TECHNISCHE UNIVERSITÄT WIEN

Diplomarbeit

Leveraging Semantic Technologies for the application in Multi-Domain Digital Twins

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs (Dipl.-Ing. oder DI) unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Manfred Grafinger
(Institut für Konstruktionswissenschaften und Produktentwicklung)

sowie unter der Betreuung von
Projektass. Dipl.-Ing. Stefan Dumss BSc
(Institut für Konstruktionswissenschaften und Produktentwicklung)

eingereicht an der Technischen Universität Wien
Fakultät für Maschinenwesen und Betriebswissenschaften

David Kern BSc
01226528
(066 482)

Ich nehme zur Kenntnis, dass ich zur Drucklegung meiner Arbeit unter der Bezeichnung

Diplomarbeit

nur mit Bewilligung der Prüfungskommission berechtigt bin.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass die vorliegende Arbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen von mir selbstständig erstellt wurde. Alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, sind in dieser Arbeit genannt und aufgelistet. Die aus den Quellen wörtlich entnommenen Stellen, sind als solche kenntlich gemacht.

Ich nehme zur Kenntnis, dass die vorgelegte Arbeit mit geeigneten und dem derzeitigen Stand der Technik entsprechenden Mitteln (Plagiat-Erkennungssoftware) elektronisch-technisch überprüft wird. Dies stellt einerseits sicher, dass bei der Erstellung der vorgelegten Arbeit die hohen Qualitätsvorgaben im Rahmen der geltenden Regeln zur Sicherung guter wissenschaftlicher Praxis „Code of Conduct“ an der TU Wien eingehalten wurden. Zum anderen werden durch einen Abgleich mit anderen studentischen Abschlussarbeiten Verletzungen meines persönlichen Urheberrechts vermieden.

Wien, August 2023 David Kern BSc

Danksagung

Ich möchte hier die Gelegenheit nützen allen zu danken die mich während der Erstellung dieser Arbeit und während meines gesamten Studiums fachlich sowie persönlich unterstützt haben.

Zuerst bedanke ich mich bei meinem Diplomarbeitsbetreuer Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Manfred Grafinger für die Möglichkeit mich mit diesem spannenden Thema zu beschäftigen. Desweitern bedanke ich mich bei Dipl.-Ing. Stefan Dumss, für die hervorragende fachliche Betreuung und die spannenden Diskussionen.

Bedanken möchte ich mich ebenfalls bei meinen Studienkollegen:innen die mich auf dem Weg durchs Studium begleitet haben. Ohne euch wäre dieser Weg nur halb so lustig gewesen. Besonders möchte ich Matteas Jelović und Phillip Rückeshäuser die mich in der finalen Phase dieser Arbeit unterstützt haben.

Anschließend möchte ich mich bei meiner Freundin Michaela bedanke, dass sie diese doch nicht immer leichte Zeit mit mir durchgestanden hat und mir stehts den Rückhalt gegeben hat den ich brauchte.

Abschließend möchte ich meiner Familie danken, ohne euch wäre das Ganze nicht möglich gewesen. Im Besonderen möchte ich meinen Eltern danken. Ihr standet mir nicht nur während des Studiums stets mit Rat und Tat beiseite. Danke Mama, Danke Papa!

Abstract

The concept of a Digital Twin (DT) has gained traction recently. A DT is the virtual representation of a physical asset and allows for data exchange. A vital part of such DTs are simulation models. To facilitate large-scale digital twins, multiple simulation models can be used. This enables modeling subsystems separately rather than within a large model. An example of such a large-scale domain is the railway domain. As it consists of multiple different heterogeneous domains and subsystems, data exchange between the various domains poses a problem as they use different standards and tools. Semantic data model offers a technique to integrate data from different domains in a machine-interpretative form. In order to achieve that, semantic technologies can be used to develop an ontology.

This thesis sets out to create an ontology for the application in the Rail4Future (R4F) project to describe components of a simulation model as well as provide additional information about such models. It follows a state-of-the-art methodology for ontology engineering. Finally it proposes a workflow to integrate the ontology in creating simulation topologies. It provides a tool that enables non-ontology experts to integrate their data into the Knowledge Graph (KG).

Kurzfassung

Das Konzept des Digitaler Zwilling (Digital Twin, DT) hat in letzter Zeit an Zugkraft gewonnen. Ein DT ist die virtuelle Darstellung einer physischen Anlage und ermöglicht den Datenaustausch. Ein wesentlicher Bestandteil solcher DTs sind Simulationsmodelle. Um groß angelegte digitale Zwillinge zu ermöglichen, können mehrere Simulationsmodelle verwendet werden. Dies ermöglicht es, Teilsysteme separat zu modellieren, anstatt sie in einem großen Modell zusammenzufassen. Ein Beispiel für einen solchen groß angelegten Bereich ist der Eisenbahnbereich. Da dieser aus mehreren verschiedenen heterogenen Domänen und Teilsystemen besteht, stellt der Datenaustausch zwischen den verschiedenen Domänen ein Problem dar, da unterschiedliche Standards und Werkzeuge verwendet werden. Semantische Datenmodell bietet eine Technik zur Integration von Daten aus verschiedenen Bereichen in einer maschineninterpretierbaren Form. Um dies zu erreichen, können semantische Technologien zur Entwicklung einer Ontologie verwendet werden.

Ziel dieser Arbeit ist es, eine Ontologie für die Anwendung im Projekt Rail4Future (R4F) zu erstellen, um Komponenten eines Simulationsmodells zu beschreiben und zusätzliche Informationen über solche Modelle bereitzustellen. Dabei wird eine dem Stand der Technik entsprechende Methodologie für das Ontologie-Engineering verwendet. Schließlich wird ein Arbeitsablauf vorgeschlagen, um die Ontologie in die Erstellung von Simulationstopologien zu integrieren. Die Arbeit bietet ein Werkzeug, mit dem Nicht-Ontologie-Experten ihre Daten in den Knowledge Graph (KG) integrieren können.

Contents

Acronyms

API	Application Programming Interface
AQL	ArangoDB Query Language
ASCII	American Standard Code for Information Interchange
CSV	Comma-separated values
CWA	Closed-World Assumption
DC	Dublin Core
DCAT	Data Catalog Vocabulary
DT	Digital Twin
ERA	European Union Agency for Railways
FOAF	Friend of a Friend
FMI	Functional Mockup Interface
FMU	Functional Mockup Unit
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
IRI	Internationalized Resource Identifier
JSON	JavaScript Object Notation
JSON-LD	JSON for Linking Data
KG	Knowledge Graph
OM	Ontology of units of Measure
ORSD	Ontology Requirements Specification Document
ODP	Ontology Design Pattern
OWA	Open-Word Assumption
OWL	Web Ontology Language
QUDT	Quantity, Unit, Dimension and Type
QUDV	Quantities, Units, Dimensions, Values
RDF	Resource Description Framework
RDFS	RDF Schema
RIF	Rule Interchange Format
R4F	Rail4Future
Shacl	Shapes Constraint Language
SoSa	Sensor, Observation, Sample, and Actuator
SPARQL	SPARQL Protocol And RDF Query Language
SPDX	Software Package Data Exchange
SSP	System Structure and Parameterization
SWRL	Semantic Web Rule Language
TCP/IP	Transmission Control Protocol/Internet Protocol
UI	User Interface

UIC	International Union of Railways
UML	Unified Modeling Language
UNA	Unique Name Assumption
UO	Units Ontology
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSD	XML Schema Definition

1. Introduction

1.1. Motivation and problem statement

The railroad industry is considered one of the key instruments to achieve climate targets. In response to growing demand, the railway industry has made significant investments in both infrastructure and rolling stock. These endeavors aim to address the need for increased capacity while maintaining the railways' reliability and safety as a preferred mode of transportation. However, building new infrastructure is not only cost-intensive but also not a viable long-term solution. Instead, stakeholders focused on optimizing the utilization of their current infrastructure. This approach includes extending the lifetime of different infrastructure systems, such as tracks, turnouts, tunnels, and bridges, by leveraging information and communication technology to a greater extent. Among those technologies, the Digital Twin (DT) is widely regarded as one of the most promising technologies. The DT continually monitors the state of the replicated entity throughout its life cycle, offering a comprehensive understanding of its functionality and operational status and providing a way of predicting its future behavior. Currently, most DT are being developed for specific domains with closed architectures, often relying on proprietary solutions, which hinders their interoperability. This interoperability is particularly important for railway infrastructure systems, given that they consist of a large quantity of subsystems that function collaboratively and interactively. To meet these requirements, it is often beneficial to integrate models from various domains in order to configure a simulation. Multiple domains also mean different organizations and experts who develop the models using different simulation tools. This is one reason that integrating and connecting simulations presents a challenging task since the person configuring the simulator is typically different from the person constructing the models. With this, configuring a simulation topology includes analyzing the available information to infer if and how different models can be connected. Besides, gaining knowledge of available information and domain knowledge connecting these simulation models represents a time-consuming task. Another challenge is the distribution of data across different organizations and systems without common semantics. This fact further complicates the interoperability of different data sources. A jointly agreed data semantic and data model is needed to assist in interconnecting these heterogeneous resources. Semantic data models can be used to store context alongside the data itself in a machine-readable form. The representation of knowledge is accomplished by defining entities and their relations, where each entity and connection has an understood meaning. The exact interpretation of relationships and entities can be defined by an ontology. An ontology provides a way of formalizing a particular domain and representing it in a machine-readable form.

1.2. Aim of the work

In order to facilitate the aforementioned interoperability this work sets out to develop a semantic data model for the usage in the Rail4Future (R4F) project. The work presented in this thesis builds upon the current state-of-the-art methodologies for ontology engineering. Besides providing an overview of the existing literature for applying semantic technologies in the railway domain and multi-domain simulation and distributed digital twins, the thesis presents the development and application of the ontology. To assist with the mentioned task, the work aims to answer the following question:

RQ1 How are semantic technologies currently used to describe the railway domain?

RQ2 Are there applications of semantic technologies in multi-domain simulations?

RQ3 How can semantic technologies be used for metadata sharing of distributed heterogeneous data sources?

RQ4 How can domain experts with limited knowledge of ontologies utilize these semantic technologies?

In order to answer these questions, the thesis is structured in the following way:

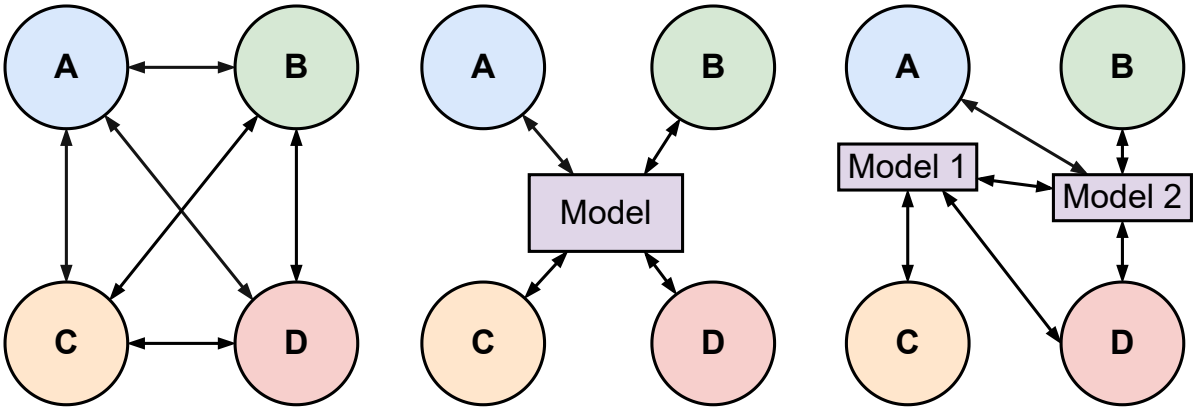
- Chapter 2 lays the theoretical foundation for the ontology development process.
- Chapter 2.5 presents the current state-of-the-art for ontologies in the railway domain and multi-domain simulations.
- Chapter 3 documents the development of the ontology based on the NeOn methodology.
- Chapter 3.2 discusses the application of the ontology in the Rail4Future project.

2. Theoretical foundations and related work

This chapter provides an overview of the theoretical foundation needed for this thesis. It sets out the need for semantic interoperability and introduces the concept of ontologies. Further, it explains the foundation of the semantic web stack and summarizes the core technologies for creating and using computer ontologies, and outlines technical notation used throughout the rest of the thesis. The concept of a Digital Twin (DT) and the Functional Mockup Unit (FMU) Standard are introduced as a way of building modular multi-domain digital twins.

2.1. Semantic Interoperability

Interoperability plays a crucial role in information management as most of our current information needs to involve drawing data from multiple sources. Multiple sources often also mean numerous different interfaces and data models. Figure 2.1a shows the problem if each source has its data model. This leads to a significant number of interfaces and mapping between these standards. These mappings are no longer necessary with a common data model, as seen in Figure 2.1b, this would be an ideal case. However, a more common approach is shown in Figure 2.1c. Here multiple sources use the same model, and a mapping between the different models can be achieved.



(a) Without Standard Data Model

(b) With Standard Data Model

(c) With multiple Standard Data Models

Figure 2.1.: Interfaces in Complex Systems. Source: Adapted from [1]

Sheth [2] discusses four different levels of interoperability — system, syntax, structure, and semantic.

- **System interoperability** refers to the ability of different systems to connect and communicate with each other physically. This level of interoperability has been addressed through common communication standards such as Transmission Control Protocol/Internet Protocol (TCP/IP) and standardized data encoding formats.
- **Syntactic interoperability** refers to the capability of different systems to share and understand specific document syntaxes, such as Extensible Markup Language (XML) or Hypertext Markup Language (HTML). This allows the software to extract pieces of data using concepts in these syntaxes.
- **Structural** or **Schema interoperability** is achieved when systems follow the same data model or schema, allowing data to be categorized and its meaning to be inferred through context. This is commonly achieved through adherence to modern computing standards.
- **Semantic interoperability** enables systems to understand the meaning of data without a predefined schema by defining all necessary knowledge within the data, allowing systems to interpret and integrate it.

According to Tutcher [1], achieving complete semantic interoperability is unrealistic as each concept must be defined by a multitude of other concepts, creating an infinite number of necessary axioms. Another problem with semantic interoperability is the so-called semantic clash, where two or more concepts or terms used in a specific context or system have different meanings or interpretations, leading to confusion or misinterpretation. For example, in one context, the term *"load"* might refer to the weight or force that a structure or component is designed to bear. In contrast, in another context, *"load"* might refer to the amount of energy a machine or system uses. One possible solution to overcome this problem is the usage of ontologies, which will be discussed in Section 2.2.

2.2. Ontology

The term ontology originates in philosophy and is a branch of metaphysics concerned with the nature of existence, including the relationships between entities and categories. This also includes the question of which entities exist and how they can be grouped or subdivided according to their similarities and differences.

In computer science, the term refers to the representation of knowledge as a set of concepts within a specific domain and the relationship between these concepts. One of the most prevalent and most cited definitions of an ontology is that of Gruber [3, p. 199], who defines an Ontology as an *'explicit specification of a conceptualization'*. Studer

et al. [4, p. 184] extended that definition stating that *'An ontology is a formal, explicit specification of a shared conceptualization'* also including Borst [5] definition who added that the conceptualization should be expressed formally, i.e., machine-readable, as well as it should be a shared view rather than an individual view. Figure 2.2 shows an example of a simple ontology and some sample instances. It offers some conceptualization in the Pop Music domain and the respective relationships.

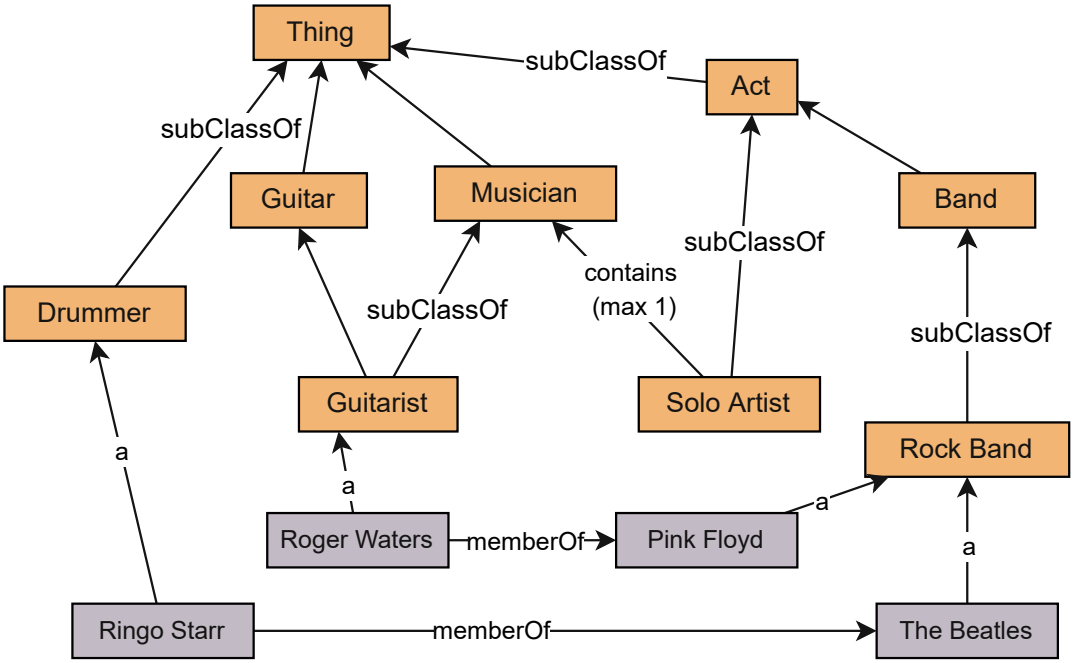


Figure 2.2.: Example of an Ontology. Source: [1, p. 24]

Although ontologies sometimes include individuals or instances, as seen in Figure 2.2, use the same languages (RDF, RDFS, etc.) and tools they are distinct to knowledge bases [6]. Knowledge bases build upon ontologies. Ontologies provide the building blocks, i.e., the vocabulary and the formal specification of that vocabulary, to create a knowledge base. This distinction is not always apparent, as individuals can belong to an ontology. As an example, *Vienna*, as an instance of the class *City*, can be part of a tourist ontology, whereas a specific train connection to that city is not [6].

Ontologies are usually expressed in an ontology language, which offer pre-set rules and constructs for expressing knowledge semantically. Figure 2.3 illustrates this spectrum. Ranging from simple lists and glossaries that provide a way of sharing knowledge in natural language to aid information exchange, but the semantics are not captured [1], [7]. Taxonomies enable the hierarchical structuring of the data. Thesauri extend that structure by allowing other statements about the concept to be made. For more semantic richness, conceptual models and logic theory can be utilized. Ontologies expressed as a conceptual model introduce a distinction between classes and individuals as well as generalize relations. Ontologies as conceptual models are the basis for machines to infer

information based on concepts and relations. Logical theory ontologies can be viewed as conceptual model elements focusing on real-world semantics and extending these models with axioms and rules.

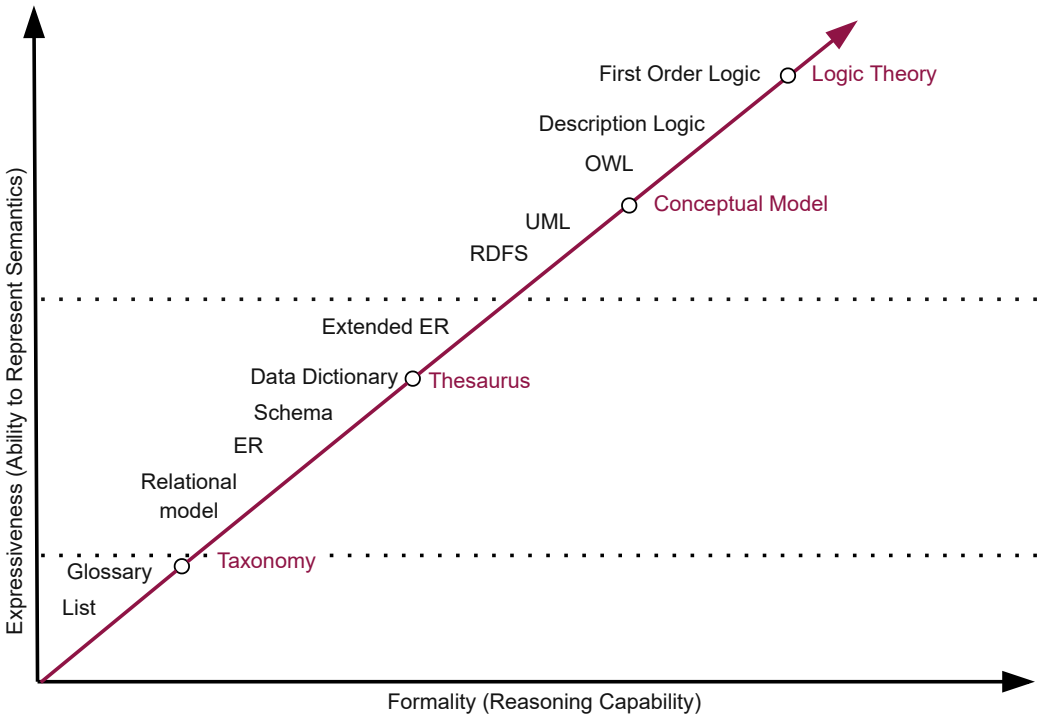


Figure 2.3.: Ontology Spectrum. Source: Adapted from [7]

2.2.1. Ontology Types

Ontologies can be classified according to their level of dependence on a specific task or viewpoint [8]. Figure 2.4 shows this classification.

- **Top-Level ontology or Upper Level** Ontologies are used to describe general concepts common to all domains, i.e., time or events. These ontologies offer the highest level of re-usability as they are independent of a particular problem or domain.
- **Domain ontology and task ontology** represents knowledge about a generic domain, such as medicine or biology, or a task, such as selling.
- **Application ontology** are developed with a specific domain or task in mind. These often combine specializations of both the corresponding domain and task ontologies.

