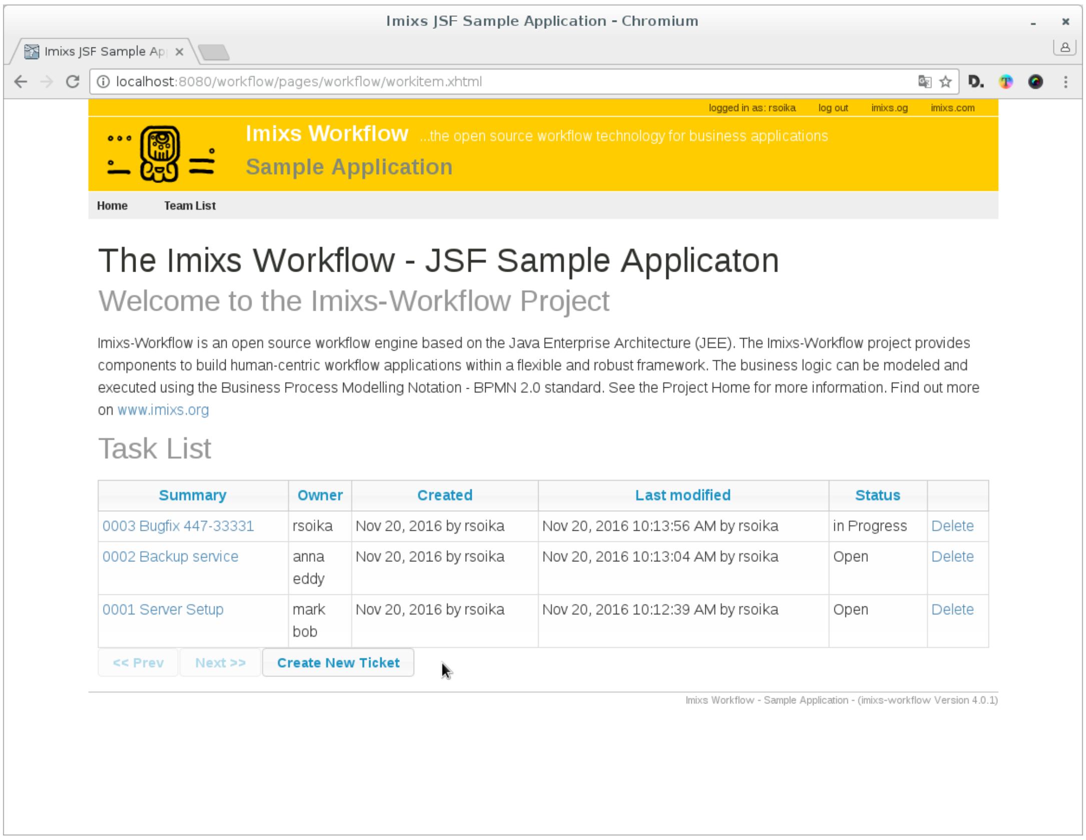
4.1.4 Imixs Workflow

Imixs-Workflow is an engine optimized for human-centric business process management. It is written in JAVA and it is open source.

The engine consists of different service components:

- the WorkflowService, i.e., the core component to create and update a process instance;
- the ModelService, i.e., the management component for BPM models;
- the DocumentService, i.e. the data access layer to store workflow related data;
- the ReportService, i.e., a service to create data reports.

All services can either be embedded in a custom application or accessed through the REST API.


Imixs JSF Sample Application - Chromium
Imixs JSF Sample Ap
localhost:8080/workflow/pages/workflow/workitem.xhtml
logged in as: rsoika
log out
imixs.og
imixs.com
Imixs Workflow
...the open source workflow technology for business applications
Sample Application
Home
Team List
<< Prev
Next >>
Create New Ticket

The Imixs Workflow - JSF Sample Applicaton

Welcome to the Imixs-Workflow Project

Imixs-Workflow is an open source workflow engine based on the Java Enterprise Architecture (JEE). The Imixs-Workflow project provides components to build human-centric workflow applications within a flexible and robust framework. The business logic can be modeled and executed using the Business Process Modelling Notation - BPMN 2.0 standard. See the Project Home for more information. Find out more on www.imixs.org

Task List

Summary	Owner	Created	Last modified	Status	
0003 Bugfix 447-33331	rsoika	Nov 20, 2016 by rsoika	Nov 20, 2016 10:13:56 AM by rsoika	in Progress	Delete
0002 Backup service	anna eddy	Nov 20, 2016 by rsoika	Nov 20, 2016 10:13:04 AM by rsoika	Open	Delete
0001 Server Setup	mark bob	Nov 20, 2016 by rsoika	Nov 20, 2016 10:12:39 AM by rsoika	Open	Delete

Imixs Workflow - Sample Application - (imixs-workflow Version 4.0.1)

Figure 26. The Imixs Workflow JSF Sample application

4.1.5 Orchestra

Orchestra is an open source system to orchestrate teams of experts as they complete complex projects with the help of automation. It is a Django Web app written in Python and is open source.