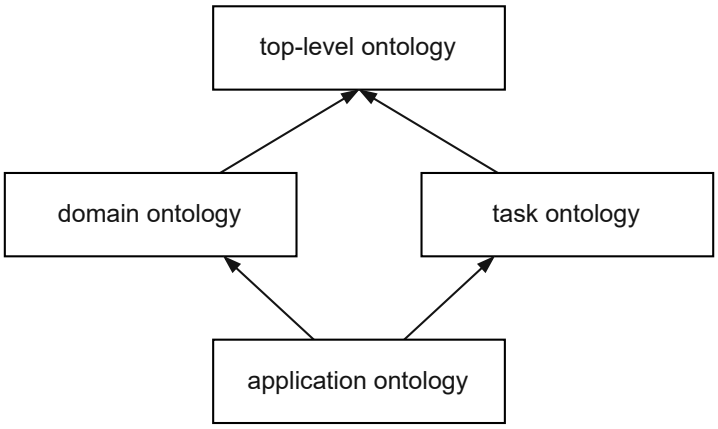


Figure 2.4.: Classification of different types of ontology. Source: [8, p. 145]

2.2.2. Methodologies for Ontology Development

Ontology development methodologies guide the process of the construction of an ontology. Ontologies can either be built from scratch or by reusing other available ontologies. Several specific methodologies for ontology development exist. The following gives a compact overview of some of the most popular methodologies.

METHONTOLOGY [9] is one of the first methodologies that was influenced by software engineering methodologies [10]. It presents a prototyping life cycle by defining a set of activities to build ontologies from scratch. These activities include requirement specification, knowledge acquisition, domain conceptualization, integration, implementation, evaluation, and documentation. METHONTOLOGY includes a list of activities to reuse ontologies. However, it does not provide a detailed guideline for such a process. In general, METHONTOLOGY is a versatile approach for creating ontologies that simplifies the process of modifying and expanding the ontology for developers.

Ontology Development 101 [11] is an iterative method using seven steps to develop an ontology. After an initial development phase, the guideline highlights revising and refining the developed ontology. The proposed steps include the definition of domain and scope, reuse of existing ontologies, enumeration of essential terms, define classes and hierarchies, the definition of class properties, the definition of facets to the classes, and the creation of instances. The methodology uses a simple example to emphasize each step. Furthermore, the authors use Protégé-2000 [12] as an ontology development tool.

The NeON Methodology [13] is a scenario-driven ontology engineering framework. It uses the so-called divide and conquer strategy by breaking the general problem into smaller sub-problems. The methodology describes nine scenarios composed of processes and activities such as the potential for ontology reuse, resources available, and intended application [1]. The NeON methodology promotes the development

of "ontology networks," which are collections of focused ontologies interconnected to accomplish a specific task. Besides the scenarios, the NeOn methodology defines two +ontology life cycle models. A waterfall model where concrete stages must be completed before the next stage. This model assumes that the requirements are completely known since backtracking is not allowed until the maintenance phase. The iterative-incremental model comprises several iterations, where each iteration uses a waterfall model.

2.3. Semantic Web Stack

According to the World Wide Web Consortium (W3C), the Semantic Web *'provides a framework that allows data to be shared and reused across application, enterprise, and community boundaries'* [14]. It is an extension to the World Wide Web and aims to make Internet data machine-readable, thus creating a web of data rather than documents. The intended meaning, i.e., the semantics, is explicitly specified in a machine-processable form to achieve this. This formal semantics enables the system to conclude from the information based on its context [15]. The so-called Semantic Web Stack, as depicted in Figure 2.5, illustrates the technologies and principal languages enabled in the semantic web. Each layer exploits the features of the levels beneath it.

In this section, a brief overview of the lower layers, i.e., *Identifiers*, *Characters* and *Syntax* is given, as well as a short overview over the top and side of the stack. The core technologies are covered in more detail in the following sections.

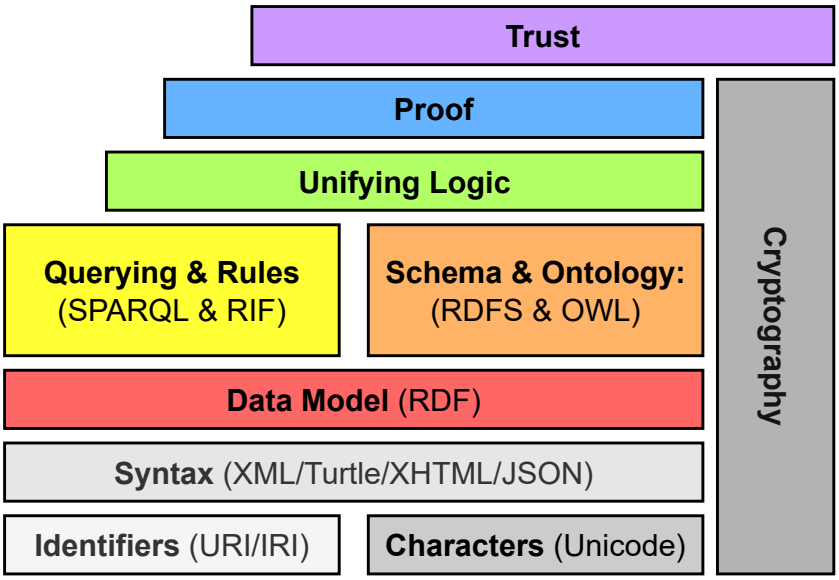


Figure 2.5.: The Semantic Web Stack. Source: Adapted from [16, p. 7]

Characters: Unicode builds the basis for the Word Wide Web and thus the Semantic Web as it is the default encoding of HTML and XML. It provides a standard for representing text by assigning a unique number to every character. It allows the exchange of text data internationally without corruption and is implemented in all modern operating systems and computer languages [17].

Identifiers: A Uniform Resource Identifier (URI) is a string of characters used to identify an abstract or physical resource [18]. The most common URI is a Uniform Resource Locator (URL). It is used to identify a location on the internet and specifies how it can be accessed [19]. The other type of identifier is a Uniform Resource Name (URN). URN are used to independently and persistently identify the same resource over time, unlike a URL [20]. A valid URI can only consist of ASCII characters. To overcome this problem, the Internationalized Resource Identifier (IRI) has been defined as a complement to URI. An IRI can contain any sequence of Unicode characters [21]. Figure 2.6 shows the relation among IRI, URI, URL and URN.

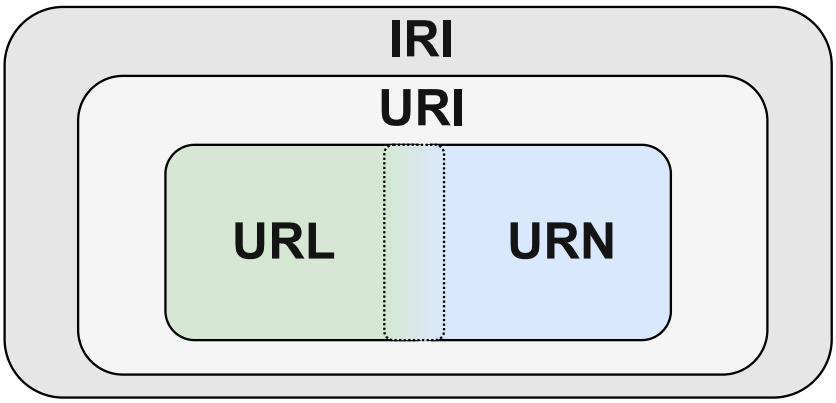


Figure 2.6.: Relation among IRI, URI, URL and URN. Adapted from [22]

Syntax: The syntax of a language defines the set of rules for the structure of that language. This allows a computer to parse content automatically. Generic syntaxes such as XML and JavaScript Object Notation (JSON) are widely used in modern systems and the current web. Although these syntaxes allow for legacy tools and can be utilized in the semantic web, custom syntaxes, i.e., Turtle, have been created [16]. Section 2.3.1 will discuss some of these syntaxes.

Querying and Rules : Like data stored in a relational database, data encoded in RDF needs to be queried and processed according to some rules. The current standard to query Resource Description Framework (RDF) data is SPARQL Protocol And RDF Query Language (SPARQL). SPARQL provides a way to write queries ranging from simple graph pattern matching to complex queries [16], [23]. Rule Interchange Format (RIF) is the current standard format for interchanging rules over the Web and was developed to facilitate rule set integration and synthesis. [16], [24].

Unifying Logic: The role of the unifying logic layer is to act as an interoperability layer and to enable the lower-level technologies to be used as a whole [16].

Proof: One idea of the semantic web is to enable software agents to combine data from different, decentralized sources and apply reasoning and querying to accomplish a goal. With the proof layer, the agent should also be able to answer how it concluded the decision. This proof could be used by a client to validate the procedure or information used [16],[25].

Trust: This layer refers to the trust in the information source, not the trust in the decision itself, as it is with all automated decision-making. This layer is strongly linked with the proof layer beneath it since it defines which data sources can be trusted in a proof [16],[26].

Cryptography: Cryptography covers all layers of the semantic web, indicating that it is somewhat relevant for all technologies. In terms of enabled technology, these can be borrowed from the standard web, such as digital signatures, public-key encryption/decryption algorithms, or secure protocols [16].

2.3.1. Resource Description Framework

The Resource Description Framework (RDF) is one of the core technologies of the semantic web. It enables systems to share data over the internet while still preserving their original meaning. In contrast to XML, RDF is a data model rather than a serialization format. XML represents data in a tree structure, therefore presenting data in an ordered hierarchical structure. RDF, on the other hand, represents data as a set of triples. A triple models two entities or concepts, called a *Resource*, and their relationship in the form of *Subject*, *Predicate*, and *Object* [1] as shown in Figure 2.7.

Figure 2.7.: Basic RDF triple.

Consider the statement '*Isaac Newton is a physicist*', Listing 2.1 and 2.2 show two valid ways of modeling that statement in XML. Listing 2.3 shows the same statement encoded in RDF. The triples in Listing 2.3 are informally expressed in pseudo-code. The examples show that there are multiple ways to encode data with XML, whereas, with RDF, there is only one possible way suggesting that the former is more syntactic and the latter more semantic [15].

Listing 2.1: Example-1 XML Markup

```
<person>
<name>Isaac Newton</name>
<job>physicist</job>
</person>
```

Listing 2.2: Example-2 XML Markup

```
<person name="Isaac Newton">
<job>physicist</job>
</person>
```

Listing 2.3: Example-3 RDF Markup (Pseudocode)

```
<person1> <is a> <Person>.
<person1> <name> <"Iscaac Newton">.
<person1> <job> <"physicist">.
```

As mentioned above, RDF is a data model used to make statements about resources. A set of triples is called an RDF Graph, which can be visualized as a directed labeled graph where subjects and objects are drawn as labeled vertices and predicates as directed, labeled edges [16]. Vertices or nodes are either IRIs, literals, or blank nodes [27]. Edges are denoted as IRI. Literals are basic values that are not IRIs and can only appear in the object position of a triple. Literals can be associated with a data type. [27] provides a non-exhaustive list of datatypes which, in addition to defining its own, re-uses the XML schema build-in datatypes such as *boolean*, *integer* and *date*. + RDF is limited to only represent binary relationships, i.e., one subject, one object. This makes modeling some real-world phenomenon difficult. There are two mechanisms to help model n-ary relations. RDF Reification decomposes RDF triple into separate RDF entities. In that way, it is possible to make statements about statements. Another solution is using blank nodes as an intermediate resource.

Figure 2.8 shows an RDF-Graph with all three node types.

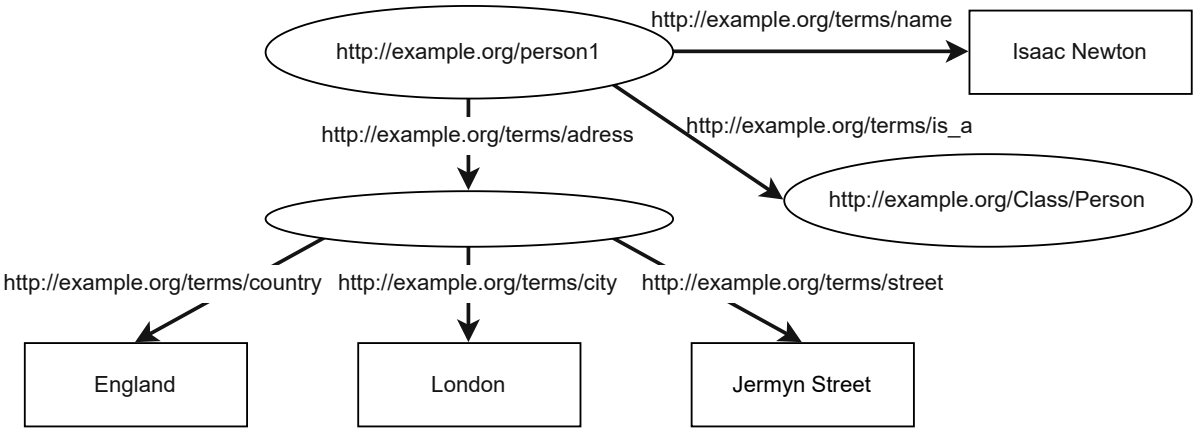


Figure 2.8.: RDF-Graph.

RDF Serialization

RDF is defined as an abstract syntax, i.e., it is independent of a particular concrete syntax. There are a number of different serialization formats [28]. Serializations of the same graph results in the same triples, therefor these different syntaxes are logically equivalent [28]. In this section, a short overview of the following formats is given. All examples use the same example as in Listing 2.1 to 2.3

1. The Turtle family of RDF languages (N-Triples, Turtle, TriG, and N-Quads)
2. JSON for Linking Data (JSON-LD)
3. RDF/XML

Turtle family of RDF languages: These formats are closely related. ***N-Triples*** provides a basic plain-text syntax to represent triples [28]. Each line represents a triple statement, whereby each IRI is enclosed by angle brackets (< >), and literals are set in quotes ("") and separated by whitespace [28]. Each line is terminated by a dot (.). Listing 2.4 shows an N-Triples example.

Based on the N-Triples syntax ***Turtle*** extends it with several shortcuts such as the definition of prefixes to shorten IRIs and a shorter way of expressing triples with the same subject [28]. The most common abbreviation besides the use of prefixes is the shortcut for the definition of *rdf:type*, which can be written as *a*. Listing 2.5 shows a Turtle example.

Turtle only allows specifying a single graph without naming it. ***TriG*** extends the turtle syntax with that feature to specify multiple graphs in the form of an RDF dataset [28].

N-Quads is the last concrete syntax of that family. It extends N-Triple to enable the exchange of RDF datasets by adding a fourth element to the triple, which defines the graph to which the statement belongs.

JSON-LD: JSON-LD, as the name suggest, is based on the JSON standard and allows systems which already utilize JSON to handle RDF data with minimal changes [29]. Just like other concrete syntaxes, JSON-LD offers universal identifiers to identify JSON objects as well as data typing, language referencing, and named graph support [28]. Listing 2.6 illustrates a JSON-LD serialization.

RDF/XML: RDF/XML was the first syntax developed for RDF. Although it uses two fundamentally different concepts, i.e., XML like tree structure and triple-based graphs, it is still widely used. Triples are specified within a XML element as shown in Listing 2.7.

Listing 2.4: N-Triples

```
<http://example.org/person1> <http://www.w3.org/1999/02/22-rdf-
   ↪ syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/person1> <http://xmlns.com/foaf/0.1/name> "
   ↪ Isaac Newton" .
<http://example.org/person1> <http://schema.org/jobTitle> "
   ↪ Physicist" .
```

Listing 2.5: Turtle

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix schema: <http://schema.org/> .

<http://example.org/person1>
    a foaf:Person;
    foaf:name "Isaac Newton";
    schema:jobTitle "Physicist".
```

Listing 2.6: JSON-LD

```
  "@context": {
  "schema": "http://schema.org/",
  "foaf": "http://xmlns.com/foaf/0.1/name"},

  "@id": "http://example.org/person1",
  "@type": "http://schema.org/Person",
  "foaf:name": "Isaac Newton",
  "schema:jobTitle": "Physicist"
  }
```

Listing 2.7: RDF/XML

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:foaf="http://xmlns.com/foaf/0.1/"
         xmlns:schema="http://schema.org/">

  <foaf:Person rdf:about="http://example.org/person1">
    <foaf:name>Isaac Newton</foaf:name>
    <schema:jobTitle>Physicist</schema:jobTitle>
  </foaf:Person>
</rdf:RDF>
```

To illustrate any RDF example and idea in this thesis, Turtle notation is used.

2.3.2. RDF-Schema

RDF Schema (RDFS) is a semantic extension of RDF that enables grouping related resources in classes and describing the nature of their relationships [30]. It is the most basic schema language in the semantic web and can be expressed in every concrete RDF syntax. It also provides the basics to infer additional information from instance data [1]. The following is a brief overview of the most notable characteristics. Note that the prefixes *rdf:* and *rdfs:* denote the corresponding namespace.

rdf:type states that a resource is an instance of a class [30]. Although defined in the RDF vocabulary, the semantics of *rdf:type* are only specified within the RDFS specification.

rdfs:Class refers to the class of all classes and is used to classify resources [30]. Explicit stating that a resource is a class is optional, since the statement that a resource is of *rdf:type* makes that type a class [15].

rdf:Property is the class of all properties [30]. It is used to link resources to one another, i.e., the predicate in an RDF triple [27]. Just like *rdfs:Class*, it doesn't have to be explicitly stated [1].

rdfs:subClassOf offers the ability to create a hierarchical structure of classes. Even though not explicitly stated in the W3C recommendation, the definition of a subclass enables the inheritance of properties with its domain and range from its superclass.

rdfs:domain and rdfs:range are both instances of the property class [30]. *rdfs:domain* denotes that a resource with a given property is an instance of a specific class. *rdfs:range*, on the other hand, states that the value of a property is an instance of a class.

Listings 2.8 to 2.9 show examples of RDFS-statements and the knowledge that can be inferred. Note that the prefixes have been omitted for better readability.

Listing 2.8: RDFS example

```
ex:Textbook rdf:type rdfs:Class;
    rdfs:subClassOf ex:Book.
ex:Book1 ex:title "Principia Mathematica";
    a ex:Textbook.

# infers

ex:Book rdf:type rdfs:Class.
ex:Book1 rdf:type Book.
```

Listing 2.9: Demonstration of Inference based on rdfs:domain:and rdfs:range

```
ex:writtenBy rdfs:domain ex:Author;
    ex:range ex:Book.
ex:Book1 ex:writtenBy ex:IsaacNewton.

# infers

ex:IsaacNewton a ex:Author.
ex:Book1 a ex:Book.
```

2.3.3. Web Ontology Language

RDFS allows to define simple ontologies and permits to infer new knowledge from that statements. However, it is impossible to model more complex statements [15]. For that purpose, the W3C introduced Web Ontology Language (OWL), and it became the de facto standard for modeling ontologies, and the current recommendation refers to its version 2 (OWL 2). OWL 2 is usually noted as RDF but there are other syntaxes, e.g. Manchester Syntax [31]. With this standard, there are two alternative ways of assigning meaning to ontologies. The direct model-theoretic semantics, called *OWL 2 DL* and RDF-based semantics, called *OWL 2 Full*. OWL2 DL assigns meaning based on the *SROIQ* description logic. OWL 2 Full is an extension of RDFS semantics and views ontologies as RDF graphs. In addition, OWL 2 DL defines three sublanguages, or profiles, (OWL 2 EL, OWL 2 QL, and OWL 2 RL) in order to suffice for a variety of applications. Each profile is more restrictive than OWL 2 DL and provides different expressive power to benefit the efficiency of reasoning [32]. Figure 2.9 compares these profiles with other ontology languages. In this thesis, the focus lays on the full expressiveness OWL2 Full. A small subset of the features provided by OWL 2 are summarized in the following.

owl:ObjectProperty is a subclass of *rdf:Property*. It is a so-called abstract role connecting individuals with individuals.

owl:DataProperty is a subclass of *rdf:Property*. It is a substantial role that connects individuals with data values.

owl:inverseOf states that a property is the inverse of another property. The inverse property connects the same two individuals but in the other direction.

owl:FunctionalProperty states that a resource can, at most, be one connection to another resource of a specific type. If two resources are named, this implies that these two resources refer to the same resource.

owl:TransitivProperty states that an object property is transitive, i.e., that each element in such a relationship is connected to elements further down that relation.

owl:sameAs notes that two resources refer to the same thing. This property can be used if different IRIs are used for the same thing.

owl:SymmetricProperty is a class where all its members are used to specify bidirectional relations.

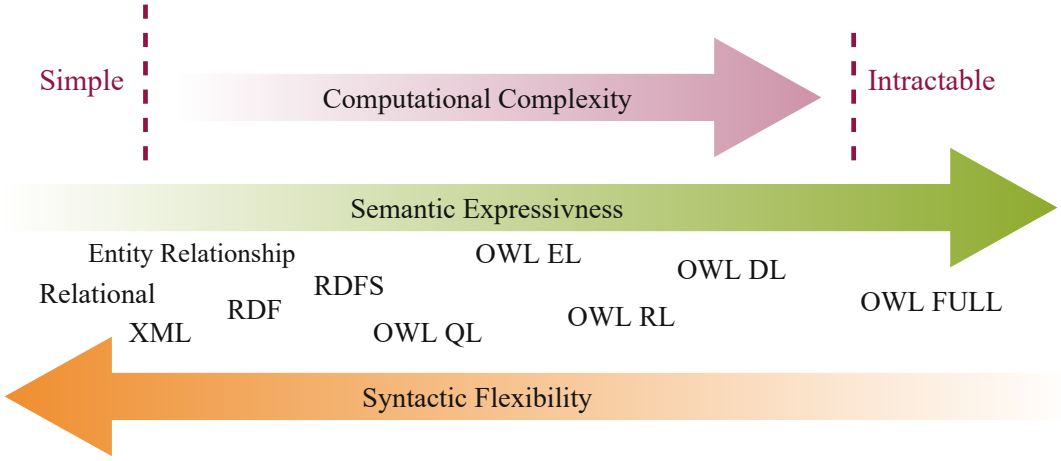


Figure 2.9.: Expressivity Characteristics of OWL Profiles. Source: [1, p. 14]

OWL is designed to support the Open-Word Assumption (OWA), allowing incomplete or uncertain information to be handled. The OWA means that the absence of information about an individual or a relationship does not necessarily imply that it does not exist. In contrast, relational databases operate under the Closed-World Assumption (CWA), where a missing value indicates falsehood [1]. For example, if we wanted to model the relationships between people in an OWA system, one can state that'*John and Mary are married*'. For all other individuals in our ontology, this information is unknown. In an CWA system, this information would lead to the conclusion that all other individuals are not married.

Another key characteristic of OWL is the No Unique Name Assumption (UNA). The assumption is that two entities with different IRIs are not to be unique. Generally, two entities are not two separate resources unless it is explicitly stated that the two resources

are different. As an example, if we wanted to model the relationships between people, one could say '*Bob* has friend *Mary*' and '*Bob* has friend *John*'. If we ask the question '*How many friends does Bob have*', the system can only say at least one since we did not explicitly state that *Mary* and *John* are different individuals.

2.4. Digital Twins and Modular Simulation

Since the first mention of the Digital Twin (DT) by Michael Grieves in 2002, [33] the DT gained traction. In its initial definition, the DT was such that it only mirrored a product. At the same time, the current state of the art also allows processes to be represented in the virtual space to gain the same benefits [34]. The DT is considered a key enabler for Industry 4.0 since it provides the means to connect the physical and virtual spaces. In the literature, there are a couple of different concepts and solutions of a DT, due to this Kritzinger *et al.* [34] categorized different concepts into three categories based on their level of integration. Each category builds upon the previous and can be viewed as part of the development process of a DT. Besides creating a virtual representation, the definitions of interfaces for data exchange and other services are also part of developing a DT.

Digital Model has no automated data exchange between virtual and physical space, as seen in Figure 2.10. It is, therefore, a virtual copy of a planned or existing entity. This could be a simulation or mathematical model of a planned entity or any other physical object without automated data exchange.

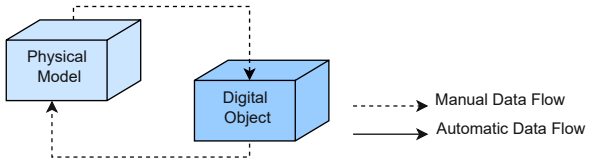


Figure 2.10.: Digital Model. Source: [34, p. 1017]

Digital Shadow is a virtual representation of a physical entity with an automated way to exchange data. However, this data flow is limited to one way, i.e., a state change in the physical world leads to a change in the virtual world but not vice versa. Figure 2.11 shows this one-way data flow.

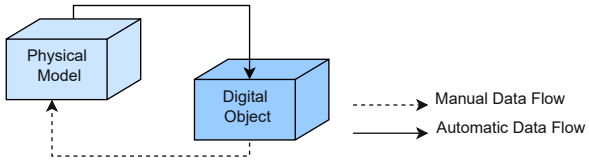


Figure 2.11.: Digital Shadow. Source: [34, p. 1017]

Digital Twin extends the connection of the Digital Shadow further. Here the automated data exchange can be both ways. A change in state in either entity can change the state of the other and therefore affect each other. This automated exchange of data can be seen in Figure 2.12.

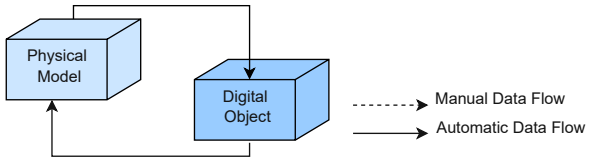


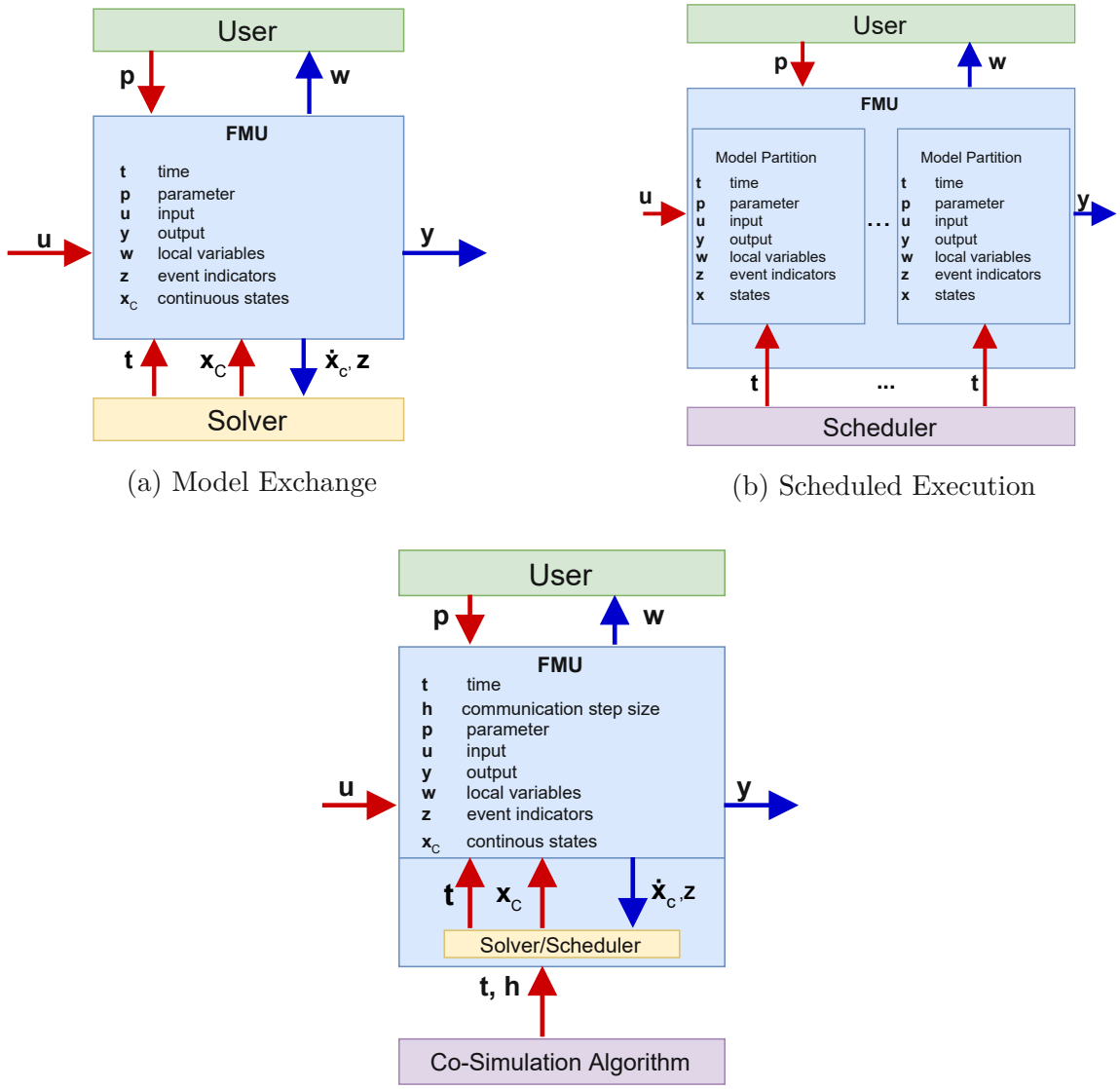
Figure 2.12.: Digital Twin. Source: [34, p. 1017]

Modern systems are often composed of individual components. As each component can be modified individually, the simulation model can be complex [35]. Moreover, such systems seldom cover only one domain. These problems make it impossible to cover a full-system model and require individual models from different modeling tools. Thus resulting in a modular multi-domain DT. The Functional Mockup Interface (FMI) standard provides a standardized interface for such modular models to develop complex systems. Full system models can be assembled from individual Functional Mockup Units (FMUs). This FMU is a ZIP archive containing binaries and/or the source code as a set of *C-functions* implementing the Application Programming Interface (API) functions, an XML file containing the definition of the variables, model structure as well as additional data such as documentation, maps, and tables used by the model [36], [37]. FMI defines three interface types:

Model Exchange is intended to share a simulation model to be utilized by other modeling and simulation environments. With this method, the importer has to handle advancing time, setting states, handling events, etc.

Co-Simulation is used to link two or more simulation tools as well as the coupling of subsystem models. This method includes the export of the solvers themselves. Master algorithms ensure the data exchange between subsystems, but this exchange is limited to discrete communication points. During those discrete intervals, the models are solved independently from each other by their respective solvers.

Scheduled Execution allows for concurring computation of model partitions on a single computational resource. The scheduler has to be provided by the importer and is responsible for advancing the overall simulation time, activating time-based and triggered clocks for all exposed model partitions.



(a) Model Exchange

(b) Scheduled Execution

(c) Co-Simulation.

Figure 2.13.: Schematic view of data flow for each interface.
Source: Adapted [38, p. 17-18]

2.5. Related work

This section outlines the current state of the art of different (semantic) data models to describe the railway domain, focusing on different aspects of the domain and motivated by different use cases and the application of semantic web technologies for multi-domain systems and (distributed) Digital Twins.

2.5.1. Railway Domain

Data interoperability in the railway domain presents some difficulties as it typically involves many subsystems. To overcome these interoperability problems, both company-specific and industrial consortium multi-domain data models were proposed. Such models include the Unified Modeling Language (UML) based *RailTopoModel*[1]. A data model developed by the industrial consortium and standardized as IRS30100 by the International Union of Railways (UIC). The model provides a way of describing the topological aspects of the railway domain and is currently implemented in *RailML*[2] and *EULYNX*[3]. However, according to Rojas *et al.* [39], most of these models lack semantic definitions and therefore hinder the data exchange across organizations.

The usage of semantic technologies in the railway domain was already established in the past. *InteGRail* was one of the first European Union (EU) project utilizing an ontology to integrate railway subsystems. The resulting *Railway Domain Ontology* was used to build a Proof-of-Concept platform to augment the data with context so that a structured and meaningful exchange of information between railway stakeholders is supported [40]. Another EU project that utilized semantic technologies is *Smart Rail*[4], a lighthouse project for the EU *Shift2Rail*[5] initiative to seek focused research in the railway domain. Smart Rail focused on modeling stakeholders and physical resources of the railway infrastructure. For this purpose, an ontology[6] was developed.

Semantic Transformations for Rail Transportation (ST4RT)[7] is another project under the *Shift2Rail* initiative. The project aimed to provide an ontology-based demonstrator tool that can enable the transformation between different standards and protocols, resulting in enhanced semantic interoperability between disparate, heterogeneous legacy systems. The Rail Core Ontology (RaCoOn) was initially developed for the representation of signaling and rail infrastructure, and the ontology transformed into a more general model covering more generic railway concepts [41]. Rail Core Ontology (RaCoOn) has a layered approach extending the core model with subdomains, i.e., timetabling or rolling

[1] https://www.railtopomodel.org/homepage.html
[2] https://www.railml.org/en
[3] https://eulynx.eu/
[4] https://smartrail-project.eu
[5] https://rail-research.europa.eu
[6] https://ontology.tno.nl/smart-rail
[7] http://www.st4rt.eu

stock, as well as an upper-level model to define concepts more broadly than just the railway domain, e.g., transport. Bischof and Schenner [42] highlight the need for an open standard ontology for railway topologies based on existing standards such as the aforementioned RailTopoModel. The authors propose the Rail Topology Ontology[8] as a way of integrating disconnected data sources for railway topologies [43]. One of the most recent projects applying semantic technology was carried out by Rojas *et al.* [39] in cooperation with European Union Agency for Railways (ERA). ERA is a European authority assigned to provide the legal and technical framework to integrate European railway systems by making trains safer and able to cross national borders without stopping. Rojas *et al.* [39] presented an ontology[9] for the railway infrastructure and authorized vehicle types, a reusable Knowledge Graph describing the European railway infrastructure. This cost-efficient system architecture enables high flexibility for use case development and an RDF Web application to support route compatibility checks[10].

All of these approaches remained academic exercises, with the exception of the ERA project. The ontologies that were created have become unmaintained or are no longer available.

2.5.2. Multi Domain Simulation and (distributed) Digital Twins

As mentioned in Section 2.4, a DT can be composed of multiple different DT and, therefore, multiple simulations. This can often result in distributed and heterogeneous data sources and different stakeholders. According to Moshrefzadeh *et al.* [44], such systems can be called distributed DT. The authors developed a concept focused on the data integration process by providing a data catalog as a registry for resources of all types and their stakeholders. This concept is based on the Data Catalog Vocabulary (DCAT) version 2 standard. An *InformationResource* is defined as a general class for all types of resources and holds information about that resource. The *Distribution* class is used to control the accessibility of a resource and specific representation of a dataset. Links between different Resources are realized with the class *Relationship*, which specifies *InternalLink* and *ExternalLink* depending on whether the resource is registered in the catalog or not. Stakeholders are managed with the *User* and the *Organization* classes.

Kasper *et al.* [45] proposed a DT platform for the industrial energy system domain. A five-dimensional DT modeling approach is used for the platform. For each dimension, physical space, virtual entity, service dimension, data dimension, and connection, the implementation issues are addressed, and a universal solution is proposed. Focusing on the virtual entity dimension, the authors defined the FMI standard as an interface for accessing the metadata of model instances. Including a metadata file, which contains

[8] `https://w3id.org/rail/topo`
[9] `https://data-interop.era.europa.eu/era-vocabulary`
[10] `https://data-interop.era.europa.eu/route-compatibility`

descriptions of the connection types, alongside the FMU data, provide enough information to describe the virtual entity. In the data dimension, the added semantic data is used to build a so-called Knowledge Graph (KG), a knowledge-based system consisting of an ontology and a built-in reasoner capable of acquiring and integrating external information sources. The KG holds several ontologies that describe different parts of the DT platform, i.e., plant equipment, topology, and instrumentation. A mapping between the relational database structure allows for ontology-based data access, where the data can be stored in the original local database while utilizing the ontology's capability to make the implicit knowledge explicit. Rather than develop a new ontology, existing ontologies are integrated into the platform, including the Sensor, Observation, Sample, and Actuator (SoSa)[11] ontology or the Ontology for Computer-aided Process Engineering (OntoCAPE)[12]. To store the semantic data, i.e., the instances created with the ontologies, the RDF framework RDF4J[13] is used. The Java framework provides an easy way of storing, querying, and reasoning with RDF data.

Nordahl *et al.* [46] presented a method to automatically connect variables of different simulation models using an ontology-based approach. The used marine systems model interface (MSMI) ontology[14], a part of the Open Simulation Platform (OSP) Interface Specification, captures all necessary information surrounding connections between simulation models as can be seen in Figure 2.14. This information can be used to infer valid connections between simulation models.

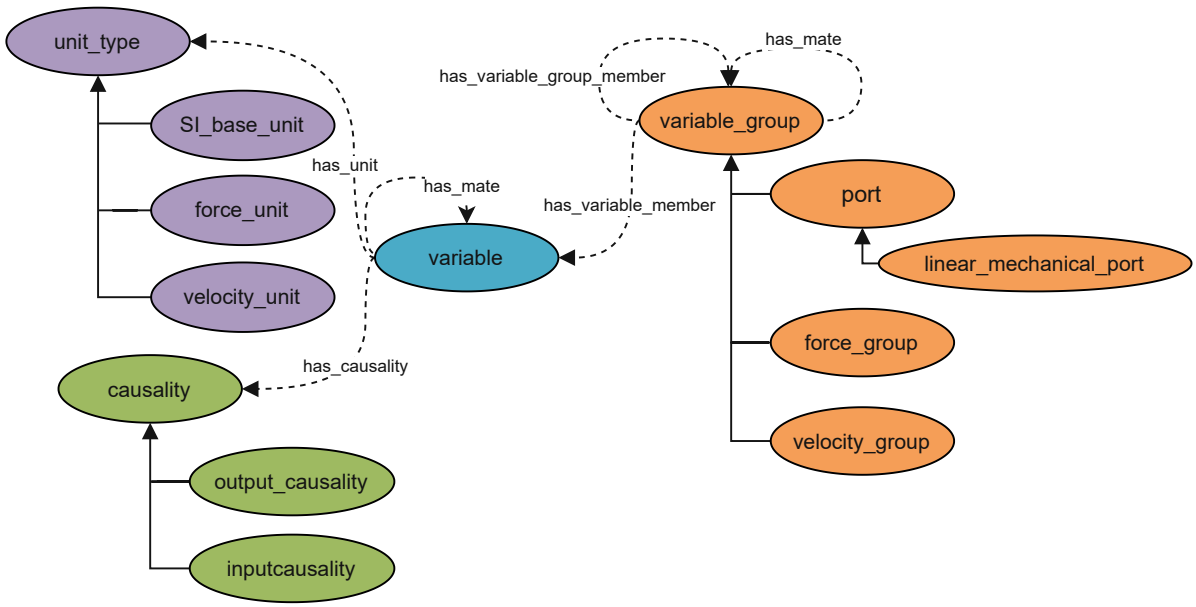


Figure 2.14.: Central concept of the MSMI ontology. Source: [46]

[11] https://www.w3.org/TR/vocab-ssn/
[12] https://www.avt.rwth-aachen.de/cms/AVT/Forschung/Sonstiges/Software/~ipts/OntoCape/
[13] https://rdf4j.org/
[14] https://data.dnv.com/osp/

The ontology defines a *variable* concept, which contains a unit, causality, and data type. This variable can be linked together by the property *has_mate*. This connection is only valid if the two variables have opposite causality, identical data types, and compatible units. Multiple variables logically belonging together can be grouped in a *variable_group*, whereas a *variable group* can also contain other variable groups. These groups have a *type* that denote their semantic meaning. *Ports*, a specific type of a *variable_group*, are used to model two *variable_groups* with opposing causality. This concept enables the authors to model bonds from bond graph theory. An important part of the ontology is the unit definition since it allows for validation and restrictions of connection of two different variables. Nordahl *et al.* [46] used the Units of Measure ontology[15] and the concepts from the FMI standard to detect if the units match or can be converted to the same base unit. With the explained ontology and a suitable reasoning system, the authors were able to obtain answers to different types of queries, such as *Given variable group A and B, are they compatible?* or *List all variable groups of type C.* In addition, they showed the potential of a graphical modeling tool to automate the validation process of an already configured simulation or identify valid connections.

Mitterhofer *et al.* [47], [48] describes the structure and usage of the *FMUont* an ontology for annotating FMUs and automatically infer the simulation topology of an arbitrary number of contributing simulation modules. Figure 2.15 shows the structure of the *FMUont*.

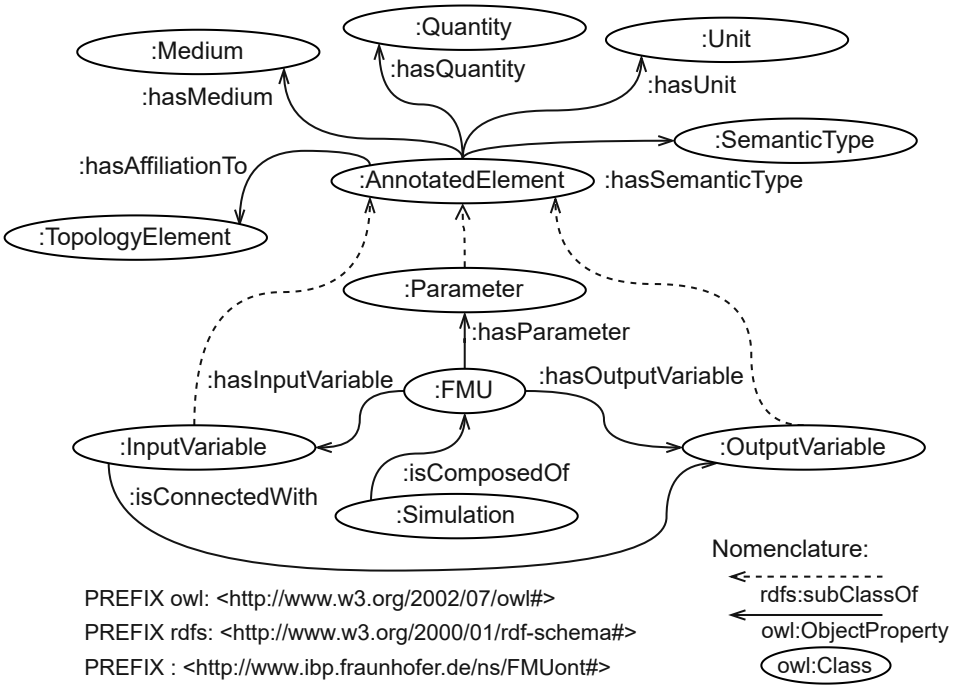


Figure 2.15.: Structure of FMUont. Source: [48]

The ontology's core concept is the *FMU* class, representing a single FMU simulation

[15] `http://www.ontology-of-units-of-measure.org/`

environment. Multiple FMUs are combined to a *Simulation*. *InputVariable*, *OutputVariable* and *Parameter* are related to their associated FMU instance via the corresponding object property. *InputVariable* and *OutputVariable* are further connected to each other via the *isConnectedWith* property and are subclasses of the *AnnotatedElement* class. To provide more detailed information about the variables and parameters, the *Medium*, *Unit*, and *SemanticType* classes are added. For the *Unit* and *Quantity* description, the authors reuse the ontology of units of measure[16]. *SemanticType* are required to give a precise meaning to input and output data points and include WeatherData, Heating, or Ventilation. In addition to that description, a *hasAffiliationTo* property links variables to their corresponding object in a system-level ontology. In the discussed work, the authors are building performance simulation-specific ontology. Furthermore, a Graphical User Interface (GUI) is presented to provide information about the input and output variables and guides the planner through the annotation process. The resulting annotation file is stored as an *.owl* file alongside the original FMU files. To build a modular simulation environment, the *.owl* file is extracted and merged to infer matches based on the required in- and output variables.

Wiens *et al.* [35] discussed the strategy of building digital twins from individual FMUs with predefined model interfaces based on an ontology for renewable energy systems. The authors put forward a framework to enable the automation of the model assembly process. The ontology, as a core part of that framework, defines adapters that map the in- and outputs of the simulation model to a predefined connector structure where the connectors specify which models or components are compatible with each other. A controller contains a list of signals including measurements and directions. Compatible connectors must have matching units and opposing directions. An example of the system ontology and interaction of components can be seen in Figure 2.16, and an example of a connector instance can be seen in Figure 2.17.