The key concepts of the system can be described as follows.

The system supports the managing of a process, which is called a **workflow**, a workflow is composed of steps. A step can be completed by an expert or a machine. A step can require a **review**, from a more experienced expert. Each step emits a JSON blob with structured data and have access to the data emitted by the previous steps it depends on. The instance of a workflow is a **project**, the instance of a step is a **task**, hence a project is a series of interconnected tasks.

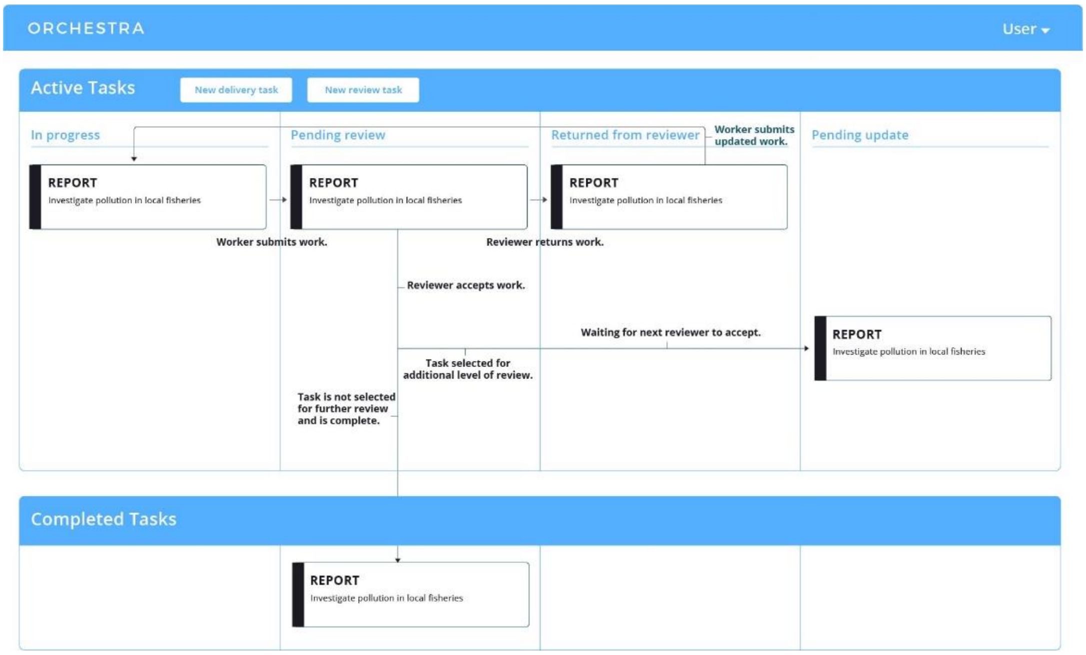

ORCHESTRA
User
Active Tasks
New delivery task
New review task
In progress
Pending review
Returned from reviewer
Worker submits updated work.
Pending update
REPORT
Investigate pollution in local fisheries
REPORT
Investigate pollution in local fisheries
REPORT
Investigate pollution in local fisheries
Worker submits work.
Reviewer returns work.
Reviewer accepts work.
Waiting for next reviewer to accept.
REPORT
Investigate pollution in local fisheries
Task selected for additional level of review.
Task is not selected for further review and is complete.
Completed Tasks
REPORT
Investigate pollution in local fisheries

Figure 27. Example of the Orchestra dashboard of a working expert

4.2 REUSE OF EXISTING ASSETS AND ARTIFACTS SYNTHESIS

A catalogue of assets, described according to an ontology, enables the possibility to enrich the contents of the catalogue itself by automatically executing a set of pre-defined tasks coordinated by the lifecycle management. The act of publishing an asset can then become just the first step of a complex process which reuse all the information contained in the Asset Manager to automatically provide the users of the IF ecosystem with additional assets and artifacts to be directly used and exploited.

Possible examples of the expected results are the following:

- adding an ontology can start an automatic documentation job which is able to add both its textual and visual representation to the ontology asset.
- The description of a Converter, in terms of expected source and destination specifications, can drive the actual synthesis of a Converter artifact which then provided to the user.
- The knowledge about the available Converters can be exploited to provide the users different representations of the content of an asset, each one obtained by exploiting a specific Converter.

Such a process goes under the general concept of Build automation, i.e., the automatic creation of a software build and its verification against predetermined tests. Build automation is the basis of Continuous Integration (CI) and Continuous Delivery (CD), i.e. the processes of validating and releasing, respectively, the software changes.