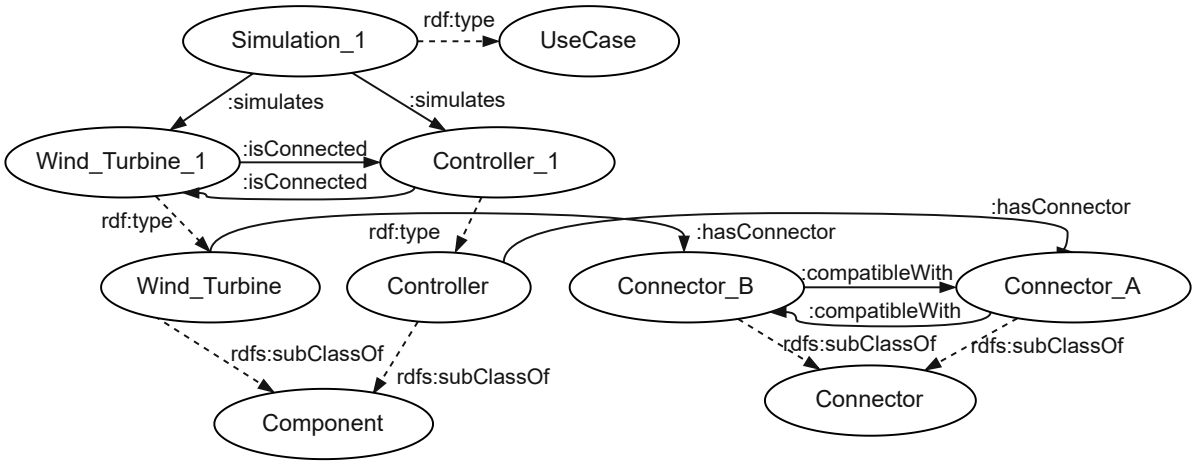


Figure 2.16.: Example for the system ontology and interaction of components.

[16] http://www.ontology-of-units-of-measure.org/

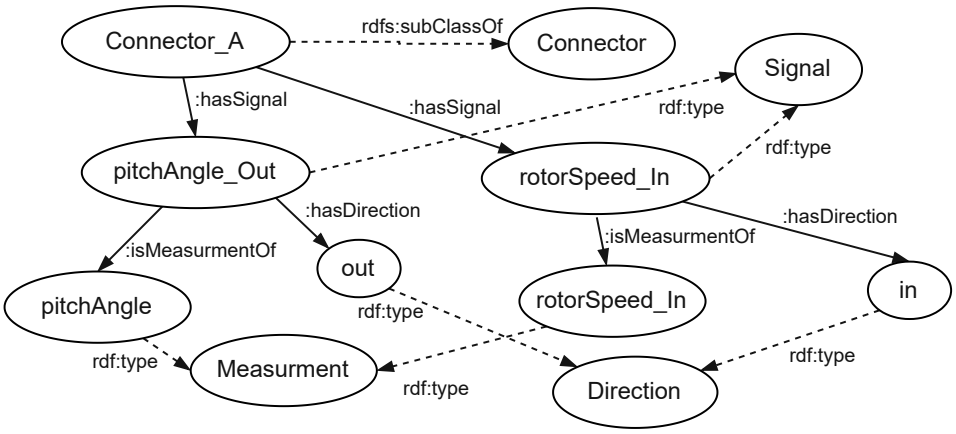


Figure 2.17.: Example Connector Instance

The connections between individual models are specified by the user on the top level and details are handled by the framework. A general description of the process for the development of model-based digital twins is given. This includes creating a new simulation model, defining required adapter connection models in the specific modeling tool based on an ontology, connecting the model to the corresponding adapter and exporting the model to a FMU, and adding it to the database. To build complex systems, the necessary components are collected and connected. The communication adapters are exported to an System Structure and Parameterization (SSP) package. This package can be handled by an orchestrator to run the simulation.

3. Implementation

3.1. Ontology Development

This section presents the methodology used in designing and implementing the ontology. None of the earlier work, given in section 2.5 satisfied the requirements, or the presented ontology is not publicly available. However, ideas were taken from the mentioned literature. The design approach adopted in the Rail4Future (R4F) methodology draws heavily upon some of the modular ontology engineering tasks described in the NeOn methodology [13]. Figure 3.1 gives an overview of the different scenarios and included steps in the NeOn methodology. The used approach utilized methods from scenario 1 (From specification to implementation), scenario 2 (Reusing and re-engineering non-ontological resources), and scenario 3 (Reusing ontological resources). The following subsections provide a detailed description of the steps taken. Note that all phases include an evaluation process. This activity is not discussed during the individual phases.

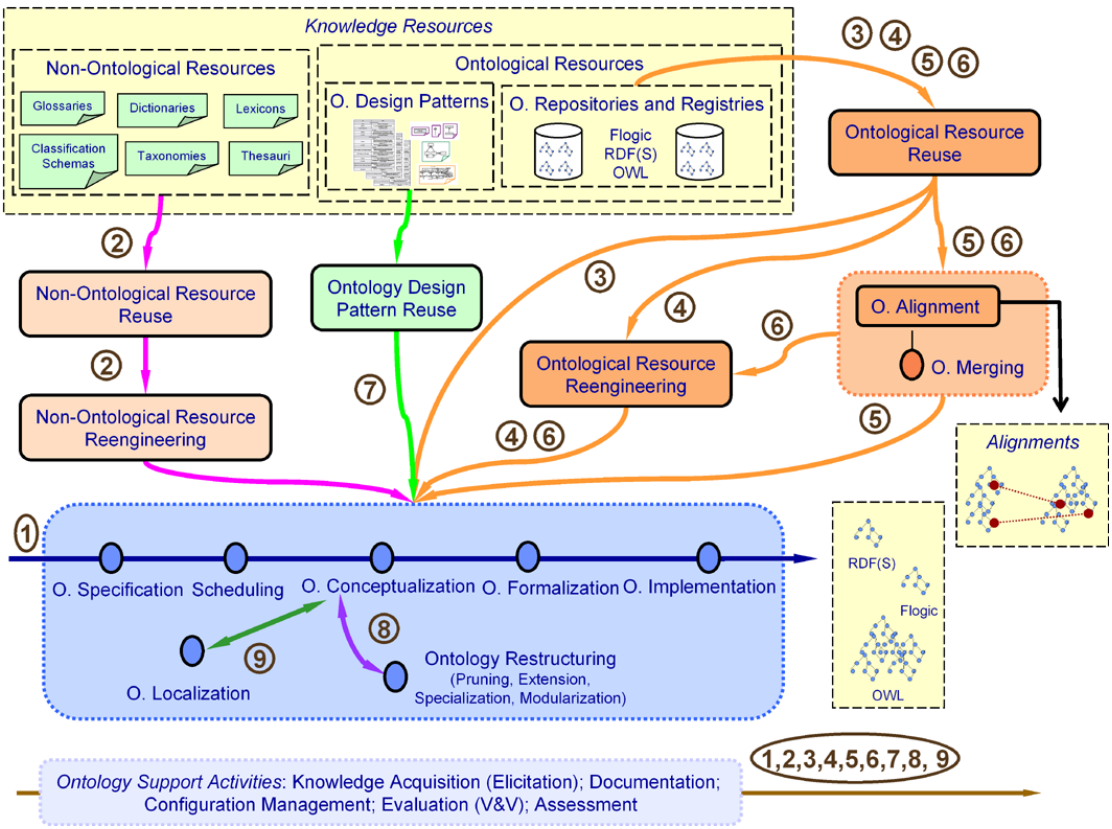


Figure 3.1.: Ontology development processes in the NeOn methodology. Source: [13]

3.1.1. Initiation Phase

The first step in the ontology development process is to identify its requirements. The so-called Ontology Requirements Specification Document (ORSD) supports the development of ontology in various ways, including:

1. Help define the knowledge that should be represented in the ontology
2. Assist in reusing knowledge resources by directing the search toward the specific knowledge resources needed for the ontology
3. Enabling verification of the ontology to ensure it meets the necessary requirements

Figure 3.2 shows the task defined by the NeOn methodology for the ontology requirements specification process. The functional requirements are formulated as competency questions. Competency questions define and limit the scope of knowledge captured in the ontology and are phrased in natural language. From these questions and their respective answers, the main concepts and the relation between those concepts, i.e., properties, can be extracted. They also play a role in the evaluation of the ontology as they form a requirement specification against which the ontology can be evaluated. Tasks 1 to task 4 are described below. The resulting ORSD can be found in full in Appendix B.1.

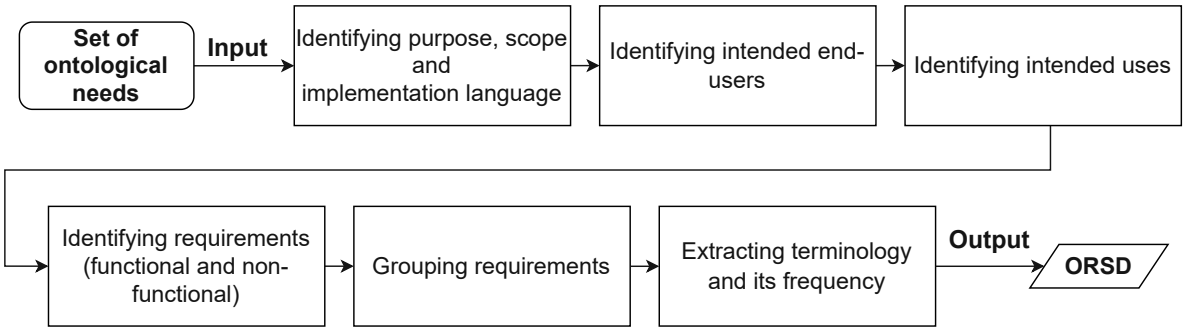


Figure 3.2.: Ontology Requirements Specification Process. Source: Adapted from[49]

Identifying the purpose, scope, and implementation language: The purpose of the R4F Ontology is to provide a consensual knowledge model of the R4F domain to be used by the operators of the R4F digital twin platform. The ontology not only has to focus on describing simulation models and their respective input and output variables but also provide additional information about relevant metadata about those variables and simulation model for easier data exchange and data integration. The ontology must be implemented using the OWL language to utilize the first-order logic based reasoning capability.

Identifying the intended end-users: The users of the ontology and the applications based on it are the following:

User 1. Developer of the digital twin platform with limited knowledge about ontologies

User 2. Ontology engineers for maintenance and enhancement of the ontology network

User 3. Domain experts who submit simulations to the digital twin platform with limited knowledge about ontologies

User 4. User of digital twin platform with no ontological knowledge

Identifying the intended uses:

Use 1. Publish information about a specific asset or asset group

Use 2. Searching for information about a specific asset or asset group

Use 3. Assist the automated generation of the simulation topology

Use 4. Automatically verify a given simulation setup

Identifying non-functional requirements: Non-functional requirements are general requirements or aspects the ontology should fulfill. The following non-functional requirements are defined for the R4F ontology.

NFR 1. The ontology must support a multilingual scenario in the following languages: English, German

NFR 2. The ontology must be based on standards used in the R4F project

Identifying requirements: Most of the competency questions defined in the R4F are related to the simulation model and the respective input and output variables which are the most relevant concept in the presented work. The complete list of the competency questions is included in Appendix B.1. Some examples are:

QC1. What is the name of the simulation model?

QC3. What versions exist of the simulation model?

QC7. What aspect of the R4F domain does the simulation model?

QC8. To which model does a variable belong?

QC9. What is the causality of a variable?

QC11. Do variable A and variable B have the same base unit?

From the competency question, the terminology is extracted. This is represented by the ontology by means of concepts, attributes, and relations. A list of identified terms can be found at the bottom of the ORSD in Appendix B.1.

3.1.2. Reuse and Re-engineering Phase

The process of reusing and re-engineering phase can benefit the ontology development process. Mainly because it can increase the quality of new ontologies by using already tested components. Mapping between two ontologies that use the same concepts becomes easier. This phase can be split into two parts, the reusing phase and the re-engineering phase. Figure 3.3 shows the reuse phase in detail. When reusing existing components, one can divide those into three main categories: General ontologies, domain ontologies, and non ontological resources (NOR), i.e., classification schemes, thesauri, or glossaries.

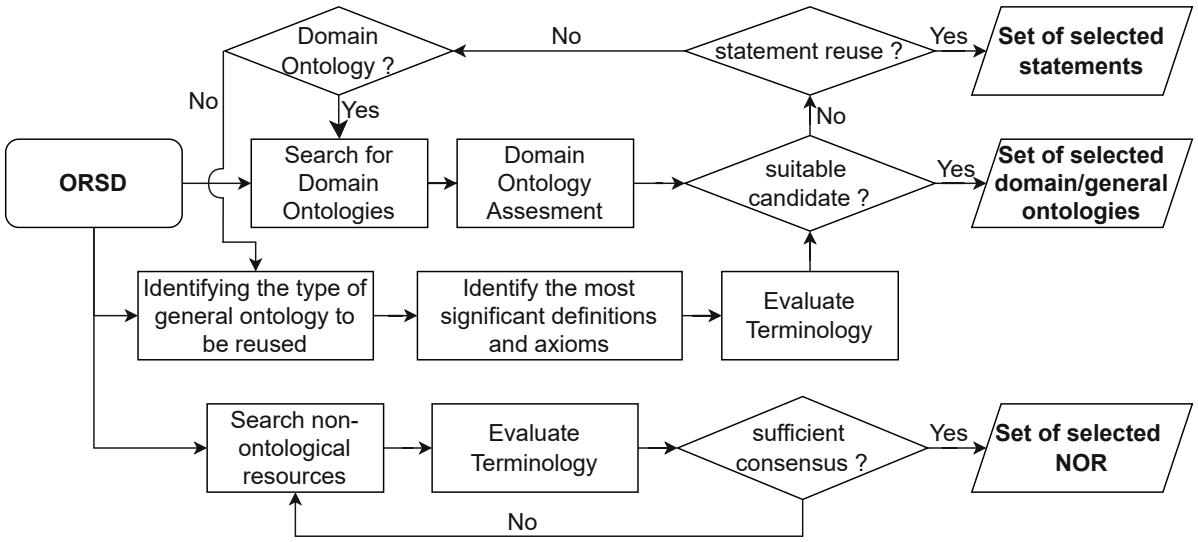


Figure 3.3.: Ontology Reuse Process.

General ontologies cover a broad range of concepts and entities not constrained to particular domains and can therefore be reused in different domain ontologies. Such ontologies cover concepts like time, space, or events. Domain ontologies, on the other hand, focus on a specific subject area. When reusing general or domain ontologies, it is sometimes not practical to reuse the ontology as a whole, since it may contain a large amount of knowledge that may not be needed when developing a particular ontology. In this case, only some knowledge, e.g., statements, can be reused. Non-ontological resources are highly heterogeneous in their data model and contents; nevertheless, they present a good knowledge base to integrate into an ontology as they are usually related to significant concepts in a specific domain. As they have not yet been formalized by an ontology, they need to be re-engineered. The NeOn methodology offers a design patterns library[1] to re-engineering non ontological resources into ontologies. The library includes 12 patterns that can be used to transform the most common non-ontological resources. Figure 3.4 shows the process of re-engineering adapted for this work. Based

[1] http//ontologydesignpatterns.org

on the ORSD from the previous stage, the reuse and re-engineering process for each of the mentioned resources is discussed below. Note that the re-engineering process for ontological resources in the NeOn methodology is vague, and no clear steps are suggested. In this work, most of the re-engineering of ontological resources is done by adapting the definition of the terms to the current domain. This does not pose a problem; however, this could prove different for other more complex ontology development processes.

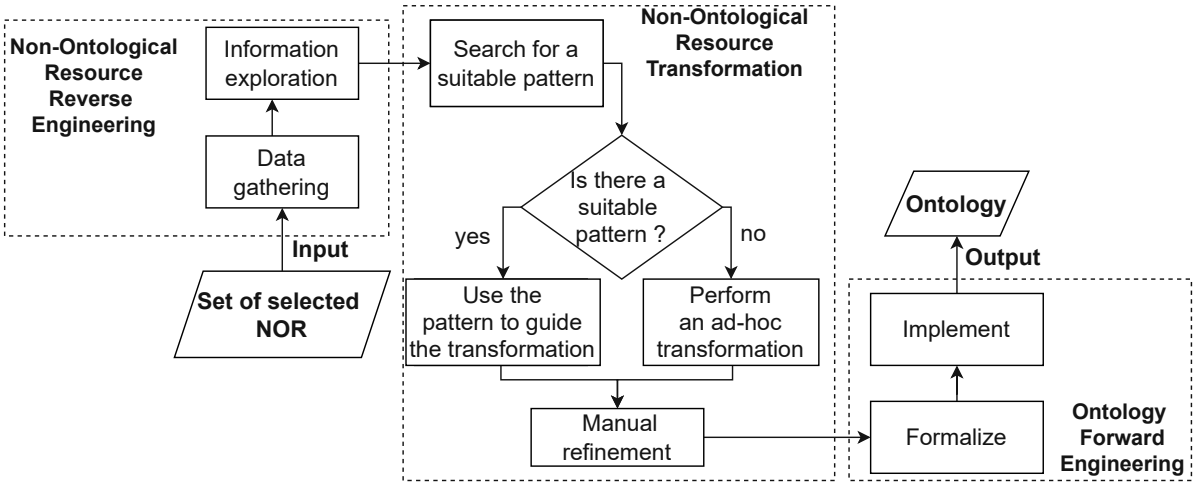


Figure 3.4.: Non-ontological resource re-engineering process. Source: Adapted from [50]

In the following, the selected general and domain ontologies, the selected non ontological resources, as well as the ontologies from where only statements have been used, are presented. The specifically chosen statements are discussed in the Design Phase (3.1.3).

General Ontologies: Based on the identified terms in the ORSD, an assessment of general ontologies is undertaken. Terms such as *Author* or *Publisher* suggest that some kind of agents are responsible for a simulation model. One possible solution is the *Friend of a Friend (FOAF)* ontology[2]. This small ontology describes people and social relationships on the web. *FOAF* is descriptive vocabulary expressed in RDF and OWL. Since *FOAF* is not only able to describe people, but also organizations, it is used in the R4F ontology to identify and provide information about relevant agents. In addition, it is used to provide metadata for documents such as documentation of simulation models since *FOAF* does not distinguish between physical and electronic documents. This additional information would require an additional self-defined subclass. Another term that would need to be self-defined, or use existing ontologies, such as the organization ontology[3], is a specific type of relationship between agents, such as affiliation, sub-organization, or similar. As this is not the aim of the R4F ontology, this is not implemented.

[2] http://xmlns.com/foaf/0.1/
[3] https://www.w3.org/TR/vocab-org/

Since the *FOAF* ontology also defines classes to model online accounts and personal information, only some terms are reused and implemented in the R4F ontology.

Another general ontology that is reused by the R4F ontology is the Dublin Core (DC) ontology[4]. DC is used to describe generic metadata and is divided into two vocabularies: *DC elements* and *DC terms*. The *DC elements* defines 15 terms published as a standard, e.g., ISO 15836. The *DC terms* not only extends these 15 terms with seven more, but also defines terms with the same names under its own namespace. The main difference between those two vocabularies is that the *Element* set namespace doesn't have a range, and the 15 properties from the *Terms* namespace are sub-properties of the 15 same-named terms from the *Element* set. The DC recommends using the more precise *Terms* namespace, henceforth called *DCTERM*. Although *DCTERM* also defines a *Agent* class, the R4F ontology still uses the aforementioned *FOAF* ontology since *DCTERM* does not provide specific terms for describing agents. Furthermore, the *FOAF* namespace declares the *DCTERMS Agent* as an equivalent class. The *DCTERMS* ontology is weakly constrained as it is not OWL2 DL ontology, which means it is particularly suitable to reuse and redefine the terms.

As the *FOAF* ontology focuses more on describing people, agents, and social web entities. The *DCTERMS* focuses on generic metadata, the *vCard*[5] ontology can be used to model contact information for people and organizations, i.e., addresses, email addresses, and telephone numbers. There are some overlaps between these ontologies, but each focuses on a different part of relationship modeling. *vCard* is originally a specification developed by the Internet Engineering Task Force (IEFT) and has been mapped to create the OWL ontology. From the *vCard* ontology, only statements are reused.

From the competency questions, one can see that some describe concepts related to Measurements. For this, a unit ontology can be reused. There are some unit ontologies available such as the Quantity, Unit, Dimension and Type (QUDT)[6] ontology, the Quantities, Units, Dimensions, Values (QUDV)[7] ontology, the Units Ontology (UO)-2[8] or the Ontology of units of Measure (OM)[9]. In order to reuse one, the ontologies mentioned are compared and assessed if they satisfy the needs in the R4F domain. Table 3.1 shows the assessed ontologies and the criteria used. The assessment is based on the studies carried out from [51], [52], [53], and the ontologies documentations.

[4] https://www.dublincore.org/specifications/dublin-core/dcmi-terms/
[5] https://www.w3.org/TR/2014/NOTE-vcard-rdf-20140522/
[6] https://www.qudt.org/
[7] https://www.omgwiki.org/OMGSysML/doku.php?id=sysml-qudv:quantities_units_dimensions_values_qudv
[8] https://www.ebi.ac.uk/ols/ontologies/uo
[9] http://www.ontology-of-units-of-measure.org/

	QUDT	OM-2	QUDV	UO
Unit Implementation	Individuals	Individuals	Individuals	Classes
Number of Units	>1500	>1300	<20	>200
Quantity Implementation	Individuals	Classes	Individuals	Ambiguous
Number of Quantities	>900	>500	<10	>324
Implementation Language	OWL	OWL	OWL	OWL
Unit Conversion	Yes	Yes	Yes	No
Application Area	Science, biology, physics, engineering	Science, engineering, materials	Physics, agriculture	Biology, biomedicine
Last Updated	March 2023	March 2023	October 2009	October 2022

Table 3.1.: Assessed Unit Ontologies

After the assessment, the QUDT ontology was chosen as the ontology to reuse for the R4F ontology. This is because of its good documentation, ongoing development, and support. The following paragraph briefly summarizes and explains the QUDT ontology.

The QUDT ontology is a collection of ontologies used to model physical quantities, units of measure, and their dimensions in various measurement systems. The *QUD* ontology was initially developed for the NASA Exploration Initiatives Ontology Models (NExIOM) project and is now maintained by the QUDT.org organization. The core design pattern of the QUDT ontology is shown in Figure 3.5. Every unit has an associated *QuantityKind* and *QuantityKindDimensionVectors*. A *QuantityKind* represents all kinds of things that could be measured, e.g., length, area, torque, or velocity. Each unit is associated with at least one *QuantityKind*. For example for a pascal (symbol: Pa) *QUDT* defines, among others, the *QuantityKind*: *ForcePerArea*, *ShearModulus* and *ModulusOfElasticity*. The *QuantityKindDimensionVectors* vocabulary can be used to determine whether one unit can be converted into another or perform a dimensional analysis of equations. The *QUDT* ontology is used to help describe the variables of a simulation model and can be used to verify connections of different simulation models.

The Software Package Data Exchange (SPDX)[10] ontology is an open standard used for communicating software bill of material information. This includes li-

[10] `https://spdx.org/rdf/terms/`

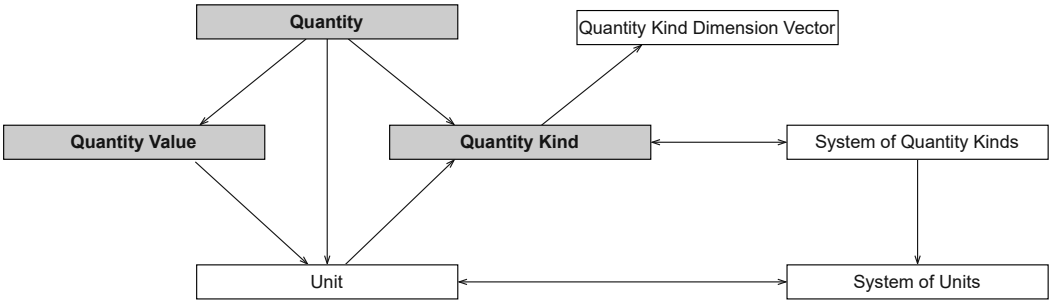


Figure 3.5.: Core Design Pattern of the QUDT Ontology. Source: [54]

censes, copyrights, and security references. In the R4F ontology, SPDX is used for additional information about the simulation model and supporting documents and distributions. Such information includes any license info, checksum, and associated checksum algorithm.

Domain Ontologies: Derived from the ORSD, the main domain is related to the simulation model and the description of the variables. As already outlined in the Related work section, there are not many domain ontologies covering that domain. Nordahl *et al.* [46], Wiens *et al.* [35], and Mitterhofer *et al.* [48] present their respective solution to model the collection between the simulation model and the corresponding variables. However, none of the mentioned ontologies are publicly published, and only part of the ontology is visualized in the referenced papers. Nevertheless, some ideas have been taken from those papers, mainly from Mitterhofer *et al.* [48] where the *SemanticType* class, which is used to give a precise meaning to input and output variables, can be used to assist the automated generation of the simulation topology and verify the manual model configuration. The *VariableGroup* concept is taken from [46] and adapted to the needs of the R4F domain. Since those terms lack a formal definition, they are re-engineered and implemented.

To model the distributed data sources and utilize some of the mentioned general ontologies, the Data Catalog Vocabulary (DCAT)[11] is used. The DCAT vocabulary is intended to enhance the compatibility of data catalogs available on the Web and enables publishers to describe datasets and data services. For the application in the R4F domain, some terms for the vocabulary are re-engineered because the initial definition is too specific to a traditional data catalog. However, this would still allow us to recreate a data catalog in the future if needed. A data catalog could enhance data discovery without centralizing the data itself.

Non ontological resources: As a non ontological resource the FMI specification[12] version 2.0.4 was chosen. This is due to using the standard in the R4F project to build co-simulations. The reason why version 2 instead of the current version 3 is used

[11] https://www.w3.org/TR/vocab-dcat-3/

[12] https://github.com/modelica/fmi-standard/releases/download/v2.0.4/FMI-Specification-2.0.4.pdf/

is simply due to the fact that version 3 was only available as a beta version when the project was started. However, since the terminology of version 3 hasn't only changed slightly compared to version 2, the ontology itself is version independent. The terms specified in the standard match those of the ORSD. The re-engineering process of those terms is discussed below.

Table 3.2 summarizes the used ontological and non ontological resources and provides links to their respective specification.

Resource	**URL to specification**
dcterms	`https://www.dublincore.org/specifications/dublin-core/dcmi-terms/`
foaf	`http://xmlns.com/foaf/0.1/`
qudt	`https://www.qudt.org/`
spdx	`https://spdx.org/rdf/terms/`
vCard	`https://www.w3.org/TR/2014/NOTE-vcard-rdf-20140522/`
dcat	`https://www.w3.org/TR/vocab-dcat-3/`
FMI 2.0.4	`https://github.com/modelica/fmi-standard/releases/download/v2.0.4/FMI-Specification-2.0.4.pdf/`

Table 3.2.: Used Ontology Summary

As already mentioned above, NeOn is vague when it comes to re-engineering ontological resources. One defined process is the modularization of ontologies. The main task of modularizing an ontology is to identify components of an ontology that can be used independently from the others in that ontology. Hence "cut-down" large ontologies into smaller, more manageable modules. This could benefit performance and facilitate the development and maintenance of the ontology. NeOn recommends this activity as part of scenario 3 (Reusing ontological resources), scenario 4 (Reusing and re-engineering ontological resources.), and scenario 8 (Restructuring ontological resources). In this work, the re-engineering of ontological resources is limited to adapting the selected terms into the R4F namespace and modifying the terms' definitions to fit the intended use.

Figure 3.4 shows the re-engineering process for non-ontological resources. This activity is split into three main parts, non-ontological resource reverse engineering, non-ontological resource transformation, and ontology forward engineering. The first part focuses on analyzing the non-ontological resources to detect the relevant components and identify the resource's underlying schema and data model. For the non-ontological resource transformation part, a conceptual model of the resource has to be developed. To this end, NeOn recommends using Ontology Design Pattern (ODP) to improve the efficiency of the re-engineering process and make the transformation easier. NeOn provides an ontology design patterns library[13] that includes 12 patterns used during the re-engineering process. These patterns include:

[13] `http//ontologydesignpatterns.org`

- Re-engineering a classification scheme, which follows the adjacency list model, to design an ontology schema
- Re-engineering a classification scheme, which follows the flattened model, to design an ontology schema
- Ontological representation of a specific domain concept conceptualized using a Faceted Classification Scheme (FCS).
- Re-engineering a term-based thesaurus that follows the record-based model to design an ontology schema
- Re-engineering a term-based thesaurus, which follows the relation-based model, to design an ontology schema

If a suitable pattern is found, the transformation can be conducted according to the re-engineering procedure established in the pattern. The proposed patterns help with the TBox as well as the ABox transformation. In short, the TBox (Terminology Box) contains terms used to describe the concepts and their relationships in an ontology. The ABox (Assertion Box) contains assertions or statements about individuals in a specific domain. The Tbox Transformation transforms the resource content into an ontology schema, while the ABox transforms the resource schema into an ontology schema and the resource content into ontology instances. If no pattern is suitable for the selected non-ontological resource, an ad-hoc procedure is needed to transform the non-ontological resource into a conceptual model. The ad-hoc procedure may be generalized to create a new ODP. The last activity, ontology forward engineering, generates a part of actual ontology by formalizing and implementing the transformed resource.

The FMI standard presents all relevant information about the FMU in a text file called *modelDescription.xml* in an XML structure. The structure of this XML file is defined in the *fmiModelDescription.xsd"* XML Schema Definition (XSD) file. Although the ODP catalog lists a pattern for re-engineering certain embedded structures within XSD file, the pattern can not be used for this use case. Instead the ReDeFer[14] approach is used. This approach enables a mapping of XML schemes to OWL called *XSD2OWL*. Table 3.3 shows some of the translations from ReDeFer.

The *fmiModelDescription.xsd* utilizes some helper schemes to further define the base concepts of the FMI standard. The *fmi2ScalarVariable.xsd* is the most relevant one, as it defines one of the central concepts for the R4F domain, which is the information about the exposed variables. With this, the re-engineered concepts can be created and are further discussed in the Implementation Phase (3.1.4).

[14] `https://rhizomik.net/redefer`

XML Schema	OWL	Shared informal semantics
complexType|group| attributeGroup	owl:Class	Relations and contextual restrictions package
extension@base| restriction@base	rdfs:subClassOf	Package concretises the base package
element|attribute	rdf:Property owl:DatatypeProperty owl:ObjectProperty	Named relation between nodes or nodes and values
element@substitutionGroup	rdfs:subPropertyOf	Relation can appear in place of a more general one
element@type	rdfs:range	The relation range kind
sequence choice	owl:intersectionOf owl:unionOf	Combination of relations in a context

Table 3.3.: XSD2OWL translations for the XML Schema constructs and shared semantics with OWL constructs Source: [55, p.117]

3.1.3. Design Phase

The design phase mainly consists of the conceptualization activity. Besides that, NeOn also defines two additional activities, ontology localization, and ontology evolution. The first activity refers to the adaptation of an ontology to a particular language. In contrast, the latter refers to facilitating the modification of an ontology by preserving its consistency. As one of the non-functional requirements states that the ontology should be multilingual, the localization activity is performed. However, as this mainly consists of translating terms and definitions to German, it is not further explained. The translated concepts and attributes can be found in Appendix B.2.2. Ontology evolution typically occurs after the ontology has been deployed and needs to be updated/changed. Since this is not the case, this activity is also not further discussed.

The main goal of ontology conceptualization is to structure the acquired domain knowledge from the previous phases. NeOn suggests using other methodologies for this activity, such as Methontology. Methontology itself refers to the guidelines developed by Gómez-Pérez *et al.* [56]. Figure 3.6 illustrates the suggested steps to conceptualize a domain ontology. Note that, although the steps are shown in chronological order, they are not necessarily sequential in the sense of a waterfall life-cycle model, but can overlap each other. Not all suggested steps are performed in this work, since, for example, formulas are not part of the ontology scope. In the following section, the relevant steps are discussed.

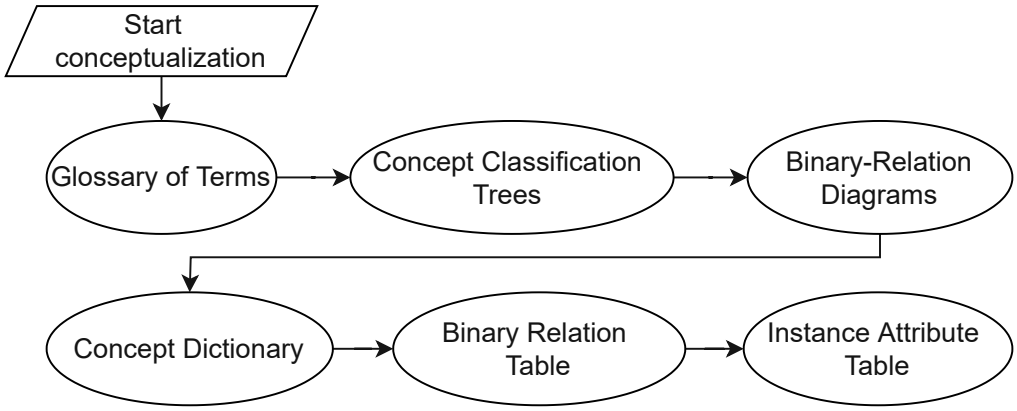


Figure 3.6.: Conceptualization according to Methontology. Source: Adapted from [56],[57]

The first step is to build a *Glossary of Terms*. The glossary should include all terms used in the ontology and its meaning. This step builds upon the pre-glossary terms identified in the ORSD and is extended by additional terms. Table 3.4 shows an example of a glossary entry. The complete glossary can be found in Appendix B.2.1 and Appendix B.2.2, respectively.

Name	Description in natural language
Connection	The connection between variables refers to the relationship between two or more variables.
Simulation Model	A simulation model is a mathematical representation of a real-world process or system over time. It is a simplified abstract view of the complex reality. It can be used to compute its expected behavior under specified conditions.
Variable	A variable represents a specific parameter or factor of a simulation model.

Table 3.4.: Part of the glossary of terms

After the glossary, containing most of the terms, is constructed a *Concept-Classification Tree* is built to group domain concepts in taxonomies. A concept classification tree organizes the concepts in a class-subclass relation and defines which concepts are linked by special subclass relations, e.g., mutually-disjoint and exhaustive-subclass-of relation. Within the R4F domain, there are not many concepts that are in a class-subclass relation with other concepts. The main concepts evolve around the *Real World Entity* concept. Figure 3.7 shows a part of the concept classification tree in the R4F domain. The concepts are mutually-disjoint. The classification tree can be found in Appendix B.3.

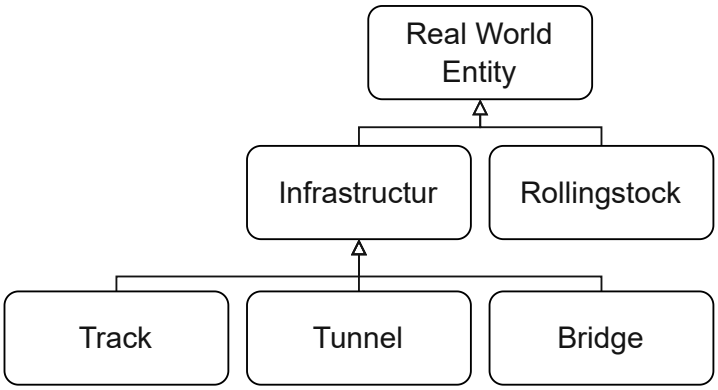


Figure 3.7.: Real World Entity Taxonomy.

The next step is to link the individual concepts and concept classification trees together via a so-called *Binary-Relations Diagram*. With this diagram, the different concepts are linked with other concepts of the same or other ontologies. A simplified example of a binary-relations diagram can be seen in Figure 3.8. Here the *Variable* concept is connected with the *SimulationModel* concept. One of the relations between these concepts is called *hasVariable* and links the *SimulationModel* concept to the *Variable* concept. There is also an inverse relation modeled; the *isVariableOf* links the *Variable* to the *SimulationModel*. This is done for all the concepts listed in the glossary of terms.

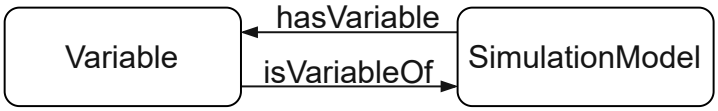


Figure 3.8.: Example for a binary relation diagram.

The *Concept Dictionary* contains concept names, class and instance attributes, and the relations for each concept in the domain. The concept dictionary aims to guarantee the completeness of the knowledge for each concept and to ensure there are no similar concepts that can lead to redundancies. The complete dictionary can be found in Appendix B.4.

The next step is to build the *Binary-Relations Table*, which contains the name of the resource, the name of the source and target (range) concept, the inverse relation, and so forth for each concept of the domain. For the R4F domain, the binary relation tables can be seen in Appendix B.5.

The next step is to build a class and instance attribute table. Class attributes describe concepts, not concept instances. This means that class attributes are shared by all instances of a class. On the other hand, instance attributes are attributes that are defined in the concept, but that take values in its instances. The attribute is unique to each individual instance. The R4F domain does not contain any class attributes; therefore, no class attribute table is constructed. The whole instance attribute table can be in the Appendix B.6.

Methontology defines more steps to help the conceptualization process, such as the construction of a formulas table, a logical axioms table, and a constants table. These steps have not been performed in this work since instances are not part of the ontology itself, and the other tables are not needed to model the R4F domain. However, the relevant instances are listed in the connect classification tree. (Appendix B.3)

Note that in RDF, relationships are called properties, as already mentioned in Section 2.3.2, and are similar concepts as attributes, as discussed above, but differ in their usage and scope. Attributes are used to represent property-value pairs associated with a resource, such as its name, age, or color. On the other hand, properties are used to describe relationships between resources. Properties are used to indicate how resources are related to each other, and they can be shared by multiple resources.

3.1.4. Implementation Phase

The main goal of the implementation phase is to transform the conceptualized model from the previous stage into a formal language, thus making it computable. As defined in the ORSD, the implementation language is OWL. One of the key benefits of using OWL to describe a domain in an ontology is that it is built on a decidable subset of first-order logic, allowing for automated reasoning to be performed. In the following, some considerations during the implementation phase are discussed.

URIs: One thing to consider when publishing an ontology on the web is the vocabulary namespace, i.e., the URI that identifies the vocabulary. There are two solutions that could be used: 303 URIs, also called slash URIs, and hash URIs. An example of both can be seen in Listing 3.1. Which one to use depends on the situation; both have advantages and disadvantages. With a hash URI, the server does not need to 303 redirect from the URI for the thing to the document about the thing. This means that a hash URI reduces the number of necessary Hypertext Transfer Protocol (HTTP) round-trips, reducing access latency. On the downside, a client has to always load the data for all other resources as well because they are in the same file. 303 URIs, on the other hand, are very flexible because a redirect can easily be configured for each target separately. However, for large data sets, many redirects affect the performance considerably. [58] In the case of the R4F ontology, the hash URI approach has been used because the number of terms is rather small, and the number of terms to be added in the future is unlikely to grow out of control.

Listing 3.1: Slash and Hash URIs

```
http://purl.org/dc/elements/1.1/title

http://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

Namespace: Another convention already used throughout this thesis is the namespace declaration and prefix for the R4F ontology. With the consideration discussed above, the proposed namespace for R4F is *http://rail4future.at/ns/main#* with the preferred prefix being *rff*.

Naming conventions: The naming conventions in the R4F ontology concerning the classes, the relations, and the attributes in the ontology, are as follows: class labels are composed of one or more words, written with a capital first letter for each word and without any space or alphanumerical symbols between the words if it is a two word one, i.e., camel-case. An example of this is *rff:SimulationModel*. The same rule applies for relations and attributes, except that they start with a non-capital letter, e.g., *rff:isConnectedWith*. The concepts, relations, and attributes always use the singular form.

Domain and Range: A common misconception when it comes to *owl:domain* and *owl:range* is that they behave as constraints. Just like other properties, they are axioms from which a reasoner can make inferences. This has already been shown in Listing 2.9. The two axioms connect pairs of individuals rather than classes. If one would need to express a relation between classes, one can use the OWL restrictions. In the R4F context, the connection between the Simulation Model and Variables has been modeled using OWL restrictions. Listing 3.2 shows the connection between the two classes using a strict binding by declaring that all values that are linked to the *SimulationModel* via the *hasVariable* property are of the class *Variable*.

Listing 3.2: OWL restriction for the SimulationModel and Variable class

```
rff:SimulationModel
    a owl:Class;
    rdfs:subClassOf
        [ a owl:Restriction ;
          owl:onProperty rff:hasVariable ;
          owl:allValuesFrom rff:Variable
        ].
```

With the same mechanism, one could define restrictions on cardinality or more restrictions on specific values. In the R4F ontology, however, no restrictions are set. Note that OWL restrictions are not meant to be used to validate RDF data since its intent is to infer new knowledge. For validation, other standards, such as the Shapes Constraint Language (Shacl)[15], should be used.

As already mentioned in the Reuse and Re-engineering Phase (3.1.2), the FMI specification is re-engineered to be used in the R4F ontology. For that process, the XSD2OWL translation mechanism is chosen. This is used to re-engineer the *Variable* and parts of the *SimulationModel* concepts. Both concepts are separated in their respective XSD files.

[15] https://www.w3.org/TR/shacl/

The *Variable* concept is explained below as an example of the transformation process. The schema can be seen in Figure 3.9, and the full XSD file can be found in Appendix A. This re-engineering process contains an exception to the rules stated in Table 3.3 as the *Real*, *Integer*, *Boolean*, *String* and *Enumeration* which themselves represent a *complexType* element are modeled as Instances of a newly created superclass *Datatype* rather than separated classes. This is due to the fact that these elements would not add value to the ontology if modeled otherwise. The specification states that exactly one of those datatypes has to be present; this statement can be modeled as a *owl:Restriction*. Other restrictions regarding those elements or classes, such as values restrictions, are not integrated into the R4F ontology.

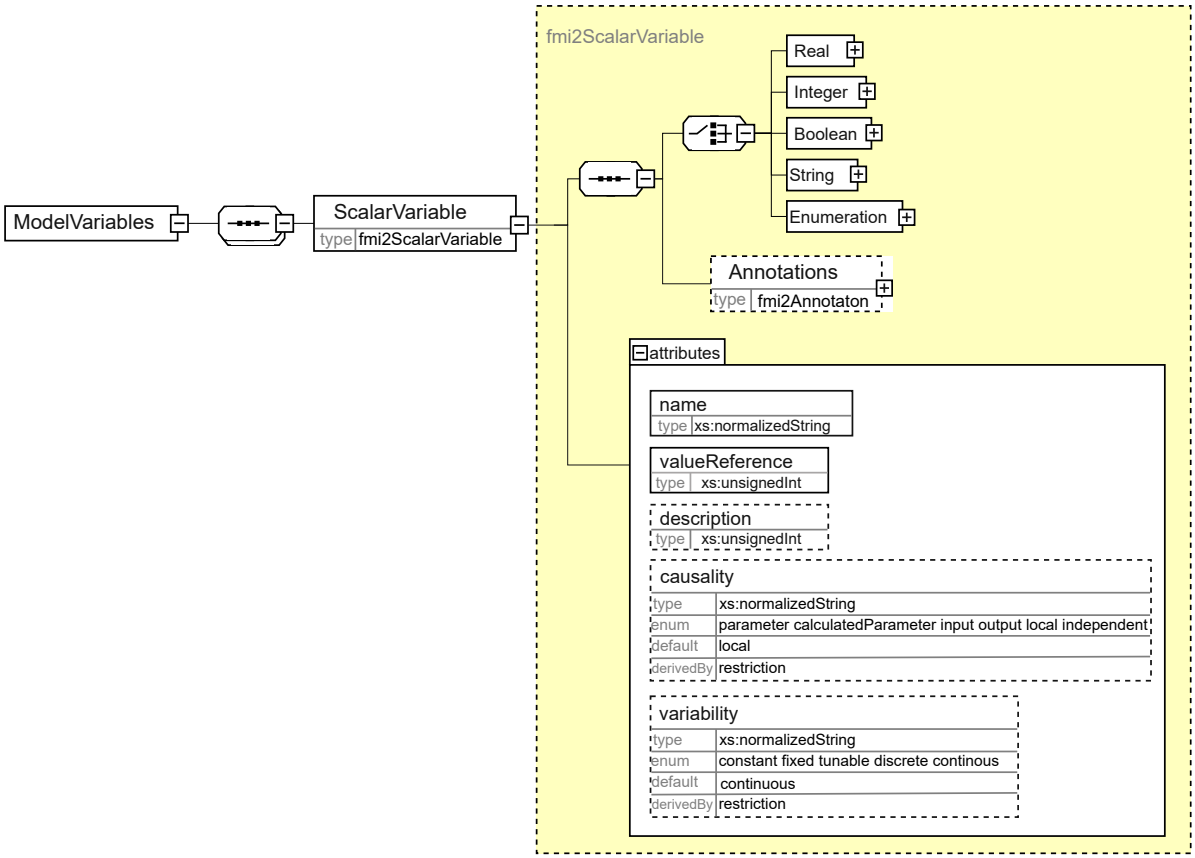


Figure 3.9.: ScalarVariable Schema. Source: Adapted from [59, p. 46]

The relevant attribute elements, i.e., *name*, *description*, and *variability*, are modeled as properties. For the *name* property, the *dc* term *dc:title* is reused, and for the *description*, the *rdfs:comment* property can be used. The *causality* element is also modeled as a class where the *input* and *output* terms are reused and represented as instances of that class. This concept is needed to link variables to one another; therefore, a separate class is beneficial. The *Annotations* element is not required for the ontology and is therefore not included. Listing 3.3 shows an example of a variable representation. The full OWL ontology can be found in Appendix A.

Listing 3.3: Variable representation in turtle