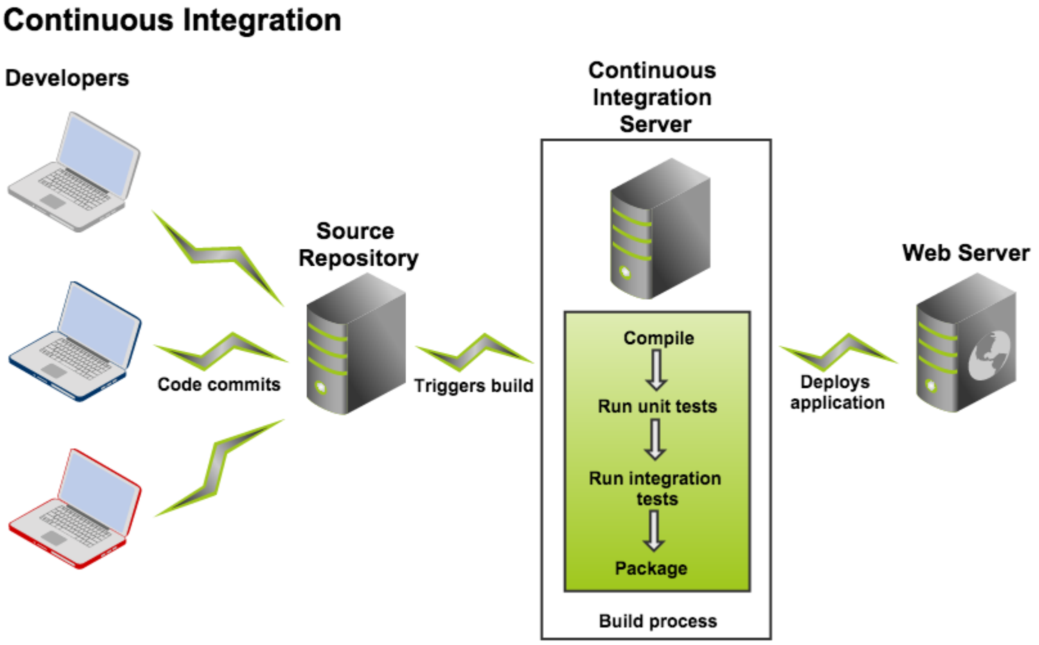


Figure 28. Schematic representation of the Continuous Integration methodology

Some of the existing tools to build CI/CD solutions are:

- Bamboo: https://www.atlassian.com/software/bamboo/
- GitLab: https://gitlab.com/
- Jenkins: https://jenkins.io/
- Travis CI: https://travis-ci.com/

Those solutions can be used to implement the automatic build process. Below is given a brief description for each of the aforementioned tools.

4.2.1 Bamboo

Bamboo is a continuous integration and deployment server. It assists software development teams by providing automated building and testing of software source-code status. It's a proprietary software of the Atlassian company[11].

Bamboo uses the following concepts to configure and order the actions in the workflow:

- Project, which has none, one, or more, plans.

---

[11] https://www.atlassian.com/

- Plan, which processes a series of one or more stages that are executed sequentially using the same repository. It specifies how the build is triggered, and the triggering dependencies between the plan and other plans in the project.
- Stage, which processes its jobs in parallel, on multiple agents, if available. It must successfully complete all its jobs before the next stage in the plan can be processed. It may produce artifacts that can be made available for use by a subsequent stage.
- Job, which processes a series of one or more tasks that are run sequentially on the same agent.
- Task, i.e., a small discrete unit of work, such as source code checkout, executing a Maven goal, running a script, or parsing test results. It's run sequentially within a job.

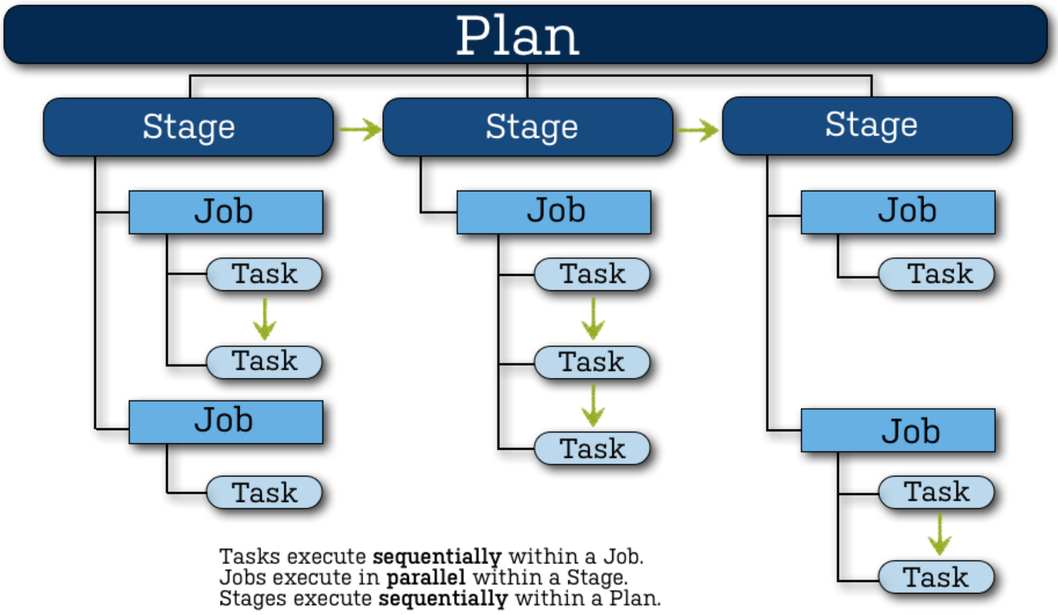

Plan
Stage
Stage
Stage
Job
Task
Task
Job
Task
Job
Task
Task
Task
Job
Task
Job
Task
Task
Tasks execute **sequentially** within a Job.
Jobs execute in **parallel** within a Stage.
Stages execute **sequentially** within a Plan.