```
rff:G_Scenario_Params_Track_Superelevation a rff:Variable;
    dcterms:description "Track superelevation"@en;
    dcterms:description "Ueberhoehung der Gleise"@de;
    rff:isVariableOf rff:ErlkoenigMKSModel;
    rff:hasCausality rff:InputCausality;
    rff:hasDatatype rff:Real;
    rff:hasSemanticType rff:TrackCrossLevel;
    qudt:unit unit:M;
    qudt:hasQuantityKind quantitykind:Height;
.
```

3.2. Discussion

In this section, the R4F ontology is discussed, and the functional and non-functional requirements in the ORSD (Appendix B.1) are analyzed to verify the developed ontology meets those requirements. After that, the focus is on the implementation for some uses mentioned in the ORSD.

The first non-functional requirement states that the ontology should support multilingual scenarios in German and English. This is implemented in the ontology by translating the class labels and the descriptions as can be seen in the *Glossary of terms*, in Appendix B.2.2, the *Concept Dictionary* in Appendix B.4, as well as the *Instance Attribute Tables* in Appendix B.6. The second non-functional requirement specifies that the ontology should build and reuse standards that are used in the R4F project. The relevant standard for describing simulation models and their respective variables is the FMI standard, which is reused as discussed in the Reuse and Re-engineering Phase (3.1.2) as well as in the Implementation Phase (3.1.4).

Figure 3.10 shows the structure of the R4F ontology. The majority of the competency questions in the ORSD focus on describing the variable and simulation model. Therefore these two are the main concepts of the R4F ontology. Both concepts have an instance attribute *title* and description (QC1, QC9) as well as a short *description* to further assist the human readability of the resource. To denote the creator and maintainer of a simulation model (QC2, QC5), the *Agent* class has been utilized. Different versions of a simulation model can be described using the three different object properties *previousVersion*, *lastVersion*, and *nextVersion* (QC3). The creation date of a simulation model can be set with the instance attribute *created* (QC4), and the attribute *modified* annotates the date of the last change to a model. Additional documents belonging to the simulation model can be linked via the *supportingDocument* property and are instances of the *Document* class (CQ6). The *isLinkedTo* property links two *SimulationModel*s together with the *RealWorldEntity* class and the corresponding subclasses. With the property the specific domain which is covered by a simulation model can be expressed

(CQ7). Variables and simulation models are connected via the *isVariableOf* and its inverse *hasVariable* property (QC8, QC14). The variable is described by the aforementioned title and description as well as its variability (QC11), unit (QC12), quantity, semantic type (QC15), causality (QC10), and datatype. The terms to describe the variability of a variable are taken from the FMI standard. Units and quantities are reused from the QUDT ontology. This has not only the advantage that these terms are stable and well documented but also that the QUDT ontology provides ways of converting different units (QC13). For this purpose two JAVA libraries[16],[17] and a Python library[18] can be used. The *SemanitcType* plays an important role in connecting variables and simulation models, i.e., the *isConnectedWith* property, and is discussed in detail in the next section (QC16). The *Causality* of a variable is modeled using two instances of the same name. Although the FMI standard defines six different causalities (e.g., parameter, local, independent) the R4F ontology only utilized two of them, the *Input* and *Output*. As previously mentioned, the instances of the *Datatype* class correspond to the datatypes supported by the FMI 2 standard. The FMI standard only allows for the definition of scalar variables. To support variable collections that logically belong together, e.g., the three dimensions of a force, the *VariableGroup* class has been defined (QC17). To further extend the semantic description of different VariableGroups, the subclasses *VariableArray* and *VariableVector* are created. To further fix the need of the R4F project different subclasses can be added. However, the superclass *VariableGroup* can also be used to group different variables together, creating complex hierarchies.

A dataset can consist of one to multiple variable groups and be published in distributions (QC19). A specific distribution can have various different formats, such as a Comma-separated values (CSV) file or a simple text file. The Internet Assigned Numbers Authority (IANA)[19] media types should be used to define the media type of the distribution (QC20). Like the simulation model, the different versions of a distribution are described by the three object properties (QC22). In order to verify the integrity of a dataset, the *checksum value* attribute can be used (QC21). To assist the communication of licenses, copyrights, security references for distributions, and simulation models, the SPDX ontology is suggested. The distribution can be accessed via its *accessURL* or downloaded via the *downloadURL* (QC18). The service that provides these is described by the *DataService* class (QC24, QC25) and contains information about the endpoint of the service and access rights. With respect to different agents creating and maintaining datasets and simulation models, the R4F ontology provides a way to model the contact information for each agent (QC23). For this purpose, the Vcard ontology is recommended.

[16] `https://github.com/qudtlib/qudtlib-java`
[17] `https://github.com/egonw/jqudt`
[18] `https://github.com/eigendude/pyqudt`
[19] `https://www.iana.org/assignments/media-types/media-types.xhtml`

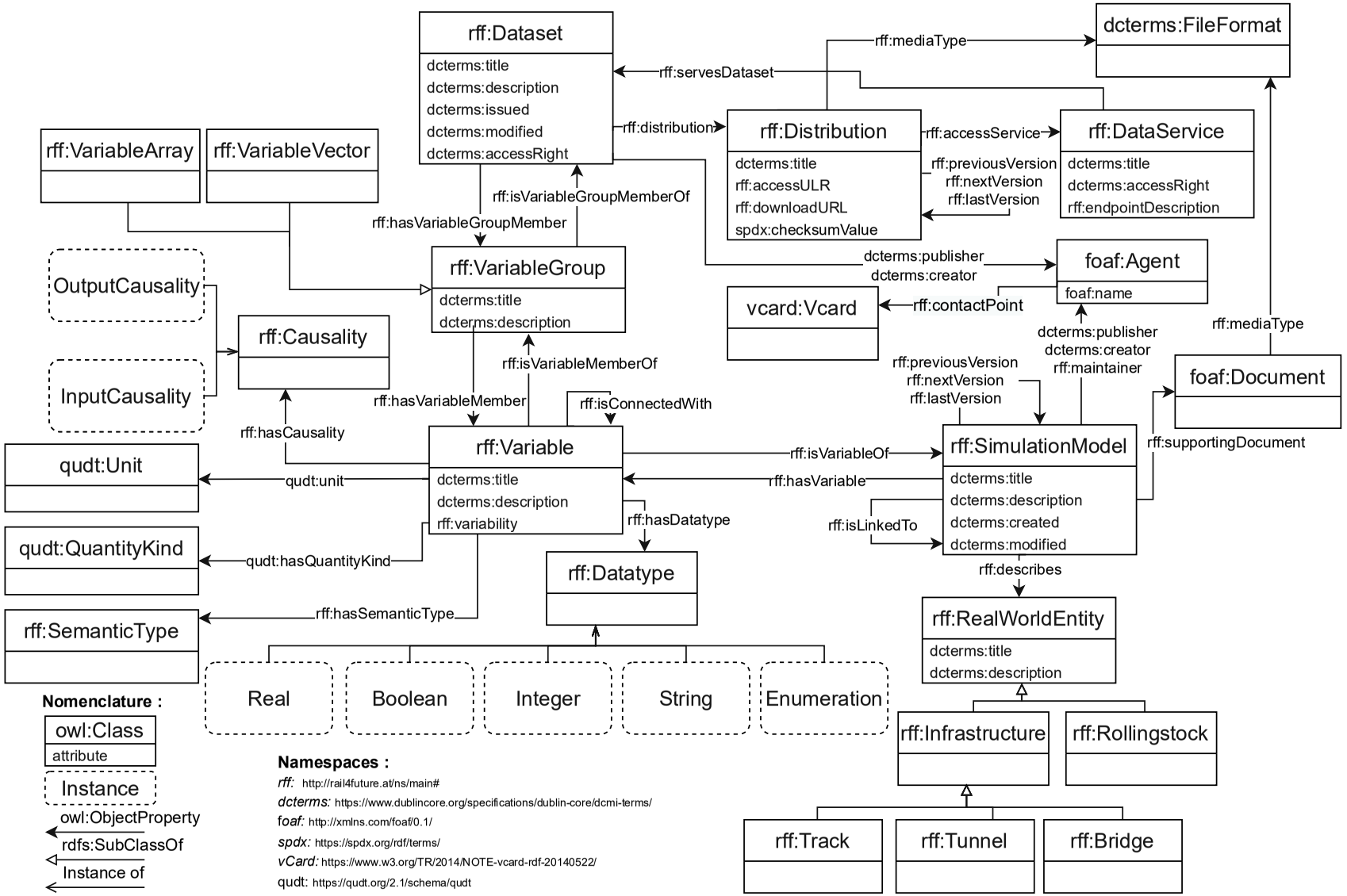


Figure 3.10.: The structure of the RFF Domain.

3.2.1. Semantics for Modular FMI assembly

One problem when describing variables is that they can hardly be distinguished if they have the same unit and quantity. In Listing 3.4, an example for that case can be seen. Note that although the presented variables do not have the same unit, they use the same base unit.

Listing 3.4: Variables with same unit and quantity

```
rff:Track_Superelevation a rff:Variable;
    dcterms:description "Track superelevation"@en;
    rff:hasDatatype rff:Real;
    qudt:unit unit:M;
    qudt:hasQuantityKind quantitykind:Height;

rff:Track_LongitudinalLevel a rff:Variable;
    dcterms:description "Track Longitudinal Level"@en;
    rff:hasDatatype rff:Real;
    qudt:unit unit:MilliM;
    qudt:hasQuantityKind quantitykind:Height;
```

While both variables have the same unit and quantity, a domain expert can tell that they do not represent the same variable by looking at the variable name, or URIs. The first variable, *Track_Superelevation*, refers to the difference in height between the top surfaces of two rails in a curve. The second variable *Track_LongitudinalLevel* refers to the vertical deviation of the rails plain to its ideal mean line. Another problem is that the same variable could have different names or URIs, as shown in Listing 3.5.

Listing 3.5: Same Variables with different names

```
rff:Track_Superelevation a rff:Variable;
    dcterms:description "Track superelevation"@en;
    rff:hasDatatype rff:Real;
    qudt:unit unit:M;
    qudt:hasQuantityKind quantitykind:Height;

rff:Track_CrossLevel a rff:Variable;
    dcterms:description "Track Cross level"@en;
    rff:hasDatatype rff:Real;
    qudt:unit unit:MilliM;
    qudt:hasQuantityKind quantitykind:Height;
```

While this does not provide a significant challenge for domain experts, this does provide a barrier when trying to connect simulation models or verify a given simulation setup automatically. To solve this problem, the variable description is extended with the *SemanticType* class as previously discussed by Mitterhofer *et al.* [47]. In conjunction with the unit and quantity properties, the semantic type provides a unique description

of the variable. For this purpose, some semantic types were defined, as can be seen in Figure 3.11. These Semantic types can be classified according to sub-systems. Since the semantic description of the variable is independent from the description of the simulation model, only the meaning of the interaction with the simulation is important. This can be seen in the *WheelRailInterface* system; for the variable description, it is irrelevant if an FMU represents the *Track* or *RollingStock* category. However, the granularity of a model does play a role when providing semantic types. For example, the *BoogieGeometrie* can be refined to the model's needs.

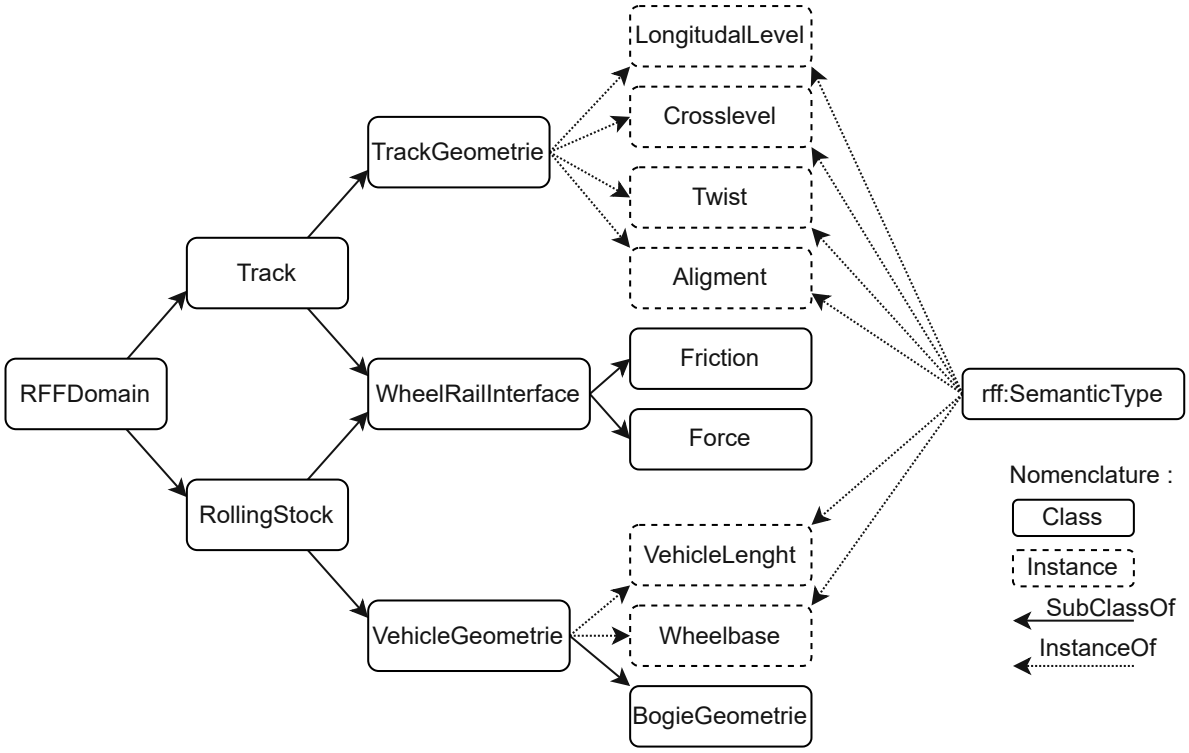


Figure 3.11.: Semantic types for the R4F domain.

In order to integrate FMUs into a co-simulation, the information of the individual simulation models and their respective variables are aggregated and combined into one KG. For inferring the *isConnectedWith* property a rule, formulated in the Semantic Web Rule Language (SWRL), is applied by a reasoner such as *Pellet*[20] or *HermiT*[21]. Listing 3.6 shows this rule to infer a connection between variables. The reasoner first collects the semantic type, causality, unit, and quantity kind for each variable (lines 1 - 9). During the reasoning, these properties are compared for each variable pair (lines 10 - 13). If the variable pair has the same semantic type, quantity, kind and unit but different causalities, the variable pair is connected via the *isConnectedWith* property (line 14).

[20] `https://github.com/lepfhty/pellet`
[21] `http://www.hermit-reasoner.com/`

Listing 3.6: SWRL rule for infering the isConnectedWith property for variables

```
1 Variable(?v1) ∧ Variable(?v2) ∧
2 hasSemanticType(?v1,?v1SemT) ∧
3 hasSemanticType(?v2,?v2SemT) ∧
4 hasCausality(?v1,?v1Caus) ∧
5 hasCausality(?v2,?v2Caus) ∧
6 qudt:unit(?v1,?v1Unit) ∧
7 qudt:unit(?v2,?v2Unit) ∧
8 qudt:hasQuantityKind(?v1,?v1QKind) ∧
9 qudt:hasQuantityKind(?v2,?v2QKind) ∧
10 sameAs(?v1SemT,?v2SemT) ∧
11 sameAs(?v1Unit,?v2Unit) ∧
12 sameAs(?v1QKind,?v2QKind) ∧
13 differentFrom(?v1Caus,?v2Caus)
14 ⟶ isConnectedWith(?v1,?v2)
```

After running the SWRL rule and adding the inferred knowledge to the graph, the graph can be queried to answer different questions. One possible question could be if a given simulation topology is valid, as stated in the intended uses in the ORSD. To answer this question, a combination of an additional SWRL rule and a simple SPARQL query can be used. Listing 3.7 shows the additional rule. It establishes the *isLinkedTo* relation between simulation models whose input and output variables are related via the *isConnectedWith* property. First, the reasoner collects all the variables that are connected to a simulation via the *hasVariable* property and their respective causality (line 1 - 5). Then the reasoner evaluates if two variables are connected via the *isConnectedWith* property (line 6). If those statements hold true, the reasoner sets the connection of the two simulation models by setting the model of the output variable as the subject and the model of the input variable as the object in the *isLinkedTo* statement (lines 7 - 9). This ensures that only direct subsequent simulation models are linked, and the relation is only valid in one direction.

Listing 3.7: SWRL rule for infering the isLinkedTo property for simulation models

```
1 rff:SimulationModel(?sim1) ∧ rff:SimulationModel(?sim2) ∧
2 rff:hasVariable(?sim1,?v1) ∧
3 rff:hasVariable(?sim2,?v2) ∧
4 rff:hasCausality(?v1,?v1Causality) ∧
5 rff:hasCausality(?v2,?v2Causality) ∧
6 rff:isConnectedWith(?v1,?v2) ∧
7 sameAs(?v1Causality,rff:OutputCausality) ∧
8 sameAs(?v2Causality,rff:InputCausality)
9 ⟶ rff:isLinkedTo(?sim1,?sim2)
```

In order to verify that a given simulation topology is valid, the query, as shown in Listing 3.8, can be used. The query utilizes the *ASK* query form, which returns a

boolean indicating whether a query pattern matches. The statement in line 3 is a so-called property path query that allows constructing complex path structure to be queried instead of being limited to direct neighbors. In the shown example, the query looks for a path between the simulation model "*Sim1*" and the simulation model "*Sim2*" via the *rff:isLinkedTo* property by one or more matches of the property, indicated by the "+". Property paths also allow for the search of inverse paths to match the nodes in reverse order.

Listing 3.8: SPARQL query to validate simulation topology

```
PREFIX rff: <http://rail4future.at/ns#>

ASK {rff:Sim1 rff:isLinkedTo+ rff:Sim4 }
```

Unfortunately, with SPARQL, it is not possible to get the names of all the nodes in such a path. For this purpose, a different query language must be used. In this work, the graph path search functionality of *Ontotext GraphDB*[22] is used. However, other languages such as ArangoDB Query Language (AQL) should also be able to perform such a task. *GraphDB* provides different path search algorithms to search for available paths between two nodes. The shortest path algorithm is used in Listing 3.9. As the name already suggests, it returns the shortest path between two nodes. If several paths have the same length, all of them are returned.

Listing 3.9: Shortest path query in GraphDB

```
PREFIX path: <http://www.ontotext.com/path#>
PREFIX rff: <http://rail4future.at/ns#>