Figure 29. Overview of the Bamboo continuous integration and deployment workflow

4.2.2 GitLab

GitLab is a web-based platform for the software development and operations lifecycle. It provides a Git[12] repository manager in conjunction with wiki, issue-tracking and CI/CD pipeline features. Its core functionalities are released as open source software.

GitLab CI/CD is a tool built into GitLab. Continuous Integration works by running a pipeline of scripts to build, test, and validate the code changes before merging them into the main branch.

Continuous Delivery consist of a step further CI, deploying the application to production at every push to the default branch of the repository.

---

[12] https://git-scm.com/

GitLab CI/CD is configured by a file placed at the repository's root. The scripts set in this file are executed by the GitLab Runner, a tool that works similarly to a computer terminal.

The scripts are grouped into jobs, and together they compose a pipeline. The scripts can be run sequentially or in parallel.

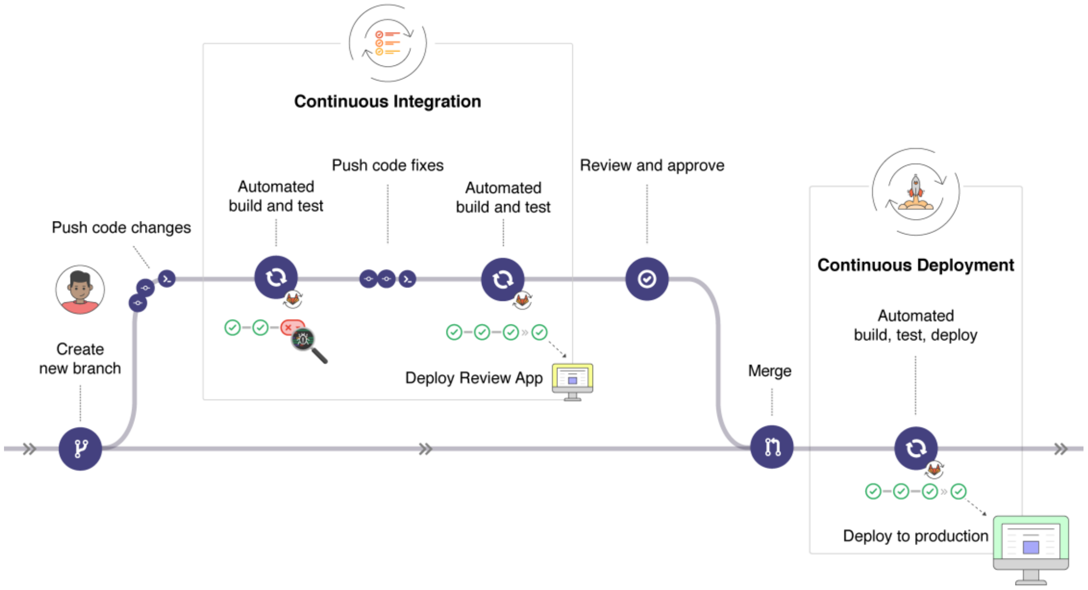


Figure 30. Example of a basic CI/CD workflow in Gitlab

4.2.3 Jenkins

Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software. The main concept of the Jenkins model is the Pipeline.

A Pipeline is a suite of plugins which supports implementing and integrating Continuous Delivery pipelines into Jenkins.

A Continuous Delivery (CD) pipeline is an automated expression of the process for getting software from version control right through to the end users. Every change to the software committed in source control goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the build through multiple stages of testing and deployment.

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines as code via the Pipeline domain-specific language (DSL) syntax.

The definition of a Pipeline is written into a text file, called a *Jenkinsfile*, which in turn can be committed to a project's source control repository. This is the foundation of "Pipeline-as-code"; treating the CD pipeline as part of the application to be versioned and reviewed like any other code.

The following concepts are key aspects of the Pipeline:

- Pipeline, i.e., a user-defined model of a CD pipeline. A Pipeline's code defines the entire build process, which typically includes stages for building an application, testing it and then delivering it.
- Node, which is a machine that is part of the Jenkins environment and is capable of executing a Pipeline.
- Stage, which is a block that defines a conceptually distinct subset of tasks performed through the entire Pipeline, e.g. "Build", "Test" and "Deploy" stages and it's used by many plugins to visualize or present Jenkins Pipeline status/progress.
- Step, i.e., a single task. Fundamentally, a step tells Jenkins what to do at a particular point in time, or step in the process. When a plugin extends the Pipeline DSL, that typically means the plugin has implemented a new step.