SELECT ?edge
WHERE {
    Values(?src ?dst){
        (rff:Var1 rff:Var2)
    }
    Service path:search{
        [] path:findPath path:shortestPath ;
           path:sourceNode ?src ;
           path:destinationNode ?dst ;
           path:resultBinding ?edge .
    }
}
```

The example in Listing 3.9 returns all the edges between the *sourceNode*, Var1, and the *destinationNode*, Var2, as a RDF-star statement. With this result, it is possible to assist the generation of the simulation topology as defined in the intended uses in the ORSD.

[22] https://www.ontotext.com/

3.2.2. Ontology Annotation

In the following section, the process of annotating information to the FMU, converting it to a OWL file and using this file to realize the automated generation of a simulation topology. Figure 3.12 shows the proposed workflow. The process starts with a domain expert generating a simulation model compliant with the FMI standard. The resulting FMU file is then extended with an OWL file. This is done by reading the *modelDescription.xml* file, which already contains some metadata of the simulations variables and the model itself. The domain expert then can add additional information and necessary information not provided by the FMU file. This is done by the *modelDescription.xml* file or directly importing the FMU file into an GUI. The GUI guides the agent through the annotation process. The resulting *.owl* file is then stored alongside the original file within the *.fmu* ZIP file. The extended FMU file can then be stored in a database whit other extended files. To generate a simulation topology, the individual *.owl* files are extracted from the database and are fed alongside the specified requirements to a reasoner. The reasoner then infers the connections of the *.fmu* file as already discussed in the previous section (3.2.1).

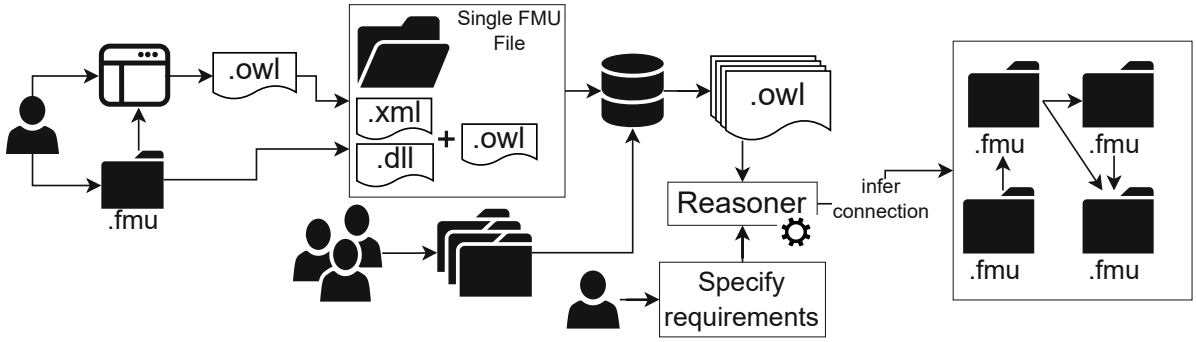


Figure 3.12.: Proposed workflow to annotate an FMU and automatically generate a simulation topology.

Figure 3.13 shows the Ontology Annotation User Interface (UI) used in the proposed workflow and developed within this thesis. The UI was developed using the PyQt[23] Framework. To transform the obtained data into an *.owl* file the Python library RDFLib[24] has been used. The GUI is developed to enable domain experts with limited to no ontological knowledge to annotate the relevant information. This is in accordance with the intended end-users as stated in the ORSD. Figure 3.13a shows the general information about the simulation model. Fields marked with an asterisk are obligatory and checked before creating the OWL file. In Figure 3.13b and Figure 3.13c, the information for the input and output variables can be set. From the drop-down menu, the variables are selected, and the available information is automatically loaded. If the FMU contains a unit definition, these must be transformed to match the QUDT syntax. As an

[23] https://pypi.org/project/PyQt5/
[24] https://rdflib.readthedocs.io/

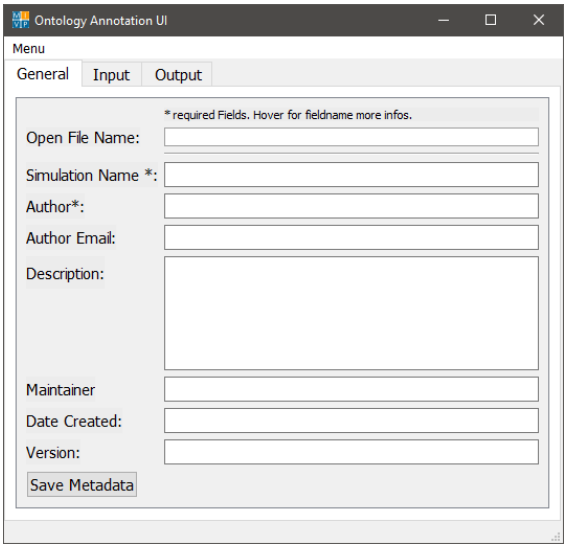
example, the FMU may contain the unit *m/s* as a unit of velocity, and the same QUDT individual is represented as *M-PER-SEC*. This conversion has to be done by the agent. For the definition of the quantity, the agent also has to follow the naming convention of the QUDT ontology. For both fields, a link to the respective documentation is provided in the help menu. A good way to start when assigning a quantity to a unit is to look at the unit's predefined quantity kinds. This process could be automated by querying these predefined values. Listing 3.10 shows an example of such a query, including the returned results. However, the predefined values of the QUDT ontology are incomplete, meaning that not all possible quantity kinds are assigned to a unit. One example of this is the results when querying the unit *unit:M* the only result is *quantitykind:Length* when *quantitykind:Height* or *quantitykind:Diameter* would also be valid options.

Listing 3.10: Query predefined quantity kinds for a specific unit

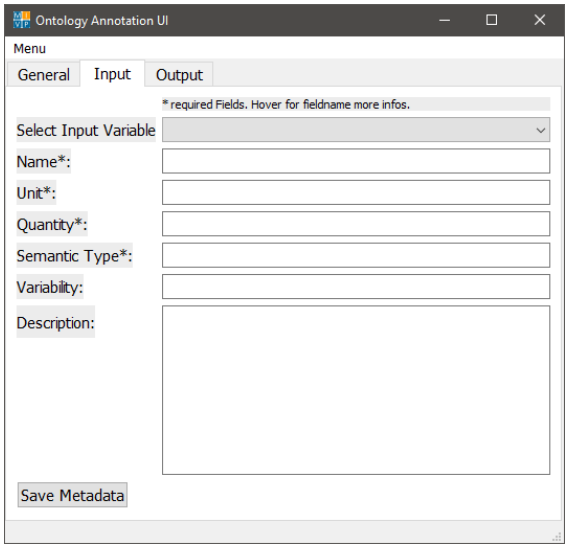
```
PREFIX unit: <http://qudt.org/vocab/unit/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX qudt: <http://qudt.org/schema/qudt/>
PREFIX quantitykind: <http://qudt.org/vocab/quantitykind/>