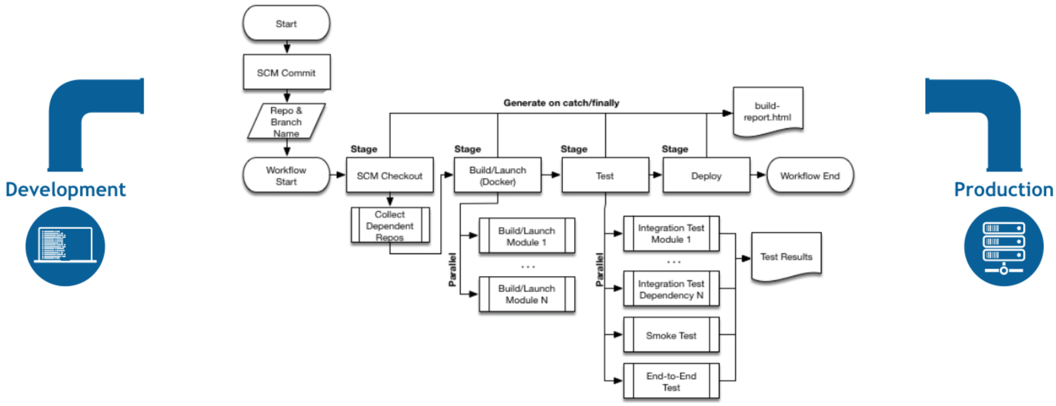


Figure 31. Example flowchart of a CD scenario modeled in Jenkins Pipeline

4.2.4 Travis CI

Travis CI is a hosted continuous integration and deployment system. It allows to test and deploy open source and private projects hosted at GitHub[13]. It is open source and has a free plan for open source projects.

Once activated on a GitHub project, it can be instructed by adding a script file in the repository. In this way it can automatically build and test code changes, providing immediate feedback on the success of the change. It can also manage deployments and notification.

When a build is run, Travis CI clones the GitHub repository into a brand-new virtual environment and carries out a series of tasks to build and test the code. If one or more of those tasks fail, the build is considered broken. If none of the tasks fail, the build is considered passed and Travis CI can deploy the code to a web server or application host.

The Travis CI model is based on the following key concepts:

---

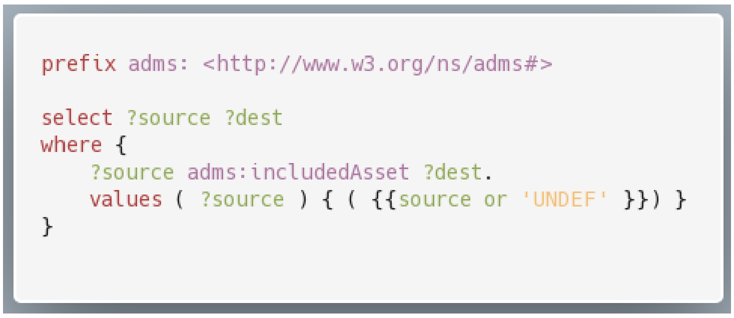
[13] https://github.com/

- phase, i.e., the sequential steps of a job. For example, the install phase, comes before the script phase, which comes before the optional deploy phase.
- job, i.e., an automated process that clones the repository into a virtual environment and then carries out a series of phases such as compiling the code, running tests, etc.
- build, i.e., a group of jobs. For example, a build might have two jobs, each of which tests a project with a different version of a programming language. A build finishes when all of its jobs are finished.
- stage, i.e., a group of jobs that run in parallel as part of a sequential build process composed of multiple stages.

4.3 IMPROVED METADATA EXPLORATION

With the aim of easing access to a potentially high number of asset metadata descriptions via common Web technologies, the Exploration API asset type (of Component Category) has also been defined. This asset type is used to describe parametric SPARQL queries, which are then exposed by the Asset Manager as Web APIs with a mechanism akin to grlc [15] and basil [16]. Using a normal HTTP GET request, the user can provide values to the parameters of the SPARQL query and obtain the results. Publishing parametric queries as assets in the catalogue allows a much higher level of control over the users' behaviors, since users are only allowed to call specific Web APIs according to their security permissions.

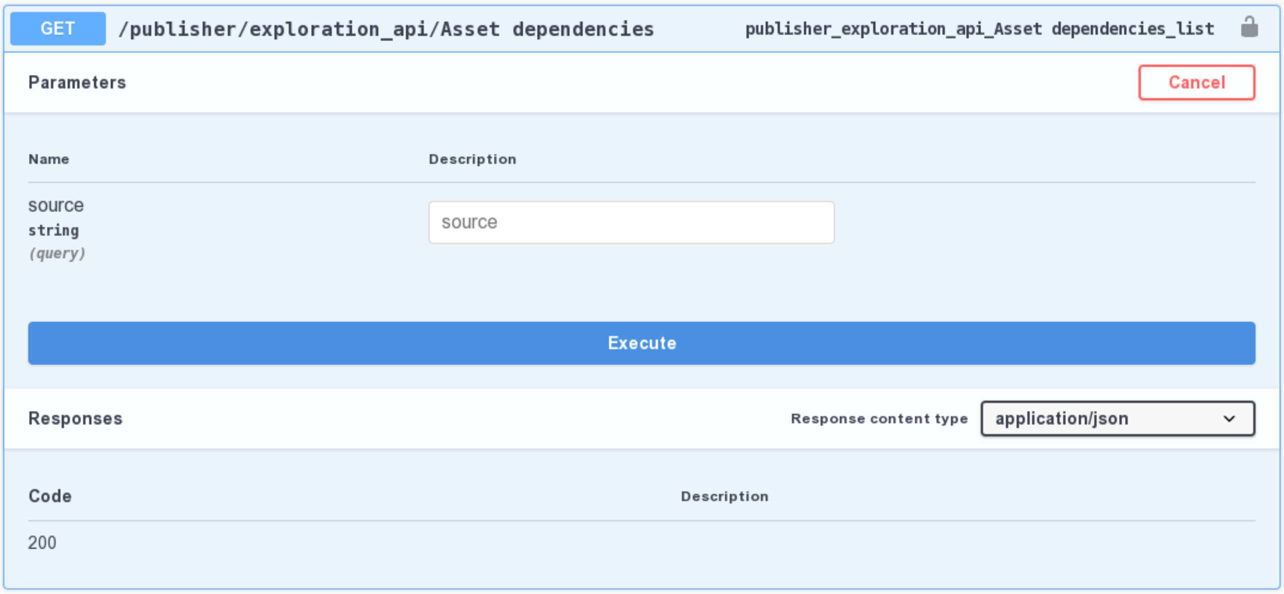
An example of a parametric SPARQL query is depicted in Figure 32. The query perform a basic dependency check on an asset, looking for all its "attachments". As can be noticed, the query is a template, and the parameter expression is enclosed in double braces. According to the parameters expressed in the query, the Swagger definition is automatically generated (Figure 33). This means that when a new Exploration API is published, a new API is added to the Swagger definition of the endpoint, and the fact that the underlying technology is based on the Semantic Web is completely hidden to the user.



```
prefix adms: <http://www.w3.org/ns/adms#>

select ?source ?dest
where {
    ?source adms:includedAsset ?dest.
    values ( ?source ) { ( {{source or 'UNDEF' }}) }
}
```

Figure 32. Parametric SPARQL query



`/publisher/exploration_api/Asset dependencies` `publisher_exploration_api_Asset dependencies_list`

Parameters

Name	Description
source string *(query)*	source

Responses — Response content type: application/json

Code	Description
200	

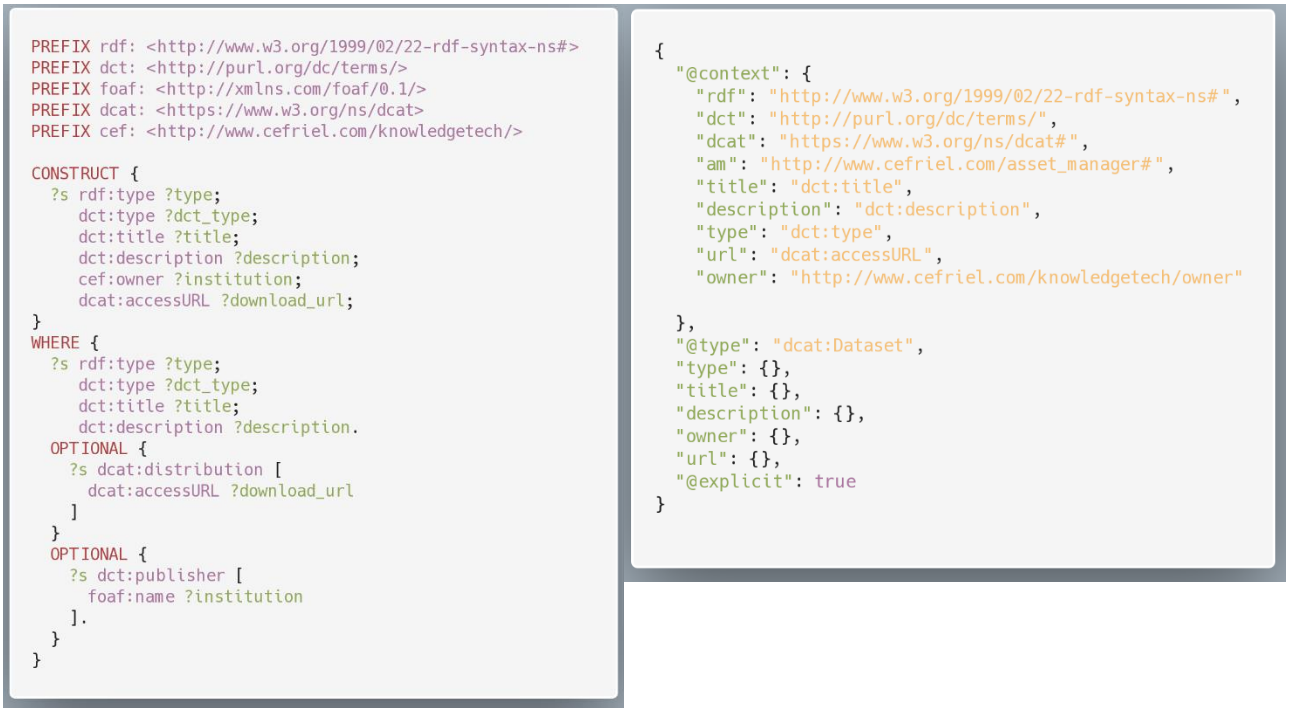
Figure 33. Swagger description of the “Asset Dependencies” Exploration API