SELECT ?qk
WHERE {
    BIND (unit:PA AS ?arg1) .
    ?un rdf:type qudt:Unit .
    FILTER (?un = ?arg1) .
    ?un qudt:hasQuantityKind ?qk
} ORDER BY ?un
-----------------------------------
Results:
-----------------------------------
quantitykind:BulkModulus
quantitykind:ForcePerArea
quantitykind:Fugacity
quantitykind:ModulusOfElasticity
quantitykind:ShearModulus
quantitykind:VaporPressure
```

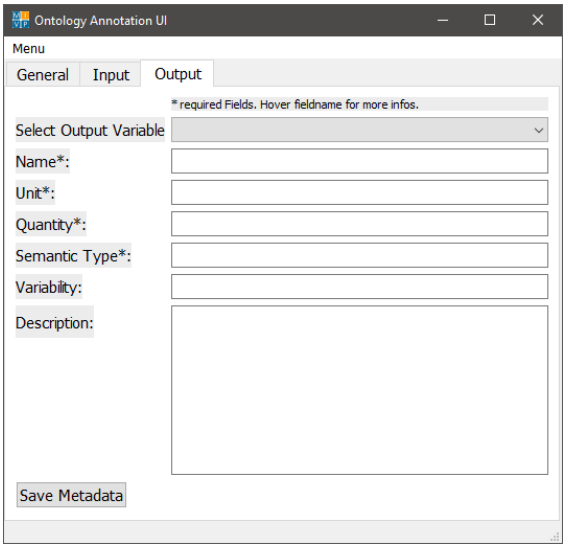
In regards to the semantic type field the agent has to select the appropriate individual from the R4F semantic type definition as shown in Figure 3.11. Note that the FMI standard restricts the values for the variability of input and output variables. This attribute is also optional and may not be automatically obtained form the FMU. For this reason the agent should refer to these restrictions as discussed in the FMI specification [59, p. 51]. For all other additional information provided by the agent there is no specific format to which the annotation should comply to. After saving the annotation file it is transformed into an RDF graph.



(a) General Information



(b) Input Variables



(c) Output Variables.

Figure 3.13.: Ontlogy Annotation UI.

4. Conclusion and Future Work

In this thesis the development process of the R4F ontology and its possible applications in Co-Simulation were presented. The main focus laid on the description of the simulation model and its variables, while still provide a vocabulary to add additional information. The research questions as defined in Chapter 1.2 are used to discuss the results and identify future research directions.

RQ1 How are semantic technologies currently used to describe the railway domain?

Chapter 2.5.1 summarizes the current application of semantic technologies in the railway domain. Most of the mentioned projects have already concluded and didn't make it past the academic research phase. The developed and proposed ontologies are not maintained further or are no longer available online. However, newer undertakings such as the ERA vocabulary advance the usage of semantic technologies in the railway domain. Another activity in incorporating semantic technologies in the railway domain is the initiation of a ontology subgroup within the *railML* community. They aim to interlink data from various standards and use them in different processes. These projects could lead to a wider use of ontologies in the railway domain.

RQ2 Are there applications of semantic technologies in multi-domain simulations?

The application of semantic technologies in multi-domain simulations is summarized in Chapter 2.5.2. The different authors all use different approaches when it comes to defining the components of a simulation model and the model itself, from predefining fixed ports or connectors and their relation to other connectors to inferring a connection via various attributes of a variable. Some concepts have been implemented in the developed ontology. The most significant one is the *SemanticType*. Even though the authors all discuss their concepts in using ontologies in their respective fields, none of the mentioned ontologies are published and available to the public.

RQ3 How can semantic technologies be used for metadata sharing of distributed heterogeneous data sources?

The presented literature in Chapter 2.5.2 focused on the description of simulation models and their variables. However, a DT is often composed of distributed and heterogeneous data sources from different stakeholders. For this, Moshrefzadeh *et al.* [44] presented a distributed DT by leveraging on concepts of data catalogs. This approach is included in the R4F ontology and enables the different stakeholders to provide additional information for their assets. Although the terms in the R4F ontology are based on existing vocabularies for data catalogs, the ontology defines its own terms to fit better the needs of a distributed DT. This approach enables easy expansions of terms in the future, as

well as an independent development of the vocabulary without the dependency on the original vocabulary.

RQ4 How can domain experts with limited knowledge of ontologies utilize these semantic technologies?

Since most simulation and domain experts have little to no knowledge of ontologies, the ontology annotation UI has been created. The simple GUI assists creators of simulation models in annotating the required information and transforming them into a machine-readable form. The GUI is part of the proposed workflow to annotate individual FMU files and include them in a reasoning process to automatically generate and verify simulation topologies. For this purpose, the necessary SWRL rules and example queries have been presented in Chapter 3.2.1.

This work outlined the development and possible applications of an ontology for the R4F project. One important part of the ontology is the *SemanticType*, which provides a unique description of the variable. This concept, however, is still under development and needs to be refined to fit the R4F domain. For this process, the domain experts should be involved in determining how detailed the instances and classes have to be modeled. Another aspect open to future work is the continued development of the proposed workflow to automatically generate a simulation topology. The workflow could be extended with the possibility of exporting an SSP file of the connected FMU files and handing this file to an orchestrator to run the simulation setup.

List of Figures

List of Listings

List of Tables

Bibliography

[1] J. Tutcher, "Development of semantic data models to support data interoperability in the rail industry," Ph.D. dissertation, University of Birmingham, UK, 2016. [Online]. Available: http://etheses.bham.ac.uk/6774/.

[2] A. P. Sheth, "Changing focus on interoperability in information systems:from system, syntax, structure to semantics," in *Interoperating Geographic Information Systems*, Springer US, 1999, pp. 5–29. DOI: 10.1007/978-1-4615-5189-8_2.

[3] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, Jun. 1993. DOI: 10.1006/knac.1993.1008.

[4] R. Studer, V. Benjamins, and D. Fensel, "Knowledge engineering: Principles and methods," *Data & Knowledge Engineering*, vol. 25, no. 1-2, pp. 161–197, Mar. 1998. DOI: 10.1016/s0169-023x(97)00056-6.

[5] W. N. Borst, "Construction of engineering ontologies for knowledge sharing and reuse," Undefined, Ph.D. dissertation, University of Twente, Netherlands, Sep. 1997, ISBN: 90-365-0988-2. [Online]. Available: https://research.utwente.nl/en/publications/construction-of-engineering-ontologies-for-knowledge-sharing-and-.

[6] M. Hepp, "Ontologies: State of the art, business potential, and grand challenges," in *Ontology Management: Semantic Web, Semantic Web Services, and Business Applications*, M. Hepp, P. De Leenheer, A. De Moor, and Y. Sure, Eds. Boston, MA: Springer US, 2008, pp. 3–22, ISBN: 978-0-387-69900-4. DOI: 10.1007/978-0-387-69900-4_1. [Online]. Available: https://doi.org/10.1007/978-0-387-69900-4_1.

[7] L. Obrst, "Ontological architectures," in *Theory and Applications of Ontology: Computer Applications*, R. Poli, M. Healy, and A. Kameas, Eds. Springer Netherlands, 2010, pp. 27–66, ISBN: 978-90-481-8847-5. DOI: 10.1007/978-90-481-8847-5_2. [Online]. Available: https://doi.org/10.1007/978-90-481-8847-5_2.

[8] N. Guarino, "Semantic matching: Formal ontological distinctions for information organization, extraction, and integration," in *Information Extraction A Multidisciplinary Approach to an Emerging Information Technology*, Springer Berlin Heidelberg, 1997, pp. 139–170. DOI: 10.1007/3-540-63438-x_8.

[9] M. Fernández-López, A. Gómez-Pérez, and N. Juristo, "Methontology: From ontological art towards ontological engineering," in *Proceedings of the Ontological Engineering AAAI-97 Spring Symposium Series*, Ontology Engineering Group ? OEG, American Asociation for Artificial Intelligence, Mar. 1997. [Online]. Available: https://oa.upm.es/5484/.

[10] H. S. Pinto and J. P. Martins, "Ontologies: How can they be built?" *Knowledge and Information Systems*, vol. 6, no. 4, pp. 441–464, Jul. 2004. DOI: 10.1007/s10115-003-0138-1.

[11] N. F. Noy and D. L. McGuinness, "Ontology development 101: A guide to creating your first ontology," Tech. Rep., Mar. 2001. [Online]. Available: http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html.

[12] M. A. Musen, "The protégé project: A look back and a look forward," *AI Matters*, vol. 1, no. 4, pp. 4–12, Jun. 2015. DOI: 10.1145/2757001.2757003. [Online]. Available: https://doi.org/10.1145/2757001.2757003.

[13] M. C. Suárez-Figueroa, A. Gómez-Pérez, and M. Fernández-López, "The NeOn methodology for ontology engineering," in *Ontology Engineering in a Networked World*, Springer Berlin Heidelberg, Dec. 2011, pp. 9–34. DOI: 10.1007/978-3-642-24794-1_2.

[14] WordWideWebConsortium, *W3C semantic web FAQ*, Accessed: 2023-01-03. [Online]. Available: https://www.w3.org/2001/sw/SW-FAQ.

[15] P. Hitzler, *Foundations of Semantic Web technologies*. CRC Press, 2010, p. 427, ISBN: 9781420090505.

[16] A. Hogan, "Linked data & the semantic webstandards," in *Linked Data Management*, A. Harth, K. Hose, and R. Schenkel, Eds., Taylor & Francis Group, 2016, p. 576, ISBN: 9781466582415.

[17] The Unicode Consortium, "The Unicode Standard," Unicode Consortium, Mountain View, CA, Tech. Rep. Version 15.0.0, Sep. 2022. [Online]. Available: http://www.unicode.org/versions/Unicode15.0.0/.

[18] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifier (URI): Generic syntax," Tech. Rep. 3986, Jan. 2005, 61 pp. DOI: 10.17487/rfc3986. [Online]. Available: https://www.rfc-editor.org/info/rfc3986.

[19] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform resource locators (URL)," Tech. Rep. 1738, Dec. 1994, 25 pp. DOI: 10.17487/rfc1738. [Online]. Available: https://www.rfc-editor.org/info/rfc1738.

[20] L. Daigle, D. van, R. Iannella, and P. Faltstrom, "Uniform resource names (URN) namespace definition mechanisms," Tech. Rep. 3406, Oct. 2002, 22 pp. DOI: 10.17487/rfc3406. [Online]. Available: https://www.rfc-editor.org/info/rfc3406.

[21] M. Duerst and M. Suignard, "Internationalized resource identifiers (IRIs)," Tech. Rep. 3987, Jan. 2005, 46 pp. DOI: 10.17487/rfc3987. [Online]. Available: https://www.rfc-editor.org/info/rfc3987.

[22] P. Krauss, Mysid, Rjgodoy, and Surachit, *File:uri venn diagram.svg*, https://en.wikipedia.org/wiki/File:URI_Venn_Diagram.svg, 2007. [Online]. Available: https://en.wikipedia.org/wiki/File:URI_Venn_Diagram.svg.

[23] W3C SPARQL Working Group, "SPARQL 1.1 overview," W3C, W3C Recommendation, Mar. 2013. [Online]. Available: https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/.

[24] M. Kifer and H. Boley, "RIF overview (second edition)," W3C, W3C Note, Feb. 2013. [Online]. Available: https://www.w3.org/TR/2013/NOTE-rif-overview-20130205/.

[25] S. Sizov, "What makes you think that? the semantic web's proof layer," *IEEE Intelligent Systems*, vol. 22, no. 6, pp. 94–99, Nov. 2007. DOI: 10.1109/mis.2007.120.

[26] D. Artz and Y. Gil, "A survey of trust in computer science and the semantic web," *Journal of Web Semantics*, vol. 5, no. 2, pp. 58–71, Jun. 2007. DOI: 10.1016/j.websem.2007.03.002.

[27] D. Wood, R. Cyganiak, and M. Lanthaler, "RDF 1.1 concepts and abstract syntax," W3C, W3C Recommendation, Feb. 2014. [Online]. Available: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[28] G. Schreiber and Y. Raimond, "RDF 1.1 primer," W3C, W3C Note, Jun. 2014. [Online]. Available: https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

[29] P.-A. Champin, G. Kellogg, and D. Longley, "JSON-ld 1.1," W3C, W3C Recommendation, Jul. 2020. [Online]. Available: https://www.w3.org/TR/2020/REC-json-ld11-20200716/.

[30] R. Guha and D. Brickley, "RDF schema 1.1," W3C, W3C Recommendation, Feb. 2014. [Online]. Available: https://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[31] W. O. W. Group, "OWL 2 web ontology language document overview (second edition)," W3C, Tech. Rep., Dec. 2012. [Online]. Available: https://www.w3.org/TR/2012/REC-owl2-overview-20121211/.

[32] I. Horrocks, Z. Wu, B. Motik, A. Fokoue, and B. C. Grau, "OWL 2 web ontology language profiles (second edition)," W3C, W3C Recommendation, Dec. 2012. [Online]. Available: https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/.

[33] M. W. Grieves, "Virtually intelligent product systems: Digital and physical twins," in *Complex Systems Engineering: Theory and Practice*, American Institute of Aeronautics and Astronautics, Inc., Jan. 2019, pp. 175–200. DOI: 10.2514/5.9781624105654.0175.0200.

[34] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital twin in manufacturing: A categorical literature review and classification," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1016–1022, 2018. DOI: 10.1016/j.ifacol.2018.08.474.

[35] M. Wiens, T. Meyer, and P. Thomas, "The potential of FMI for the development of digital twins for large modular multi-domain systems," in *Linköping Electronic Conference Proceedings*, Linköping University Electronic Press, Sep. 2021. DOI: 10.3384/ecp21181235.

[36] C. Gomes, T. Blochwitz, C. Bertsch, *et al.*, "The FMI 3.0 standard interface for clocked and scheduled simulations," in *Linköping Electronic Conference Proceedings*, Linköping University Electronic Press, Sep. 2021. DOI: 10.3384/ecp2118127.

[37] T. Blockwitz, M. Otter, J. Akesson, *et al.*, "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models," in *Linköping Electronic Conference Proceedings*, Linköping University Electronic Press, Nov. 2012. DOI: 10.3384/ecp12076173.

[38] A. Junghanns, C. Gomes, C. Schulze, *et al.*, "The functional mock-up interface 3.0 - new features enabling new applications," in *Linköping Electronic Conference Proceedings*, Linköping University Electronic Press, Sep. 2021. DOI: 10.3384/ecp2118117.

[39] J. A. Rojas, M. Aguado, P. Vasilopoulou, *et al.*, "Leveraging semantic technologies for digital interoperability in the european railway domain," in *The Semantic Web – ISWC 2021*, Springer International Publishing, 2021, pp. 648–664. DOI: 10.1007/978-3-030-88361-4_38.

[40] S. Verstichel, F. Ongenae, L. Loeve, *et al.*, "Efficient data integration in the railway domain through an ontology-based methodology," *Transportation Research Part C: Emerging Technologies*, vol. 19, no. 4, pp. 617–643, Aug. 2011. DOI: 10.1016/j.trc.2010.10.003.

[41] J. Tutcher, J. M. Easton, and C. Roberts, "Enabling data integration in the rail industry using RDF and OWL: The RaCoOn ontology," *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part A: Civil Engineering*, vol. 3, no. 2, Jun. 2017. DOI: 10.1061/ajrua6.0000859.

[42] S. Bischof and G. Schenner, "Towards a railway topology ontology to integrate and query rail data silos," en, 2020. DOI: 10.13140/RG.2.2.22571.98084.

[43] S. Bischof and G. Schenner, "Rail topology ontology: A rail infrastructure base ontology," in *The Semantic Web – ISWC 2021*, Springer International Publishing, 2021, pp. 597–612. DOI: 10.1007/978-3-030-88361-4_35.

[44] M. Moshrefzadeh, T. Machl, D. Gackstetter, A. Donaubauer, and T. H. Kolbe, "Towards a distributed digital twin of the agricultural landscape," en, *Journal of Digital Landscape Architecture*, no. 5, 2020. DOI: 10.14627/537690019.

[45] L. Kasper, F. Birkelbach, P. Schwarzmayr, G. Steindl, D. Ramsauer, and R. Hofmann, "Toward a practical digital twin platform tailored to the requirements of industrial energy systems," *Applied Sciences*, vol. 12, no. 14, p. 6981, Jul. 2022. DOI: 10.3390/app12146981.

[46] H. Nordahl, M. Rindarøy, S. Skjong, L. T. Kyllingstad, D. Walther, and T. Brekke, "An ontology-based approach for simplified FMU variable connections with automatic verification of semantically correct configuration," in *Volume 6A: Ocean Engineering*, American Society of Mechanical Engineers, Aug. 2020. DOI: 10.1115/omae2020-18135.

[47] M. Mitterhofer, G. F. Schneider, S. Stratbücker, and K. Sedlbauer, "An FMI-enabled methodology for modular building performance simulation based on semantic web technologies," *Building and Environment*, vol. 125, pp. 49–59, Nov. 2017. DOI: 10.1016/j.buildenv.2017.08.021.

[48] M. Mitterhofer, G. F. Schneider, S. Stratbu¨cker, and S. Steiger, "Semantics for assembling modular network topologies in FMI-based building performance simulation," in *Building Simulation Conference Proceedings*, IBPSA, Aug. 2017. DOI: 10.26868/25222708.2017.418.

[49] A. Gómez-Pérez and M. C. Suárez-Figueroa, *Ontology requirements specification*. [Online]. Available: http://neon-project.org/nw/book-chapters/Chapter-05.pdf.

[50] B. Villazón-Terrazas and A. Gómez-Pérez, *Re-engineering non-ontological resources*. [Online]. Available: http://neon-project.org/nw/book-chapers/Chapter-08-1.pdf.

[51] X. Zhang, K. Li, C. Zhao, and D. Pan, "A survey on units ontologies: Architecture, comparison and reuse," *Program*, vol. 51, no. 2, pp. 193–213, Jul. 2017. DOI: 10.1108/prog-08-2015-0056.

[52] J. M. Keil and S. Schindler, "Comparison and evaluation of ontologies for units of measurement," *Semantic Web*, vol. 10, no. 1, B. Brodaric, Ed., pp. 33–51, Dec. 2018. DOI: 10.3233/sw-180310.

[53] H. Rijgersberg, M. van Assem, and J. Top, "Ontology of units of measure and related concepts," *Semantic Web*, vol. 4, no. 1, pp. 3–13, 2013. DOI: 10.3233/sw-2012-0069.

[54] FAIRsharing Team, *Fairsharing record for: Quantities, units, dimensions and types*, 2015. DOI: 10.25504/FAIRSHARING.D3PQW7.

[55] R. García Gonzáles, "A semantic web approach to digital rights management," Ph.D. dissertation, Universitat Pompeu Fabra, Apr. 2006.

[56] A. Gómez-Pérez, M. Fernàndez, and A. de Vicente, "Towards a method to conceptualize domain ontologies," 1996.

[57] M. López, A. Gómez-Pérez, J. Sierra, and A. Pazos, "Building a chemical ontology using methontology and the ontology design environment. ieee intelligent systems," *Intelligent Systems and their Applications, IEEE*, vol. 14, pp. 37–46, Feb. 1999. DOI: 10.1109/5254.747904.

[58] R. Cyganiak and L. Sauermann, "Cool URIs for the semantic web," W3C, W3C Note, Dec. 2008, https://www.w3.org/TR/2008/NOTE-cooluris-20081203/.

[59] M. Association, "Functional mock-up interface for model exchange and co-simulation," Tech. Rep., Nov. 2022. [Online]. Available: https://github.com/modelica/fmi-standard/releases/download/v2.0.4/FMI-Specification-2.0.4.pdf.

A. List of code and Ontologies hosted online

Rather than providing full code and OWL listing in print, source code for the mentioned application and the full ontology designed as part of this thesis are available online, and can be accessed by the following URLs:

- Rail4Future (R4F) Ontology
  - `https://gitlab.tuwien.ac.at/mivp/rail4future/diplomarbeit_kern/tree/main/RFFOntology`
- Annotation Application Source Code
  - `https://gitlab.tuwien.ac.at/mivp/rail4future/diplomarbeit_kern/tree/main/OntologyAnnotationGUI`
- XSD Files from FMI 2.0.4 Specification
  - `https://github.com/modelica/fmi-standard/releases/download/v2.0.4/FMI-Standard-2.0.4.zip`

B. Ontology Development Documents

B.1. Ontology Requirements Specification Document

Table B.1.: Ontology Requirements Specification Document

	R4F Reference Ontology Requirements Specification Document
1	**Purpose**
	The purpose of the R4F Ontology is to provide a consensual knowledge model of the R4F domain to be used by the operators of the R4F digital twin platform.
2	**Scope**
	The ontology not only has to focus on the description of simulation models and their respective input and output variables but also provide additional information about relevant metadata about those variables and simulation model for easier data exchange and data integration.
3	**Implementation Language**
	The implementation of the ontology must be done using the OWL language in order to utilize the first-order logic based reasoning capability.
4	**Intended End-Users**
	User 1. Developer of the digital twin platform with limited knowledge about ontologies **User 2.** Ontology engineers for maintenance and enhancement of the ontology network **User 3.** Domain experts who submit simulations to the digital twin platform with limited knowledge about ontologies **User 4.** User of digital twin platform with no ontological knowledge
5	**Intended Uses**
	Use 1. Publish information about a specific asset or asset group **Use 2.** Searching for information about a specific asset or asset group **Use 3.** Assist the automated generation of the simulation topology **Use 4.** Automatically verify a given simulation setup
6	**Ontology Requirements**
	a. Non-Functional Requirements
	NFR1. The ontology must support a multilingual scenario in the following languages: English, German **NFR2.** The ontology must be based on standards used in the R4F project.

. . . continued on next page

... continued from previous page

	b. Functional Requirements: Groups of Competency Questions
	CQG1 Simulation Model & Variables
	CQ1. What is the name of the simulation model ? **CQ2.** Who created the simulation model ? **CQ3.** What versions exist of the simulation model ? **CQ4.** When i a simulation model created ? **CQ5.** Who is responsible for maintaining the simulation model ? **CQ6.** What additional documents do exists for a simulation model ? **CQ7.** What aspect of the R4F domain does the simulation model model ? **CQ8.** To which model does a variable belong ? **CQ9.** What is the name of the variable ? **CQ10.** What is the causality of a variable ? **CQ11.** Does a variable change its value during the Simulation ? **CQ12.** What is the unit of a variable ? **CQ13.** Do variable A and variable B have the same the same base unit ? **CQ14.** Which variable are input/output to the same simulation model ? **CQ15.** What aspect of the R4F domain does the variable describe ? **CQ16.** How are variable of different causality connected ? **CQ17.** How can different variables be grouped ?
	CQG2 Additional Metadata
	CQ18. Where are Datasets published ? **CQ19.** What Distribution do exist from a dataset ? **CQ20.** What is the media type of a distribution ? **CQ21.** How can be ensured that a distribution has not been modified ? **CQ22.** Are there different versions of a distribution ? **CQ23.** How can a Author/Maintainer/Publisher be contacted ? **CQ24.** Where can an dataset be found ? **CQ25.** How can Dataset be obtained ?
7	**Pre-Glossary of Terms**

Simulation Model	Contact
Version	Dataset
Document	Distribution
Aspect of Domain	Media Type
Variable	Author
Causality	Maintainer
(Base) Unit	Publisher
Variable Group	Agent
Connection	Data Service

B.2. Glossary of terms

B.2.1. English Version

Term	Definition in natural language
Agent	A person, group or organization that can do stuff.
Aspect of Domain	The aspect of a domain refers to a specific area of the R4F domain which is modeled by the simulation model.
Bridge	A bridge is constructed to traverse physical objects such as a body of water, valley, or roadway, with the intention of facilitating passage over the obstruction.
Causality	Causality of a variable refers to the relationship between the variable and the simulation model. (e.g., Input or Output variable)
Connection	The connection between variables refers to the relationship between two or more variables.
Contact	Relevant contact information for the cataloged resource. Use of vCard[1] is recommended.
Creator	The agent responsible for for designing and developing the resource.
DataService	A collection of operations that provides access to one or more datasets.
Dataset	A collection of data, published or curated by a (multiple) agents, and available for access or download in one or more representations.
Datatype	Specifies which type of value a variable has. This includes integers, Boolean values and strings.
Distribution	A specific representation of a dataset. The dataset might be available in multiple serializations that may differ in various ways, including natural language or media-type or format.

. . . continued on next page

[1] https://www.w3.org/TR/2014/NOTE-vcard-rdf-20140522/

... continued from previous page

Term	Definition in natural language
Document	A resource that represents those things which are, broadly conceived, 'documents' (e.g. Documentation, License Document).
Infrastructure	Infrastructure is the set of facilities and systems that supports the operation of the railway system. This includes the tracks, bridges, tunnels, signaling systems, and other structures and facilities that are necessary for the safe and efficient movement of trains.
Maintainer	The agent who is responsible to maintain and updating the resource.
Media Type	The file format of the resource. Use of the IANA [1]media types is recommended.
Publisher	The agent responsible for making the resource (Simulation Model, Dataset or Data Service) available.
Rolling Stock	Rolling stock refers to railway vehicles, including both powered and unpowered vehicles: for example, locomotives, freight and passenger cars (or coaches), and non-revenue cars.
Simulation Model	A simulation model is a mathematical representation of a a real-world process or system over time. It is a simplified abstract view of the complex reality. It can be used to compute its expected behavior under specified conditions.
Track	A railway track (British English and UIC terminology) or railroad track (American English), is the structure on a railway or railroad consisting of the rails, fasteners, railroad ties and ballast. It enables trains to move by providing a dependable surface for their wheels to roll upon.
Tunnel	A tunnel is an artificial passage that passes under mountains, bodies of water or other obstacles.
Unit	A unit of measure, or unit, is a particular quantity value that has been chosen as a scale for measuring other quantities the same kind (more generally of equivalent dimension).

... continued on next page

[2]https://www.iana.org/assignments/media-types/media-types.xhtml

... continued from previous page

Term	Definition in natural language
Variable	A variable represents a specific parameter or factor of a simulation model.
Variable Array	A variable array is collection of variables that represent a specific array of values (e.g. Inertia tensor).
Variable Group	A variable group is collection of variables that share a common characteristic (e.g., causality, unit or aspect of a domain).
Variable Vector	A variable vector is collection of variables that represent a vector (e.g., force, acceleration).
Version	The version indicator, usually an identifier, of a resource.

B.2.2. German Version

Term	Definition in natürlicher Sprache
Agent	Eine Person, Gruppe oder Organisation, die etwas tun kann.
Aspekt der Domäne	Der Aspekt eines Bereichs bezieht sich auf einen bestimmten Bereich des R4F Domäne, der durch das Simulationsmodell modelliert wird.
Betreuer	Der Agent, der für die Pflege und Aktualisierung der Ressource.
Brücke	Eine Brücke wird gebaut, um physische Objekte zu wie beispielsweise ein Gewässer, ein Tal oder eine Straße zu überqueren, mit dem Ziel den Übergang über das Hindernis zu erleichtern.
Datendienst	Eine Sammlung von Vorgängen, die den Zugriff auf einen oder mehrere Datensätze.
Datensatz	Eine Sammlung von Daten, veröffentlicht oder kuratiert von einem (mehreren) Agenten veröffentlicht oder kuratiert werden und in einer oder mehreren Darstellungen.
Datentyp	Gibt an, welche Art von Wert eine Variable hat. Dazu gehören Ganzzahlen (Integer), boolesche Werte (Boolean) und Zeichenketten (Strings).
Distribution	Eine spezifische Darstellung eines Datensatzes. Der Datensatz kann mehreren Serialisierungen verfügbar sein, die sich in verschiedenen unterscheiden können, einschließlich natürlicher Sprache, Medientyp oder Format.
Dokument	Eine Ressource, die die Dinge repräsentiert, die im weitesten Sinne Dokumenteßind (z. B. Dokumentation, Lizenzdokumente).
Ersteller	Der verantwortliche Agent für die Gestaltung und Entwicklung der Ressource
Gleis	Ein Gleis ist die Struktur einer Eisenbahn, die aus Schienen, Befestigungen, Schwellen und Schotter besteht. Es ermöglicht die Fortbewegung von Zügen, indem es eine verlässliche Oberfläche bietet, auf der die Räder rollen können.

... Fortsetzung auf nächster Seite

... Fortsetzung

Term	Definition in natürlicher Sprache
Herausgeber	Der Agent, der für die Bereitstellung der Ressource (Simulations Modell, Datensatz oder Datendienst) verfügbar zu machen.
Infrastruktur	Infrastruktur ist die Gesamtheit der Einrichtungen und Systeme, die den den Betrieb des Eisenbahnsystems unterstützen. Dazu gehören die Gleise, Brücken, Tunnel, Signalsysteme und andere Strukturen und Einrichtungen Einrichtungen, die für einen sicheren und effizienten Zugverkehr notwendig von Zügen erforderlich sind.
Kausalität	Die Kausalität einer Variablen bezieht sich auf die Beziehung zwischen der Variable und dem Simulationsmodell (z. B. Input- oder Output Variable).
Kontakt	Relevante Kontaktinformationen für die katalogisierte Ressource. Die Verwendung von von vCard[1] wird empfohlen
Maßeinheit	Eine Maßeinheit oder Einheit ist ein bestimmter Mengenwert, der als Maßstab für die Messung anderer Größen der gleichen gleicher Art (im Allgemeinen von gleicher Größe) gewählt wurde.
Medien Typ	Das Dateiformat der Ressource. Die Verwendung der IANA Media-Typen[2] wird empfohlen.
Schienenfahrzeug	Schienenfahrzeuge bezieht sich auf Eisenbahnfahrzeuge, sowohl auf angetriebene und nicht angetriebene Fahrzeuge: z. B. Lokomotiven, Güter- und Personenwagen (oder Reisezugwagen) und andere Fahrzeuge.
Simulationsmodell	Ein Simulationsmodell ist eine mathematische Darstellung eines realen Prozesses oder Systems im Zeitverlauf. Es ist eine vereinfachte abstrakte Ansicht der komplexen Realität. Es kann verwendet werden zur Berechnung das erwartete Verhalten unter bestimmten Bedingungen zu berechnen.
Tunnel	Ein Tunnel ist ein künstlicher Durchgang, der unter Bergen, Gewässern oder anderen Hindernissen hindurchführt führt.

... Fortsetzung auf nächster Seite

[1] https://www.w3.org/TR/2014/NOTE-vcard-rdf-20140522/
[2] https://www.iana.org/assignments/media-types/media-types.xhtml

. . . Fortsetzung

Term	Definition in natural language
Variable	Eine Variable stellt einen bestimmten Parameter oder Faktor eines Simulationsmodells dar.
Variablen Auflistung	Eine Variablen Auflistung ist eine Sammlung von Variablen, die eine bestimmte Reihe von Werten darstellen (z. B. Trägheitstensor).
Variablen Gruppe	Eine Variablengruppe ist eine Sammlung von Variablen, die ein gemeinsames Merkmal teilen (z. B. Kausalität, Einheit oder Aspekt eines Bereichs).
Variablen Vektor	Ein Variablen Vektor ist eine Sammlung von Variablen, die einen Vektor darstellen (z. B. Kraft, Beschleunigung).
Version	Der Versionsindikator, normalerweise ein Bezeichner, einer Ressource.
Verbindung	Die Verbindung zwischen Variablen bezieht sich auf die Beziehung zwischen zwei oder mehr Variablen.

B.3. Concept Classification Tree

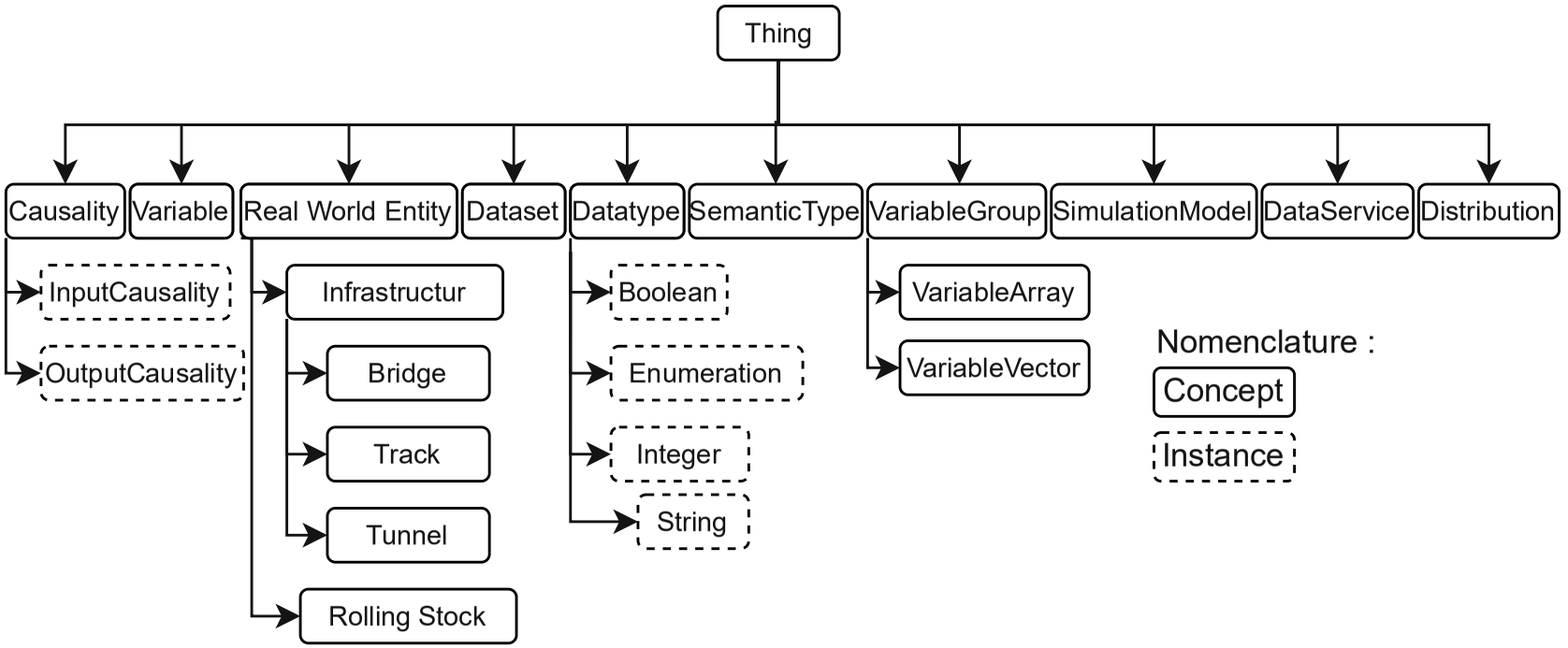


Figure B.1.: Concept Classification Tree.

B.4. Concept Dictionary

Table B.4.: Concept dictionary

Concept Name english	Concept Name german	Instance attributes	Relations
Causality	Kausalität	-	-
DataService	Datendienst	title, accessRights, endpointDescription	servesDataset, publisher, creator
Dataset	Datensatz	title, description, issued, modified, accessRights	distribution, publisher, creator, hasVariableGroup--Member, hasCausality
Datatype	Datentyp	-	-
Distribution	Distribution	title, accessURL, downloadURL, checksumValue	accessService, mediaType previousVersion, nextVersion, lastVersion
RealWorldEntity	RealWorldEntity	title, description	-
SemanticType	SemantischerTyp	-	-
SimulationModel	Simulationsmodell	title, description, created, modified	hasVariable, describes, previousVersion, nextVersion, lastVersion, publisher, creator, maintainer, supportingDocument, isLinkedTo

... continued on next page

... continued from previous page

Concept Name english	Concept Name german	Instance attributes	Relations
Variable	Variable	title, description, variability	hasCausality, unit, hasQuantityKind, hasSemanticType, hasDatatype, isVariableOf, isConnectedWith, isVariableMemberOf,
VariableGroup	VariablenGruppe	title, description	hasCausality, isVariable-GroupMember, hasVariableMember

B.5. Binary Relation Tables

Table B.5.: Binary Relation Tables

Relation name	Source concept	Target concept	Inverse relation
accessService	Distribution	DataService	-
contactPoint	Agen	Vcard	-
describes	SimulationModel	RealWorldEntity	-
distribution	Dataset	Distribution	-
hasCausality	VariableGroup, Variable	Causality	-
hasDatatype	Variable	Datatyp	-
hasSemanticType	Variable	SemanticType	-
hasVariable	SimulationModel	Variable	isVariableOf
isConnectedWith	Variable	Variable	-
isLinkedTo	SimulationModel	SimulationModel	-
isVariableGroup-MemberOf	VariableGroup	Dataset	hasVariableGroup-Member
isVariableMember	Variable	VariableGroup	hasVariableMember
lastVersion	SimulationModel, Distribution	SimulationModel, Distribution	-
maintainter	SimulationModel	Agent	-
mediaType	Document, Distribution	FileFormat	-
nextVersion	SimulationModel, Distribution	SimulationModel, Distribution	-
previousVersion	SimulationModel, Distribution	SimulationModel, Distribution	-
servesDataset	DataService	Dataset	-
supportingDocument	SimulationMode	Document	-

B.6. Instance Attribute Tables

Table B.6.: Attributes Table English

Instance attribute name	Datatype	Reused Ontology	Description
accessRights	xsd:string	dcterms	Information about who can access the resource or an indication of its security status.
accessURL	xsd:string	-	A URL of the resource that provides access to a distribution of the dataset, e.g., a landing page.
checksumValue	xsd:string	spdx	The checksumValue property provides a mechanism that can be used to verify that the contents of a file have not changed by provides a encoded digest value.
created	xsd:date	dcterms	Date of creation of the resource.
description	xsd:string	dcterms	An account of the resource.
downloadURL	xsd:string	-	The URL of the downloadable file in a given format.
endpointDescription	xsd:string	-	A description of the services available via the end-points, including their operations, parameters etc.
issued	xsd:date	dcterms	Date of formal issuance of the resource.
modified	xsd:date	dcterms	Date on which the resource was changed.
name	xsd:string	foaf	A name for an agent.
title	xsd:string	dcterms	A name given to the resource.
variability	xsd:string	-	Defines the time dependency of the variable, i.e., weather a variable can change its value (e.g., fixed, constant, continuous...).
version	xsd:float	-	Indicates the resources version

Table B.7.: Attributes Table German

Instance Attribut Name	**Datentyp**	**Wieder-verwendete Ontology**	**Beschreibung**
accessRights	xsd:string	dcterms	Informationen darüber, wer auf die Ressource zugreifen kann, oder ein Hinweis auf ihren Sicherheitsstatus.
accessURL	xsd:string	-	Eine URL der Ressource, die Zugang zu einer Distribution des Datensatzes bietet, z. B. eine Zielseite.
checksumValue	xsd:string	spdx	Die Eigenschaft bietet einen Mechanismus, mit dem überprüft werden kann, ob sich der Inhalt einer Datei nicht geändert hat, indem ein kodierter Prüfwert bereitgestellt wird.
created	xsd:date	dcterms	Datum der Erstellung der Ressource.
description	xsd:string	dcterms	Ein Bericht über die Ressource.
downloadURL	xsd:string	-	Die URL der herunterladbaren Datei in einem bestimmten Format.
endpointDescription	xsd:string	-	Eine Beschreibung der über die Endpunkte verfügbaren Dienste, einschließlich ihrer Operationen, Parameter usw.
issued	xsd:date	dcterms	Datum an dem die Resource herausgegeben wurde.
modified	xsd:date	dcterms	Datum an dem die Ressource geändert wurde.
name	xsd:string	foaf	Der Name des Agents
title	xsd:string	dcterms	Ein Name der Ressource.
variability	xsd:string	-	Definiert die Zeitabhängigkeit der Variablen, d. h., ob eine Variable ihren Wert ändern kann (z. B. fest, konstant, kontinuierlich...).
version	xsd:float	-	Gibt die Version der Ressourcen an.