As with other representations of Linked Data which describe directed graphs, a single directed graph can have many different serializations, each expressing exactly the same information. Developers typically work with trees, represented as JSON objects. While mapping a graph to a tree can be done, the layout of the end result must be specified in advance. A Frame can be used by a developer on a JSON-LD document to specify a deterministic layout for a graph.

However, given that JSON-LD represents one or more graphs of information, there is more than one way to frame the statements about several related subjects into a whole document. In fact, a graph of information can be thought of as a long list of independent statements (aka triples) that are not bundled together in any way.

The JSON-LD Framing API enables a developer to specify exactly how they would like data to be framed, such that statements about a particular subject are bundled together, delimited via { and }, and such that the subjects they relate to "nest" into a particular tree structure that matches what their application expects.

In the context of the Exploration API, such specifications can help bridging the gap between the “average programmer” and the Semantic Web, and can help developing API leveraging on the RDF graph managed by the Asset Manager. An example of the application of a JSON-LD Frame to the Exploration API is depicted in Figure 34. The query on the left is a SPARQL CONSTRUCT query which extract a set of basic information from the RDF graph, and uses them to construct and return to the caller a new, much simpler graph. When the JSON-LD Frame on the right is applied to the result, using the “@explicit” keyword we ensure that only the attributes explicitly listed in the Frame will be present in the output, and the result of the application of the Frame to the RDF output of the query is depicted in Figure 35.



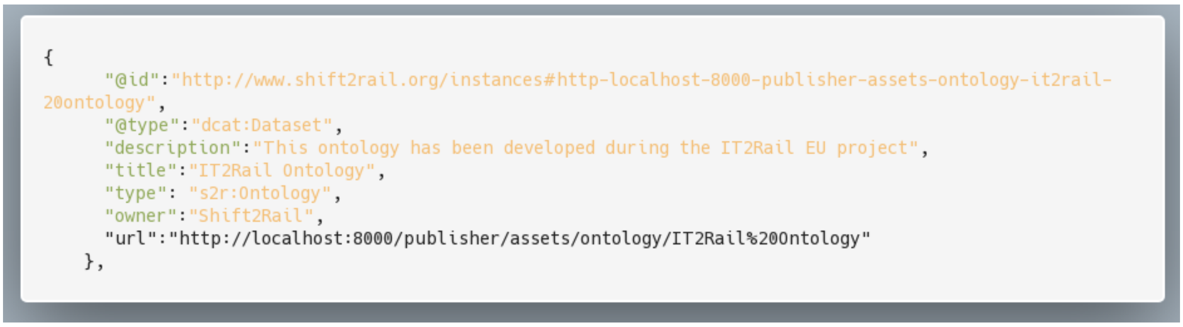
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dcat: <https://www.w3.org/ns/dcat>
PREFIX cef: <http://www.cefriel.com/knowledgetech/>

CONSTRUCT {
  ?s rdf:type ?type;
     dct:type ?dct_type;
     dct:title ?title;
     dct:description ?description;
     cef:owner ?institution;
     dcat:accessURL ?download_url;
}
WHERE {
  ?s rdf:type ?type;
     dct:type ?dct_type;
     dct:title ?title;
     dct:description ?description.
  OPTIONAL {
    ?s dcat:distribution [
      dcat:accessURL ?download_url
    ]
  }
  OPTIONAL {
    ?s dct:publisher [
      foaf:name ?institution
    ].
  }
}
```

```
{
  "@context": {
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "dct": "http://purl.org/dc/terms/",
    "dcat": "https://www.w3.org/ns/dcat#",
    "am": "http://www.cefriel.com/asset_manager#",
    "title": "dct:title",
    "description": "dct:description",
    "type": "dct:type",
    "url": "dcat:accessURL",
    "owner": "http://www.cefriel.com/knowledgetech/owner"

  },
  "@type": "dcat:Dataset",
  "type": {},
  "title": {},
  "description": {},
  "owner": {},
  "url": {},
  "@explicit": true
}
```

Figure 34. SPARQL CONSTRUCT query and JSON-LD Frames related to asset search



```
{
      "@id":"http://www.shift2rail.org/instances#http-localhost-8000-publisher-assets-ontology-it2rail-
20ontology",
      "@type":"dcat:Dataset",
      "description":"This ontology has been developed during the IT2Rail EU project",
      "title":"IT2Rail Ontology",
      "type": "s2r:Ontology",
      "owner":"Shift2Rail",
      "url":"http://localhost:8000/publisher/assets/ontology/IT2Rail%20Ontology"
    },
```

Figure 35. Exploration API output after applying a JSON-LD Frame

5. CONCLUSIONS AND NEXT STEPS

In this deliverable we have described the current status of two of the types of open source assets that are considered in the context of Shift2Rail: ontologies and mappings/annotations. We have also described the current status of the Asset Manager.

Taking that current situation into account, we have defined the first steps towards the provision of a well-defined methodological process for the truly collaborative edition of the Shift2Rail ontologies, which should overcome some of the current limitations identified in deliverables from other Shift2Rail projects. Based on the outcomes in terms of ontology development and annotation/mapping development in projects like IT2Rail and ST4RT, as well as the guidelines for governance proposed in the context of GOF4R, we have proposed an ontology development process that should rule the way in which existing ontologies will be made more publicly available and evolved, as well as to allow the creation of new ontologies. This ontology development process will be supported, from a technology point of view, by an evolution of one of the current state-of-the-art frameworks for collaborative ontology development, OnToology, which is currently strictly associated to the use of GitHub repositories and organisations. This tool is in the process of being enhanced to implement a more pure continuous integration pipeline using state-of-the-art tools, like Jenkins, and producing results that can be included in the Asset Manager.

Furthermore, this deliverable has presented the ongoing work in automating some of the activities in the process of ontology development and annotation/mapping generation. This development will continue during the rest of the period of execution of this project, and the tools will be made available to the Shift2Rail initiative for those willing to extend or create ontologies, as well as those generating mappings and annotations to provide access to data sources and services.

6. REFERENCES

[1] M. Fernández-López, A. Gómez-Pérez and N. Juristo, "METHONTOLOGY: From Ontological Art Towards Ontological Engineering," in *AAAI*, 1997.

[2] S. Staab, R. Studer, H.-P. Schnurr and Y. Sure-Vetter, "Knowledge Processes and Ontologies," *IEEE Intelligent Systems,* vol. 16, no. 1, pp. 26-34, 2001.

[3] M. Suárez-Figueroa, A. Gómez-Pérez and M. Fernández-López, "The NeOn Methodology framework: A scenario-based methodology for ontology development," *Applied Ontology,* vol. 10, no. 2, pp. 107-145, 2015.

[4] A. Gomez-Perez, M. Fernández-López and O. Corcho, Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web, Springer Verlag, 2004.

[5] M. Suarez-Figueroa, Neon Methodology for Building Ontology Networks: Specification, Scheduling and Reuse, IOS Press, 2012.

[6] T. Tudorache, J. Vendetti and N. Noy, "Web-Protege: A Lightweight OWL Ontology Editor for the Web," in *Fifth OWLED Workshop on OWL: Experiences and Directions*, 2008.

[7] M. Dragoni, A. Bosca, M. Casu and A. Rexha, "Modeling, Managing, Exposing, and Linking Ontologies with a Wiki-based Tool," in *Proceedings of LREC*, 2014.

[8] L. Halilaj, N. Petersen, I. Grangel-González, C. Lange, S. Auer, G. Coskun and S. Lohmann, "VoCol: an integrated environment to support version-controlled vocabulary development," in *Proceedings of the 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, 2016.

[9] A. Alobaid, D. Garijo, M. Poveda-Villalón, I. Santana-Perez, A. Fernández-Izquierdo and O. Corcho, "Automating ontology engineering support activities with OnToology," *Journal of Web Semantics,* 2018.

[10] D. Garijo, "WIDOCO: A wizard for documenting ontologies," in *International Semantic Web Conference*, 2017.

[11] M. Poveda-Villalón, A. Gómez-Pérez and M. Suárez-Figueroa, "OOPS! (OntOlogy Pitfall Scanner!): An on-line tool for ontology evaluation," *Int. J. Semant. Web Inf. Syst. (IJSWIS),* vol. 10, no. 2, pp. 7-34, 2014.

[12] S.Grayson, "Novel2Vec: Characterising 19th Century Fiction Via Word Embeddings," University College Dublin, Dublin, Ireland, 2016.

[13] V. L. Mikolov.T, "Exploiting Similarities among Languages for Machine Translation," 2013.

[14] J. Barnett, R. Akolkar, R. Auburn, B. M, B. D.C, C. J, M. S, L. T, M. Helbing, R. Hosn, R. T.V, R. K, N. Rosenthal and J. Roxendal, "State Chart XML (SCXML): State Machine Notation for Control Abstraction," W3C Recommendation, [Online]. Available: https://www.w3.org/TR/scxml/.

[15] A. Meroño-Peñuela and R. Hoekstra, "grlc makes GitHub taste like linked data APIs," in *European Semantic Web Conference*, 2016.

[16] E. Daga, L. Panziera and C. Pedrinaci, "Basil: A cloud platform for sharing and reusing SPARQL queries as Web APIs," in *CEUR Workshop Proceedings*, 2015.